UNIVERSIDAD NACIONAL DE LA MATANZA

SECRETARÍA DE POSGRADO

MAESTRÍA EN INFORMÁTICA

Título de Tesis:

Aplicación de Tecnologías Adaptativas -

Caso de Estudio: Sistemas Operativos Reconfigurables

Autor: Martin, Sergio Miguel

Director: Neto, João José

Buenos Aires, Febrero, 2013

ÍNDICE

1. INTRODUCCIÓN

1.1. DESCRIPCIÓN DEL PROBLEMA

Los sistemas operativos han sido desarrollados y utilizados durante décadas para abstraer de la complejidad del hardware existente tanto a los usuarios finales de las computadoras, como a los programadores de aplicaciones. Mediante ellos, la computadora y sus componentes pueden ser vistos como recursos administrables que los usuarios y sus aplicaciones pueden solicitar y utilizar. Los recursos de una computadora que deben ser administrados por el núcleo – o kernel, en inglés – del sistema operativo incluyen, aunque no se limitan a: el uso de los procesadores, la asignación de la memoria disponible entre los diferentes procesadores, y los permisos para operaciones de entrada y salida a dispositivos físicos o virtuales.

Ya sean aplicaciones de uso general o particular, o interfaces de usuario, las unidades de trabajo que hacen uso de los recursos administrados por el sistema operativo se denominan **procesos**. Para funcionar correctamente, un proceso necesita: poder tomar el control de un procesador, al menos por limitadas ráfagas de tiempo; contar con un espacio de memoria asignado para sus datos y código ejecutable; y una interfaz para solicitudes de entrada y salida. Dado que ninguno de estos recursos es ilimitado en ninguna computadora, ni tampoco lo serán en un futuro cercano, dada la perspectiva actual, los procesos se hallan en constante condición de competencia por ellos.

Los primeros sistemas operativos, por ejemplo MS-DOS, sólo permitían la ejecución de un único proceso a la vez, por lo que su función sólo se limitaba a inicializar la computadora, proveer una interfaz de línea de comandos, y ofrecer algunas funciones adicionales que facilitaban la programación de aplicaciones. El proceso ejecutado tomaba

completa posesión del procesador, la memoria, y los dispositivos de entrada y salida, por lo que el sistema operativo no cumplía ninguna función administrativa.

A partir del desarrollo de los primeros sistemas operativos tipo UNIX que permitían, no solo múltiples procesos, sino también múltiples usuarios utilizando la misma computadora al mismo tiempo, surgieron nuevos desafíos. La cuestión de qué usuarios o qué procesos debían tener un mayor tiempo de uso de los procesadores, o mayores asignaciones de memoria disponibles fue abordada de diferente manera según el criterio de cada uno de los diseñadores de sistemas operativos. Las estrategias de administración de recursos varían entre los diferentes diseños, y cada empresa o usuario podría elegir cuál sistema operativo implementar en sus computadoras o centros de cómputos según sus necesidades particulares.

Tanto la estrategia de planificación de procesos para el uso de los procesadores, o la de administración del uso de memoria, se consideran **aspectos** del sistema operativo. También se consideran aspectos del sistema operativo: las políticas de seguridad, las políticas de administración de energía, la inicialización de dispositivos físicos, las interfaces de comunicación entre procesos y sistema operativo, los sistemas de archivos soportados, entre otros. Cada uno de estos aspectos puede ser implementado de una manera diferente para cada sistema operativo. A cada una de estas implementaciones particulares para un aspecto se la denomina modo de funcionamiento, o simplemente **modo**.

El diseño de un sistema operativo depende en gran medida de las necesidades específicas del mercado al que se destinará su distribución, del entorno – hardware, peopleware, orgware, y otros componentes software – en el que funcionará, y de los estándares que se deban respetar. Dado que cada uno de los aspectos a tener en cuenta puede contar con decenas de posibles modos de implementación diferentes, encontrar la combinación correcta para las necesidades específicas a satisfacer es, por lo tanto, una tarea

que requiere un análisis exhaustivo. Una mala decisión en al menos uno de estos aspectos puede causar el fracaso en los objetivos finales – comerciales o funcionales – del sistema operativo [Vaughan-Nichols - 2009].

La elección de una de estas combinaciones, es decir, un modo de funcionamiento particular para cada uno de estos aspectos, da por resultado la configuración a bajo nivel del sistema operativo a desarrollar —nos referiremos a esto simplemente como **configuración** del sistema operativo—. A partir de esto, una de las definiciones centrales utilizadas en este trabajo es la siguiente:

*Un sistema operativo es considerado **reconfigurable** cuando puede cambiar su configuración durante su ejecución, aunque sea tan solo en el modo de funcionamiento de un solo aspecto, sin necesidad de ser re-compilado.*

Salvo contadas excepciones [Veitch - 1996], no existe necesidad de desarrollar más de una única configuración para un sistema operativo, ya que esto demandará esfuerzo y tiempo de desarrollo adicional, no solo para cada configuración, sino también para el mecanismo que permita alternar entre ellas sin necesidad de re-compilar y re-distribuir el sistema. Sin embargo, a pesar de que la mayoría de los requerimientos comerciales y hogareños pueden ser satisfechos mediante un sistema operativo no reconfigurable convencional, existe un gran potencial en el desarrollo de sistemas operativos reconfigurables. Las posibles aplicaciones pueden ser:

- Uso personal u hogareño. Un sistema operativo reconfigurable permite cambiar su configuración para responder más eficientemente a las necesidades de cada usuario particular. Por ejemplo, un usuario podría utilizar una computadora exclusivamente para el uso de aplicaciones de uso intensivo del procesador

multimedia o de entretenimiento. Otro usuario, podría tener un uso más frecuente de aplicaciones de escritorio que requieran un uso disperso del procesador. El sistema podría mejorar el rendimiento para cada comportamiento variando su configuración según qué usuario este usándolo.

- Uso comercial o de procesamiento intensivo. Un sistema operativo reconfigurable podría distribuir inteligentemente el trabajo sobre plataformas de tipo clúster, distribuidos alrededor de varios centros de cómputo de manera de optimizar la eficiencia en el cálculo o el consumo de energía.

- Uso didáctico. Desde el punto de vista didáctico, el factor distintivo del enfoque de utilizar sistemas operativos reconfigurables respecto a utilizar sistemas estáticos pre-compilados, es que permite a docentes y a alumnos efectuar un análisis de tipo *ceteris paribus* de los cambios efectuados sobre la configuración [Martin – 2012]; todo, salvo lo que se cambia, se mantiene inalterado, aislando sus consecuencias y, por lo tanto, dando una idea más clara de su funcionamiento.

El principal problema que se presenta al momento de diseñar un sistema operativo reconfigurable es la cuestión de quién va a decidir qué cambios en la configuración realizar y cuándo realizarlos:

Si se espera que los **usuarios finales** sean los que decidan qué configuración utilizar, se corre el riesgo de que éstos no cuenten con el conocimiento suficiente sobre qué cambios realizar, o cuáles pueden ser sus consecuencias. Por lo tanto, sus decisiones podrían incluso empeorar, o hasta inutilizar finalmente al funcionamiento del sistema operativo respecto a sus propias necesidades. En principio, se requeriría de un usuario o administrador avezado que

conozca los detalles del funcionamiento particular del sistema operativo reconfigurable para poder aprovechar su potencial. Sin embargo, aún así, esto podría resultar inconveniente ya que muchas cuestiones de configuración importantes pueden requerir un tiempo de decisión muy acotado, fuera del alcance aún del más capacitado de los administradores.

Si se espera que sea sólo el **sistema operativo** el que tome las decisiones sobre los cambios en su configuración, se contará con la información correcta – determinada por sus propios diseñadores – y el poder de procesamiento para dar respuesta aún a las cuestiones más urgentes. Este enfoque ofrece un gran potencial de eficiencia y adaptabilidad, ya que cuenta con las ventajas propias de los sistemas operativos reconfigurables ya expuestas, y no requiere de la intervención de un usuario experto.

El principal problema de un enfoque sin intervención del usuario, es que si el sistema permanece cerrado a las respuestas de los procesos sobre lo acertada o errada que fueron sus decisiones, no tendrá parámetros para definir cuáles son los cambios requeridos para cada situación [Veitich - 1998]. Por esta razón, la literatura científica y los desarrollos hechos hasta el momento, sólo han contemplado la realización de sistemas operativos reconfigurables con intervención explícita de los usuarios y sus aplicaciones.

En este trabajo proponemos que dicho problema de falta de información, en el caso de que sea el sistema operativo tome las decisiones sobre su configuración, puede ser solucionado si se utilizan tecnologías adaptativas como método de toma de decisión. Mediante su uso, será posible aproximarse a la mejor configuración en todo momento, tomando en cuenta indicadores y métricas de eficiencia ya disponibles por el kernel, sin necesidad intervención explícita del usuario y sus aplicaciones.

1.2. PROPUESTA DE SOLUCIÓN

En el presente trabajo abordamos una posible solución basada en la utilización de tecnologías adaptativas – específicamente: tablas de decisión adaptativas – para poder otorgarle a un sistema operativo reconfigurable la capacidad de tomar decisiones que logren adaptarse a diferentes situaciones de uso, y que, al mismo tiempo, mantenga una alta eficiencia en la ejecución de los procesos de usuario.

Para la utilización de métodos de toma de decisión adaptativos, es preciso que el sistema operativo reconfigurable no sólo sea capaz de tomar decisiones por su cuenta, sino de registrar las acciones del usuario, para aprender de ellas y mejorar el servicio provisto.

Los **dispositivos adaptativos**, tales como las **tablas de decisión adaptativas** propuestas originalmente por [Neto – 2001], se han utilizado en ambientes donde el aprendizaje en base a las acciones realizadas por los usuarios debe determinar las acciones que un sistema debe realizar. Ejemplos de esto son los sistemas de recomendación [Cozman – 2012], o el procesamiento de lenguaje natural [Contier – 2012]. Dichos dispositivos no sólo son capaces de actuar de manera diferente, en diferentes momentos, ante el mismo estímulo, dada una u otra condición, sino que también son capaces de modificar esas mismas reglas de comportamiento, basándose en información externa actualizada. Esta capacidad los diferencia de dispositivos *adaptables*[1], que tan sólo son capaces de reaccionar de diferente manera según condiciones preestablecidas, pero no son capaces de modificar su propio comportamiento.

La capacidad de modificar el propio comportamiento es la característica distintiva de lo estudiado en las tecnologías adaptativas. Dicha capacidad puede presentar un gran

[1] Aquí se hace una distinción explícita entre un dispositivo *adaptativo* y uno *adaptable* necesaria para diferenciar el comportamiento de cada uno en el marco de esta tesis. Esto se debe a la falta de una traducción del término inglés "*adaptive"* utilizado en la bibliografía del tema. En el uso normal del idioma español, ambos términos, *adaptativo* y *adaptable*, son considerados sinónimos.

potencial para satisfacer las necesidades – acaso cambiantes – de los usuarios o, de igual manera, de usuarios cambiantes de un sistema operativo, abstrayéndolos de la complejidad inherente de la configuración interna.

Dadas varias respuestas negativas de un proceso ante una acción tomada por un conjunto de reglas del tipo evento/condiciones/acciones, el dispositivo adaptativo aplicado al sistema operativo reconfigurable puede variar dicho conjunto, alterando futuras respuestas, en beneficio de la eficiencia de ejecución.

Nos proponemos entonces investigar cómo es que un dispositivo adaptativo puede detectar cambios en las necesidades de los usuarios, cómo modificar sus reglas de comportamiento, y qué modificaciones realizar sobre el sistema operativo reconfigurable para que ellas tengan efecto.

Más en general, dado que este trabajo utiliza sistemas operativos reconfigurables como caso de estudio, será posible afirmar que la aplicación de tecnologías adaptativas son una alternativa potencialmente eficiente en todo tipo de problemas complejos y cambiantes.

Los posibles beneficiarios de los resultados de esta investigación pueden ser tanto usuarios de computadoras hogareñas y personales, que podrían trabajar o disfrutar de aplicaciones de entretenimiento o juegos de manera más eficiente, según la necesidad de cada uno; pero también podrían ser aprovechados por empresas que utilicen servidores de producción, en los que, el sistema operativo deba adaptar su comportamiento, permitiendo que los trabajos que puedan ser ejecutados en diferentes ambientes precisen menor tiempo de ejecución.

1.3. OBJETIVOS

- Verificar que es posible aplicar tecnologías adaptativas como mecanismos de toma de decisión para determinar la configuración de un sistema operativo según el comportamiento del usuario y sus aplicaciones.
- Demostrar que las tecnologías adaptativas ofrecen una alternativa eficiente para problemas de toma de decisión complejos.

1.4. HIPÓTESIS

H1) Es posible definir una configuración para un sistema operativo en base al uso de sus usuarios y aplicaciones, sin que estos provean información de retroalimentación explícita.

H2) Es posible definir métricas que permitan obtener el grado de adecuación de una configuración para un escenario de uso de manera cuantificable.

H3) La configuración indicada en (H1) puede ser determinada por un dispositivo adaptativo.

H4) Es posible realizar la reconfiguración un sistema operativo en tiempo de ejecución para reflejar la configuración obtenida por (H3).

2. ANTECEDENTES

Dado que ésta es una investigación que combina dos disciplinas, se hará necesario analizar los antecedentes respecto cada una por separado. Es preciso considerar, entonces, como antecedentes: (i) los desarrollos existentes de sistemas operativos reconfigurables, (ii) en particular, el sistema operativo SODIUM sobre el que se efectuará la investigación, y (iii) la investigación existente sobre tecnologías adaptativas.

2.1.1. SISTEMAS OPERATIVOS RECONFIGURABLES

2.1.1.1. SYNTHETIX

Synthetix [Cowan – 1996] fue uno de los primeros proyectos de desarrollo de un sistema operativo que pudiera adaptarse a circunstancias particulares para obtener algún beneficio funcional o de rendimiento. Sus desarrolladores plantearon que, si bien los sistemas operativos de tipo micro-kernel ya habían abordado esta posibilidad, fallaban respecto a la pérdida de eficiencia debido a los controles de seguridad y comunicación entre sus componentes.

Basándose en datos obtenidos en ambientes de producción, por ejemplo, el caso en que la política de paginación habitual en la mayoría de los sistemas operativos es ineficiente para aplicaciones de bases de datos [Stonebraker – 1981], planteaban que las optimizaciones generales podían ser, incluso, perjudiciales para ciertos tipos de aplicaciones, y que deberían existir mecanismos de especialización.

Synthetix utiliza un enfoque sistemático para la reconfiguración dinámica del sistema operativo. Utilizando propiedades de la configuración a las que denomina invariantes y cuasi-

invariantes, realiza cambios de comportamiento individualmente para cada aplicación a partir de la información de retroalimentación recibida por estas. Las pruebas realizadas, disponibles en [Massalin – 1990], indicaron que este método puede obtener aumentos de rendimiento de un 560%.

2.1.1.2. SPIN

El sistema operativo SPIN [Bershad – 1995] fue diseñado para proveer una infraestructura extensible, en conjunto con un núcleo de servicios extensibles, que le permiten a las aplicaciones cambiar la configuración de la interfaz del sistema operativo. Las extensiones permiten que la aplicación especialice al sistema operativo subyacente para lograr un nivel particular de rendimiento y funcionalidad.

Los desarrolladores de SPIN se basaron en la investigación realizada sobre aplicaciones paralelas de alto rendimiento [von Eicken – 1992], en las que se obtuvo que las implementaciones de protocolos de redes de propósito general fueran inadecuadas para ellas logrando un rendimiento menor al que se puede lograr utilizando un protocolo especializado.

SPIN utiliza mecanismos de lenguaje y de tiempo de vinculación para exportar interfaces de granularidad fina a los servicios de los sistemas operativos. Los programadores pueden desarrollar extensiones particulares de los servicios en un lenguaje fuertemente tipado, y vincularlas dinámicamente al kernel para ser utilizada por sus aplicaciones.

2.1.1.3. KEA

Kea [Veitich – 1996] fue desarrollado como un sistema operativo de tipo micro-kernel diseñado para una máxima flexibilidad y rendimiento en las áreas de extensibilidad de kernel y aplicaciones, y reconfiguración dinámica.

De manera similar a SPIN, y a diferencia de Syntethix, el kernel de Kea no cambia su configuración en base a la retroalimentación recibida por las aplicaciones, sino que provee una interfaz por la cual las aplicaciones pueden determinar de manera explícita qué servicios específicos utilizar. Esto le permite ser extendido y configurado por cada aplicación en particular. Kea no efectúa ningún tipo de toma de decisiones durante el proceso.

Kea provee a las aplicaciones la posibilidad de comunicación entre dominio proceso/servicio mediante mensajes que son pasados a través de portales [Veitich - 1998]. Los portales definen hacia qué servicio se dirigirán las solicitudes de una aplicación. La aplicación puede, entonces, reasignar el portal por defecto para un tipo de servicio o aspecto del sistema operativo a diferentes implementaciones o modos, especializando los pedidos a sus propias necesidades. Kea provee un único syscall para la programación de aplicaciones, que es el que permite crear y apuntar los portales a los diferentes servicios.

La posibilidad de aprovechar al máximo la reconfigurabilidad explícita conseguida por Kea depende del programador de las aplicaciones. Incluso, un programador puede extender los servicios provistos por el kernel.

2.1.1.4. APERTOS

Apertos es un sistema operativo completamente basado en objetos desarrollado por SONY [Lea – 1995], y utiliza reflexión[2] y meta-objetos para permitir a los programadores adaptar a los aspectos del sistema operativo para hacia las necesidades específicas de la aplicación.

Apertos utiliza meta-objetos para describir la interfaz y funcionalidad que un objeto de nivel básico puede soportar. Todos los meta-objetos en Apertos se ubican en lo que se denomina un contexto, que resguarda el estado de los registros de ejecución. Efectuando

[2] Traducción directa de *reflection*, capacidad de un sistema de examinar la estructura y comportamiento de un objeto en tiempo de ejecución.

cambios en dichos meta-objetos, se provee la capacidad de alterar la forma en que estos se implementan. Por ejemplo, el meta-objeto *execm* que reside en el contexto *mCore* para cada aplicación en particular, sirve como interfaz para las funciones del uso de la CPU y los espacios de memoria.

El programador puede acceder a los servicios del sistema operativo a través de llamadas a las instancias por defecto de los métodos de los meta-objetos, o alterarlos asignándoles métodos propios.

2.1.1.5. ENTORNO VVM

El proyecto VVM [Piumarta – 2000] no se trató del desarrollo de un sistema operativo sino, más bien, de un ambiente de ejecución virtual dedicado a la extensibilidad y reconfigurabilidad de sus servicios. Está basado sobre el desarrollo de una plataforma de ejecución independiente del hardware llamada máquina virtual *virtual* [Folliot – 1998].

Sus creadores proponen un único entorno que soporte aplicaciones (o sus componentes) programados en cualquier lenguaje. Las aplicaciones son tipadas con un modelo de ejecución apropiado, y se la hace corresponder con una descripción de una máquina virtual llamada VMlet. Los VMlets son cargados bajo demanda, cuando un nuevo tipo de aplicación es encontrado.

Una aplicación puede configurar su VMlet de ejecución cambiando la forma en que la máquina virtual VVM le provee sus servicios, por lo que una misma aplicación puede ser ejecutada en diferentes VMlets con diferentes configuraciones. La decisión sobre la configuración con la que contará la aplicación y su VMlet está a cargo exclusivamente del programador de la aplicación.

2.1.1.6. ARQUITECTURA THINK

Al igual que VVM, THINK no es un kernel – de hecho, THINK es un acrónimo para "THink Is Not a Kernel" [Fassino – 2002]. THINK es una arquitectura formulada para el desarrollo de nuevos kernels en la que los recursos de hardware y software son estructurados como componentes. Los componentes se pueden comunicar con otros componentes mediante enlaces flexibles. Los enlaces mismos pueden ser ensamblajes de componentes, y pueden tomar muchas formas.

El enfoque propuesto por THINK [Senart – 2002] no impone abstracciones de sistema como la administración de memoria, sino que, en cambio, provee interfaces que hacen visibles a los recursos del sistema. Los sistemas operativos desarrollados utilizando la arquitectura THINK, no resultan portables, ya que no presentan funciones desarrolladas en alto nivel, pero ganan en flexibilidad y modularidad.

Para lograr reconfigurar el sistema operativo, los programadores pueden cargar y descargar componentes, y reasignar los vínculos entre ellos dinámicamente, incluso varias veces durante la ejecución de una aplicación. Tal como en el caso del entorno VVM, la decisión de qué cambios efectuar y cuando efectuarlos recae exclusivamente en el programador.

2.1.1.7. RECONFIGURABILIDAD BASADA EN COMPONENTES

El enfoque más reciente para el diseño de sistemas operativos reconfigurables planteado por Polakovic y Stefani [Polakovic – 2007] se basa en la arquitectura THINK que además utiliza Fractal, un modelo de componentes jerárquico y reflexivo.

Dicho enfoque permite que los mecanismos de reconfiguración – a diferencia de, por ejemplo, el sistema operativo SPIN – se basen por completo en software, sin afectar a la estructura de kernel. Utilizando reflexión, los programadores pueden desarrollar componentes desarrollados en cualquier lenguaje orientado a objetos, lo cual permite ofrecer interfaces independientes que pueden ser reutilizables por cualquier otra aplicación o componente.

2.1.2. SODIUM

SODIUM[3] es un proyecto de desarrollo conjunto entre alumnos y profesores de la materia de Sistemas Operativos Avanzados de la carrera de Ingeniería en Informática de la Universidad Nacional de La Matanza de un sistema operativo didáctico y reconfigurable [Casas – 2007]. Se trata de un sistema operativo de multiprogramación basado en la arquitectura Intel IA32.

Si bien SODIUM pudo haber formado parte de la anterior lista de sistemas operativos reconfigurables, es preciso realizar un análisis más exhaustivo del mismo como antecedente a esta investigación ya que fue utilizado como base para los desarrollos prácticos aquí incluidos, y que la investigación realizada sobre la reconfigurabilidad de sus aspectos y la toma de decisión sobre los cambios en sus modos de ejecución representan el antecedente más inmediato al de esta investigación.

El proyecto fue iniciado en el 2005 con el objetivo de darles una oportunidad a los alumnos de la cátedra, no solo de aprender los conceptos teóricos de la misma, sino también de poder involucrarse de manera práctica en el desarrollo de un sistema operativo. SODIUM fue evolucionando a partir de la entrega de trabajos prácticos consistentes en módulos

[3] Sistema Operativo del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza

agregables, por ejemplo: de administración de procesos o controladores de gestión de energía [Casas – 2011].

Los objetivos que fueron planteados para su realización fueron [Ryckeboer – 2008]:

- Apoyar a la enseñanza de los contenidos de la cátedra a través de la visualización de las tablas internas del sistema, la disponibilidad de información sobre los eventos que ocurren internamente en su kernel, y la posibilidad de poder cambiar la configuración del sistema operativo sin detener la ejecución.

- Permitir a los alumnos involucrarse en el desarrollo mismo del sistema operativo, permitiéndoles adquirir el conocimiento de los detalles de cada aspecto a desarrollar.

- Realizar transferencia de conocimiento a través de interfaces para cada aspecto, permitiendo que el alumno desarrolle modos o implementaciones alternativas a las existentes.

- Promover la ayuda mutua y el intercambio de información entre alumnos de diferentes años a través de los documentos elaborados durante la investigación y los medios de comunicación propuestos por los docentes.

2.1.2.1. RECONFIGURABILIDAD EN SODIUM

El primero de los objetivos antedichos definió, desde el comienzo del proyecto, que SODIUM debería permitir varios modos de funcionamiento para cada aspecto. Esto se reflejó en la

consiguiente investigación y desarrollo de diferentes modos de funcionamiento para varios aspectos del sistema operativo, entre ellos, la administración de memoria [Casas – 2008]. Sin embargo, hasta ese punto, la posibilidad de reconfiguración de los aspectos de SODIUM fue realizada *ad hoc*, es decir, que no solo el mecanismo para el cambio de modo de un aspecto, sino también los momentos en los era posible realizar el cambio fueron definidos separadamente y de manera diferente en cada caso.

Este problema fue abordado al establecer una metodología para el desarrollo de un sistema operativo reconfigurable [Martin – 2012] en la que se establece y unifica cómo diseñar aspectos que puedan ser cambiados de modo, y cómo clasificarlos.

Establece una escala de 4 niveles para determinar en qué momento se permitirá el cambio de modo de funcionamiento del aspecto: en tiempo de compilación, en tiempo de arranque, en tiempo de ejecución limitada, o en tiempo de ejecución pura. También define un modelo basado en autómatas finitos para evaluar las transiciones entre modos en tiempo de ejecución y sus condiciones, las pautas generales para la interfaz con el usuario, y los pasos para realizar el diseño.

Utilizando esta metodología, se pueden enmarcar los aspectos reconfigurables de SODIUM bajo un diseño unificado. La existencia de un diseño ordenado de sus mecanismos e interfaces es esencial para poder considerar a SODIUM formalmente como un sistema operativo reconfigurable.

2.1.2.2. TECNOLOGÍAS ADAPTATIVAS EN SODIUM

La antedicha metodología implementada en SODIUM establece que la decisión sobre qué cambios de configuración realizar recae sobre los usuarios. Esto resulta conveniente teniendo en cuenta que SODIUM es un sistema operativo didáctico que está orientado a alumnos y

docentes que deseen ejecutar lotes de prueba sobre diferentes configuraciones para comparar sus resultados sin tener que recompilar o reiniciar el sistema. Sin embargo, en la investigación también se contempló la posibilidad de que la toma de decisión pueda recaer también sobre el kernel.

La propuesta para que el kernel de SODIUM tome las decisiones sobre los cambios de configuración incluye el uso de **tablas de decisión**. Se contempla entonces, de una manera tabular, las transiciones entre modos de cada aspecto y las condiciones y eventos para cada una de ellas. Dado que dichas tablas estarán predeterminadas para uso general, y aunque incluso pudieran ser modificadas por los usuarios, aún seguirían sin poseer la capacidad adaptativa requerida para aprender de las acciones de estos últimos.

Se determinó entonces que el uso de tablas de decisión *convencionales* no sería de utilidad ni práctica, ni tampoco didáctica. Continuando la investigación, se encontró que el uso de tecnologías adaptativas, tales como las tablas de decisión *adaptativas* [Pedrazzi – 2005] podrían solucionar ambos inconvenientes al mismo tiempo.

Desde el punto de vista práctico, podría probarse que el sistema operativo puede mejorar su funcionamiento en base al comportamiento de los usuarios y las aplicaciones, sin necesidad de una **retroalimentación explícita**. Este trabajo está enfocado a probar esta última hipótesis.

Desde el punto de vista didáctico se planteó la hipótesis de que no sólo las decisiones tomadas por las tablas de decisión adaptativas, sino que también los cambios efectuados en base al uso, les podrían dar a los alumnos una mejor idea de qué criterios son tomados en cuenta al intentar optimizar el sistema a través de un cambio de configuración. Si bien esta hipótesis no forma parte de esta investigación, puede ser la base para futuras investigaciones.

Se ha planteado una primera línea de investigación para la aplicación de tablas de decisión adaptativas, específicamente para la gestión de la energía en SODIUM [Casas – 2012].

2.1.3. TECNOLOGÍAS ADAPTATIVAS

El término *tecnología adaptativa* se refiere al uso de técnicas y dispositivos con capacidad de reaccionar a estímulos de entrada modificando autónomamente su propio comportamiento. Si bien se han formulado implícitamente desde hace tiempo, muchos dispositivos adaptativos, en particular, autómatas [Shutt – 1995] [Neto – 1994] y gramáticas [Burshteyn – 1990] han sido reportados en la literatura científica.

Se ha demostrado que los autómatas adaptativos son dispositivos con potencia equivalente a las máquinas de Turing [Rocha – 2000]. La generalidad resultante del modelo, su capacidad de aprendizaje derivada de su propiedad auto-modificante, y la facilidad de representación, lo hacen muy atractivo para la resolución de fenómenos complejos, en especial aplicados a los sistemas computacionales.

Si bien los autómatas adaptativos ya cuentan con una potencia similar a la de las máquinas de Turing para el reconocimiento de lenguajes, su campo de aplicación se encuentra mayormente dentro del desarrollo de compiladores y análisis sintáctico. Sumado a esto, la aplicación de los primeros dispositivos adaptativos sobre problemas más generales no contaba con una notación que se aproximara lo máximo posible a la formulación original de los problemas de los sistemas reales. Junto a otras razones, los formalismos auto-modificantes no se han utilizado ampliamente debido a la excesiva complejidad de su notación, lo que los hace difíciles de utilizar.

Los dispositivos adaptativos basados en reglas [Neto – 2001] fueron elaborados con el objetivo de obtener una formulación mucho más clara, intuitiva, y fácil de aprender, sin añadir complejidad a la lectura, escritura, o interpretación de sus notaciones. Dicha formulación permite obtener un dispositivo adaptativo a partir de cualquier dispositivo de toma de decisiones no-adaptativo basado en reglas. Un dispositivo no-adaptativo basado en reglas es cualquier máquina formal cuyo comportamiento dependa exclusivamente de un conjunto de reglas finito que relaciona el estado actual del dispositivo hacia un siguiente estado correspondiente.

El ejemplo más utilizado de dispositivos no-adaptativos basado en reglas son las tablas de decisión. Las tablas de decisión son herramientas ampliamente utilizadas entre los programadores de sistemas e ingenieros de software. Dichas tablas pueden ser vistas como dispositivos tabulares que contienen un conjunto de reglas representadas por condiciones y acciones correspondientes a ejecutar cuando esas condiciones se cumplen.

Las tablas de decisión adaptativas, pueden ser obtenidas fácilmente a partir de las convencionales sólo agregándoles nuevas filas que indiquen las acciones adaptativas a ejecutar antes, o después de las acciones especificadas por la tabla original [Tchemra – 2009]. Cuando las acciones adaptativas son ejecutadas, la tabla original sufre cambios en su conjunto de reglas, ya sea modificando las condiciones necesarias para las acciones existentes, o removiendo o agregando nuevas reglas.

La facilidad de aprendizaje de uso y aplicación de las tablas de decisión adaptativas para sistemas de toma de decisión que ya cuentan con una tabla de decisión convencional, ha abierto las puertas a la aplicación de tecnologías adaptativas sobre los más diversos rubros. Se han aplicado tecnologías adaptativas para la selección de proveedores [Tchemra – 2008], los sistemas de recomendación y marketing [Cereda – 2009], la administración de negocios

[Pedrazzi - 2005], el modelado de cursos para software didáctico [Dizeró – 2010], la búsqueda de supervivientes utilizando vehículos aéreos no tripulados [Chaves – 2012], entre muchos otros.

Los últimos avances en la definición de las tablas de decisión adaptativas han permitido definir un mecanismo formal por el cual éstas pueden producir aprendizaje incremental, acumulando experiencia dinámicamente [Stange – 2011]. También se ha propuesto una extensión para los problemas de toma de decisión con múltiples criterios [Tchemra – 2010].

En relación con los sistemas operativos reconfigurables, se ha demostrado que las tablas de decisión convencionales (no-adaptativas) pueden ser utilizadas para decidir qué configuración utilizar, para cada aspecto en particular, dado el nuevo conjunto de condiciones cumplidas en cada momento [Martin – 2012]. Dicha tabla estaría predeterminada al comienzo mismo de la ejecución del sistema operativo, posiblemente mediante un archivo de configuración, y podría ser cambiada explícitamente por el usuario en cualquier momento.

La evolución en SODIUM hacia el uso de tablas de decisión adaptativas es, por lo expuesto, el camino natural para permitir el aprendizaje en base al uso dado por los usuarios y las aplicaciones y, por lo tanto, el tomado en el desarrollo de esta investigación.

3. PLANTEAMIENTO DEL PROBLEMA

3.1. DESCRIPCIÓN DEL PROBLEMA

Hasta el momento, en el ámbito de los sistemas operativos reconfigurables, no se ha abordado de manera suficientemente completa la posibilidad de permitirle al sistema operativo tomar decisiones de manera autónoma, en base al uso que los usuarios y las aplicaciones le dan, utilizando mecanismos auto-modificantes para el aprendizaje.

De los sistemas operativos reconfigurables analizados, tanto Kea, como SPIN y Apertos permiten ser reconfigurados, pero únicamente de manera explícita por el programador. Si bien esto le da más herramientas al programador para que su aplicación ejecute sobre una configuración del sistema operativo adecuada a ella, también lo obliga a conocer los pormenores del funcionamiento de un kernel en general o, peor aún, en particular para cada uno. Esta investigación se diferencia de ellos en que parte de un enfoque diferente, en el que se sostiene que es posible construir un sistema operativo que se adapte a cada usuario o aplicación de programador sin obligarlos a conocer los pormenores de su funcionamiento, y aún así, conseguir resultados similares a los obtenidos con enfoques explícitos.

El sistema operativo Synthetix, en particular, es el que más cerca se encuentra de la presente investigación ya que no requiere que el programador o el usuario le indique explícitamente qué cambios realizar sobre la configuración. Sin embargo, sí provee un mecanismo por el cual recibir información de retroalimentación de parte de la aplicación que debe ser utilizado explícitamente. Este enfoque, si bien reduce el nivel de conocimiento que el programador de aplicaciones debe poseer sobre el sistema operativo, aún depende de su intervención para realizar una reconfiguración. La presente investigación establece un

enfoque que se diferencia del aplicado en Synthetix, en el que se plantea que dicha información de retroalimentación puede ser obtenida de manera implícita, es decir, a través del aprendizaje sobre el comportamiento de las aplicaciones y los usuarios en base a cambios de configuración.

Los resultados obtenidos en esta investigación y sus conclusiones permitirán conocer cuáles son las ventajas y desventajas de la reconfigurabilidad adaptativa y los posibles ámbitos de aplicación sobre los cuales puede ser utilizada. Tanto desarrolladores, como usuarios y programadores podrían beneficiarse del uso de este tipo de sistemas operativos en ciertos ámbitos de ejecución de alta variabilidad que demanden una rápida adaptación de su configuración, minimizando pérdidas en tiempo de ejecución, o incluso de información.

Un sistema operativo reconfigurable adaptativo, podría ser aplicado, también, para maximizar el rendimiento sobre plataformas muy complejas, en las que una configuración manual resulte dificultosa, costosa, o hasta imposible. Sistemas de computación distribuida administrados centralmente – tal como puede ser un GRID – poseen una alta complejidad en el manejo de sus recursos, y podrían ser optimizados automáticamente con enfoque adaptativo para su configuración.

3.2. DESCRIPCIÓN DE LA SOLUCIÓN

Durante este trabajo se implementarán 3 mecanismos de toma de decisión diferentes, cada una mejorando incremental la capacidad adaptativa de la anterior. El desarrollo de dichas soluciones se llevará a cabo de la siguiente manera:

Para demostrar que es posible determinar la configuración de un sistema operativo reconfigurable de manera autónoma, seleccionaremos uno de los aspectos de SODIUM, y describiremos su estructura interna y sus diferentes modos de implementación. Luego,

detallaremos los fundamentos que nos permitirán establecer los criterios y condiciones para la toma de decisión. Esto será expuesto en detalle en el [Capítulo 4].

Como primera solución, implementaremos inicialmente una tabla de decisión convencional que sirva para formalizar las condiciones iniciales, determinadas por datos empíricos, que sirvan para describir un comportamiento básico sobre el cual implementar luego los dispositivos adaptativos que permitan aprender y tomar decisiones más complejas de manera eficiente. La tabla de decisión que describe el comportamiento básico del aspecto seleccionado será obtenida al final del [Capítulo 5].

Como se ha mencionado, las tecnologías adaptativas cuentan con potencia equivalente a las máquinas de Turing. Esto significa que cualquier dispositivo equivalente – ya sea un programa de computadora, o una máquina de Turing – puede cumplir la misma función que planteamos en este trabajo. Sin embargo, utilizaremos dispositivos adaptativos en particular debido a que, como se verá en el transcurso de este trabajo, su formulación presenta una gran facilidad de ser implementada sobre los mecanismos de toma de decisión ya elaborados – en nuestro caso, una tabla de decisión convencional. El dispositivo adaptativo obtenido será una tabla de decisión adaptativa, y su obtención se detalla durante el [Capítulo 6].

La forma inicial de la tabla de decisión adaptativa obtenida proveerá la capacidad de generar nuevas decisiones para casos no contemplados previamente. Si bien el mecanismo que implementaremos para ello será muy simple, este sentará las bases para extensiones de decisión multi-criterio que resultarán en la obtención de una tabla de decisión adaptativa extendida. La formulación y aplicación de estas extensiones se explicarán en el [Capítulo 7].

Finalmente, pondremos a prueba los tres mecanismos de toma de decisión obtenidos para comparar el rendimiento de cada uno de ellos respecto a la eficiencia requerida por el

aspecto reconfigurable de SODIUM seleccionado. También se contrastarán sus resultados respecto al conjunto de decisiones ideal que se deberían haber tomado para una eficiencia completa. Esto nos permitirá determinar, finalmente, si las extensiones adaptativas le permiten al sistema operativo reconfigurable tomar decisiones que permitan lograr una eficiencia cercana a la ideal, sin intervención explícita del usuario ni sus aplicaciones. Las pruebas y la interpretación de sus resultados se podrán observar en el [Capítulo 8].

3.3. ALCANCE DE LA INVESTIGACIÓN

A partir de trabajo desarrollado en la presente investigación se habrá logrado:

i. Desarrollar las extensiones necesarias para que SODIUM cuente con un aspecto reconfigurable y adaptativo.

ii. Demostrar los beneficios y limitaciones de la aplicación de tecnologías adaptativas en sistemas operativos reconfigurables.

iii. Sentar las bases de conocimiento para nuevas investigaciones que planteen la aplicación de tecnologías adaptativas sobre otros sistemas operativos de uso comercial, académico, o de producción.

Lo obtenido en (i) podrá ser utilizado como un complemento para la enseñanza de reconfigurabilidad y mecanismos de toma de decisiones por alumnos y docentes de las materias de sistemas operativos de cualquier ambiente académico. Además, podrá ser utilizado de ejemplo de aplicación para la enseñanza y difusión del uso de tecnologías adaptativas para la resolución de problemas complejos.

Teniendo como base lo obtenido en (i) podrán realizarse nuevos desarrollos e investigaciones para integrar nuevos aspectos al mecanismo adaptativo de reconfigurabilidad de SODIUM. Una vez que la totalidad de los aspectos en SODIUM sean reconfigurables y adaptativos, podrá ser utilizado para demostrar el potencial de aplicación de lo obtenido en (ii).

La base de conocimiento lograda en (iii) podrá ser utilizada por desarrolladores de sistemas operativos reconfigurables, ya sean comerciales, de producción o académicos que puedan beneficiarse de las ventajas obtenidas en (ii) para la implementación de tecnologías adaptativas en la toma de decisiones complejas.

4. ANÁLISIS DE RECONFIGURABILIDAD EN SODIUM

4.1. SELECCIÓN DE UN ASPECTO RECONFIGURABLE

Al momento de desarrollar la presente investigación, SODIUM contaba ya con la capacidad de reconfigurar algunos de sus aspectos, es decir, con varios modos de funcionamiento intercambiables sin necesidad de recompilar. Sin embargo, dada su naturaleza altamente colaborativa y, a veces, caótica de su desarrollo, no ha sido posible aún desarrollar mecanismos de reconfiguración para el resto sus aspectos, o para todos en general.

Los aspectos (y sus modos de funcionamiento) de SODIUM que actualmente cuentan con mecanismos adaptativos son:

- Administración de Memoria
    - Modo Paginado con Swapping
    - Modo Paginado Puro
    - Modo Particionado Variable
    - Modo Particionado Equitativo

- Planificador de Procesos
    - Algoritmo Round-Robin
    - Algoritmo Round-Robin con Prioridades
    - Algoritmo Round-Robin de Quantum Variable
    - Algoritmo de First Come First Serve
    - Algoritmo Shortest Finishing Job First
    - Algoritmo Best-Time-Of-Service

En el marco de esta investigación se utilizará el sólo aspecto de planificación de procesos como base para la aplicación de tecnologías adaptativas por los siguientes motivos:

- El resultado de ambos aspectos es relevante para la investigación, pero sólo uno es suficiente para demostrar o refutar las hipótesis planteadas.

- El mecanismo de reconfiguración de un algoritmo de planificación a otro es relativamente simple comparado al de cambiar un modo de administración de memoria.

- La planificación de procesos cuenta con muchas métricas útiles y fáciles de medir para comparar la eficiencia de un algoritmo comparado a otro, y respecto al comportamiento del usuario, expresado a través de los procesos.

- La cantidad de transiciones [4] posibles es mucho mayor al de la administración de memoria, aumentando la flexibilidad para la aplicación de tablas de decisión.

4.2. METODOLOGÍA DE ANÁLISIS DE ASPECTOS RECONFIGURABLES

Antes poder elaborar la tabla de decisión para la reconfiguración del planificador de procesos de SODIUM necesaria para la aplicación de tecnologías adaptativas, es preciso identificar, clasificar, y modelar dicho aspecto en base a una metodología ordenada. Esta metodología deberá ser aplicada a todos los restantes aspectos de SODIUM, a medida que vayan adoptando tecnologías adaptativas para su reconfiguración.

[4] Cambio posible entre un modo de funcionamiento de un aspecto a otro.

En esta investigación se ha utilizado la metodología propuesta por [Martin – 2012] en la que se establecen cuatro pasos ordenados para la elaboración de un diseño para cada aspecto reconfigurable, sus diferentes modos posibles, y las transiciones entre ellos:

1) Analizar la situación actual para cada transición –si es que existe alguna–, incluyendo factibilidad y la interfaz por la cual se puede realizar.
2) Enumerar los modos actualmente disponibles y elaborar un grafo de transiciones, incluyendo otros posibles modos, para descubrir nuevas transiciones posibles en tiempo de ejecución.
3) Realizar el diseño reconfigurable indicando detalles de implementación, nivel temporal, y elementos a verificar por posibles pérdidas de información para cada transición.
4) Diseñar las interfaces de usuario que mejor se adecuen a la facilidad de configuración del caso.

Puede observarse que esta metodología fue concebida para diseñar nuevos aspectos reconfigurables, a partir del análisis de aspectos no reconfigurables. Sin embargo, aún teniendo en cuenta que en esta investigación no se harán nuevos desarrollos de aspectos reconfigurables, sigue siendo igualmente útil para analizar el estado actual de los aspectos de SODIUM y la elaboración de las tablas de decisión correspondientes.

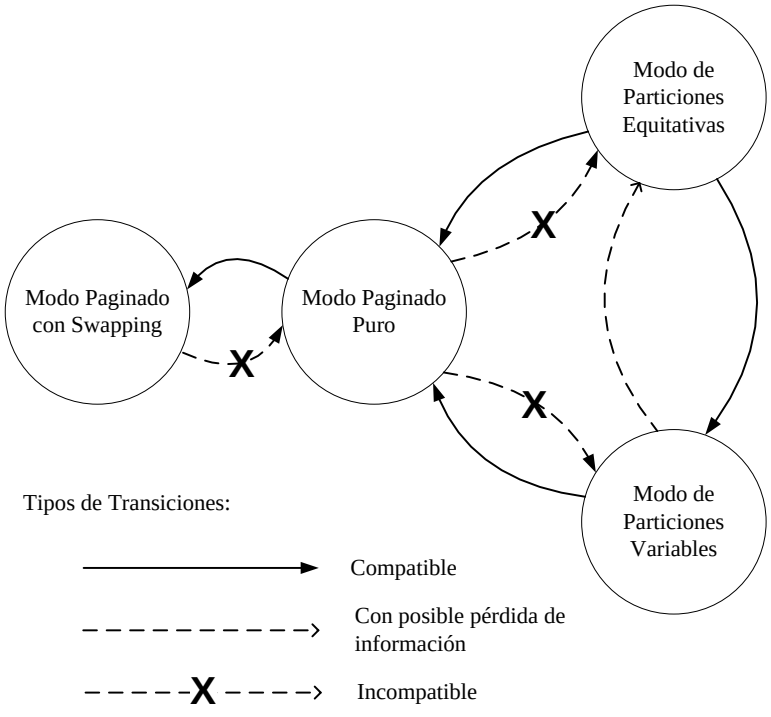
Antes de realizar el análisis planteado por el primer paso de la metodología, se hace necesario definir los siguientes conceptos:

4.2.1. TRANSICIONES

Se define como *transición* a cada cambio entre un modo de funcionamiento a otro diferente de un mismo aspecto del sistema operativo. Los mecanismos implementados que permitan

realizar las transiciones deben minimizar los efectos secundarios sobre otros procesos, recursos, o dispositivos ajenos a la reconfiguración del aspecto.

El análisis de transiciones y factibilidad debe contemplar cada una de las transiciones entre modos para el aspecto a analizar, ya que no todos son compatibles entre sí. Esto puede ser representado mediante un grafo dirigido [Figura 2.1] en el cual, la presencia de un lazo uniforme indica una transición compatible, un lazo intermitente indica una transición con posible pérdida de información o desperdicio de recursos, y un lazo intermitente tachado indica una transición incompatible.



[Figura 2.1] Ejemplo de diagrama de transiciones entre modos de administración de memoria en tiempo de ejecución en SODIUM

En la figura del ejemplo, el cambio de un modo de administración de memoria particionado a uno paginado podría realizarse de manera transparente. Sin embargo, no es

posible garantizar que el cambio inverso sea compatible: podría haber más espacio asignado a páginas en espacio virtual que no cabrían en un contexto particionado de sólo memoria física.

No es necesario contemplar las relaciones directas perfectamente compatibles que pueden ser logradas transitivamente, por ejemplo, pasar de "modo de particiones equitativas" a "modo paginado con swapping", ya que, al tratarse, en definitiva, del diseño de un producto software, contemplar todas las transiciones posibles puede aumentar las líneas de código finales, el esfuerzo de desarrollo necesario, y dificultaría su mantenimiento. Con una transitividad compatible, pasando por varios modos, se puede logar el mismo resultado con menor esfuerzo de desarrollo.

4.2.2. INTERFACES

La metodología sugiere definir una interfaz amigable y fácil de entender para permitir al usuario o al programador de aplicaciones realizar la reconfiguración.

En este punto, el sistema operativo Kea [Veitch – 1996] propone la utilización de Portales: interfaces proceso/aspecto con forma de objeto instanciable que pueden ser conectadas, por un lado, a cualquier proceso, y por el otro, a cualquier modo de implementación de un aspecto. El resultado es que diversos procesos pueden utilizar simultáneamente diferentes modos de un mismo aspecto o servicio sin interferir con los demás.

En la presente investigación, al no depender una interacción con el usuario, no será preciso definir interfaces accesibles a través de llamadas a sistema. En contraste, deberán evaluarse y definirse las funciones que le permitirán al kernel de SODIUM efectuar las operaciones previas a cada transición, así como la que le permita realizar el cambio de algoritmo de planificación.

4.3. ANÁLISIS DEL PLANIFICADOR DE PROCESOS RECONFIGURABLE DE SODIUM

A continuación se realizará un análisis del aspecto reconfigurable del planificador de procesos de SODIUM, utilizando la metodología descripta. En el [Anexo I] se encuentra la estructura general del planificador reconfigurable de SODIUM y en el [Anexo II], se encuentran detallados los algoritmos de planificación de SODIUM, que serán recurrentemente referenciados en el transcurso de este, y los siguientes capítulos.

4.3.1. MODOS Y TRANSICIONES

El planificador de procesos de SODIUM cuenta con seis algoritmos de selección de procesos, cada uno relacionado a su implementación –modo– particular. Cada modo de implementación no sólo engloba a su algoritmo, sino que también las estructuras, datos, y estadísticas que deben mantenerse.

Con la metodología utilizada, se hace necesario el uso de abreviaciones para cada modo debido que serán utilizadas para la notación de transiciones utilizadas tanto durante el análisis como en los diagramas. Las siguientes son las abreviaciones a utilizar para cada modo:

Modo	Abreviación
Round-Robin	**RR**
Round-Robin con Prioridades	**RRPR**
Round-Robin de Quantum Variable	**RRQV**
First Come First Serve	**FCFS**
Shortest Finishing Job First	**SJFS**
Best-Time-Of-Service	**BTS**

La versión base de SODIUM desde donde parte el presente trabajo provee interfaces para el cambio de planificador en tiempo de ejecución. Sin embargo, cada algoritmo maneja sus propias colas y estructuras separadamente, por lo que no se analizan los posibles

conflictos, o pérdidas de información que se generan del cambio. En cambio deben analizarse todas las transiciones, y extraerse la mayor cantidad de información posible del modo implementado anterior, para aprovechar estructuras y funciones ya definidas.

Dado que, desde cada uno de los modos disponibles, debería ser posible para el sistema operativo intercambiarlo por cualquiera de los otros cinco –algunas limitaciones a tener en cuenta se analizan a continuación–, el planificador reconfigurable de SODIUM cuenta con 30 transiciones posibles.

Si bien analizar cada una de las transiciones puede resultar una tarea intrincada en el caso del administrador de memoria, en el caso de la planificación de procesos resulta más simple y acotado, ya que no se altera el espacio de datos de los procesos, y tan sólo se altera el orden en que ejecutan.

A continuación se presenta un listado de todas las transiciones para el planificador de procesos de SODIUM, y se las identifica con un número para su posterior referencia.

ID	Transición	ID	Transición	ID	Transición
1	**RR ⇨ RRPR**	**11**	**RRQV ⇨ RR**	**21**	**SJFS ⇨ RR**
2	**RR ⇨ RRQV**	**12**	**RRQV ⇨ RRPR**	**22**	**SJFS ⇨ RRPR**
3	**RR ⇨ FCFS**	**13**	**RRQV ⇨ FCFS**	**23**	**SJFS ⇨ RRQV**
4	**RR ⇨ SJFS**	**14**	**RRQV ⇨ SJFS**	**24**	**SJFS ⇨ FCFS**
5	**RR ⇨ BTS**	**15**	**RRQV ⇨ BTS**	**25**	**SJFS ⇨ BTS**
6	**RRPR ⇨ RR**	**16**	**FCFS ⇨ RR**	**26**	**BTS ⇨ RR**
7	**RRPR ⇨ RRQV**	**17**	**FCFS ⇨ RRPR**	**27**	**BTS ⇨ RRPR**
8	**RRPR ⇨ FCFS**	**18**	**FCFS ⇨ RRQV**	**28**	**BTS ⇨ RRQV**
9	**RRPR ⇨ SJFS**	**19**	**FCFS ⇨ SJFS**	**29**	**BTS ⇨ FCFS**
10	**RRPR ⇨ BTS**	**20**	**FCFS ⇨ BTS**	**30**	**BTS ⇨ SJFS**

4.3.2. INTERFACES PARA LAS TRANSICIONES

Desde el punto de vista del diseño de software, diseñar un procedimiento distinto para cada uno de los conjuntos de acciones que cada transición supone, deberían luego programarse 30 funciones de transición diferentes. Sin embargo, A partir del análisis de las acciones necesarias para realizar cada, es posible reconocer patrones que permitirán definir luego funciones que generalicen parte del comportamiento en común que alguna de ellas tienen.

A continuación se utiliza un formato tabular para definir las nuevas interfaces y funciones que se debieron desarrollar en SODIUM, cuáles son las transiciones que las referencian, y cuál es su comportamiento.

Acción	Cambiar algoritmo base de planificación
Interfaz	sched_setbasealg(enum ALG)
Transiciones	5, 10, 15, 20, 25 (con parámetro BTS) 26, 27, 28, 29, 30 (con parámetro RR)
Comportamiento	Cambiar el valor de la variable global *siAlgoritmo,* según el valor provisto por el parámetro *ALG*. *siAlgoritmo* representa el algoritmo base a utilizar. Con el nuevo diseño reconfigurable puede tomar sólo dos valores: *ALG = 0 // Algoritmo Round Robin Generalizado* *ALG = 1 // Algoritmo Best-Time-Of-Service*

Acción	Inicializar Prioridades
Interfaz	sched_initPriorities()
Transiciones	1, 2, 17, 18, 22, 23, 27, 28
Comportamiento	Inicializar las prioridades de los procesos que no la tengan definida, asignándolas según el modo de planificación actual *Para siPlanificador = RR -> Utilizar mayor existente* *Para siPlanificador = FCFS -> Según orden de llegada*

	Para siPlanificador = SJFS -> Según duración estimada *Para siPlanificador = BTS -> Según tiempo de ejecución efectiva*

Acción	Inicializar Prioridades
Interfaz	sched_initPriorities()
Transiciones	1, 2, 17, 18, 22, 23, 27, 28
Comportamiento	Inicializar las prioridades de los procesos que no la tengan definida, asignándolas según el modo de planificación actual *Para siPlanificador = RR -> Utilizar mayor existente* *Para siPlanificador = FCFS -> Según orden de llegada* *Para siPlanificador = SJFS -> Según duración estimada* *Para siPlanificador = BTS -> Según tiempo de ejecución efectiva*

Acción	Inicializar duración de quantums
Interfaz	sched_initQTime()
Transiciones	2, 7, 18, 23, 28
Comportamiento	Inicializar las duraciones de quantum para las prioridades que no lo tengan definido, asignándolas según la fórmula: *Quantum = (40 – Prioridad) * 20 ticks*

Acción	Reordenar cola de procesos listos
Interfaz	sched_initReorder(enum Criterio)
Transiciones	3, 8, 13, 24, 29 (con parámetro *CTIME*) 4, 9, 14, 19, 30 (con parámetro *PTIME*)
Comportamiento	Reordenar la cola de procesos listos, ubicando primeros a los procesos que cumplan el criterio pasado por parámetro: - Para *CTIME*, se ordena la lista en función al momento de creación del proceso - Para *PTIME*, se ordena la lista en función a la duración estimada de ejecución

Acción	Deshabilitar modo expropiativo
Interfaz	sched_setNonPreemptive()
Transiciones	3, 4, 8, 9, 13, 14, 29, 30
Comportamiento	Cambiar el valor de la variable global *siPreemptive* a 0, donde: *siPreemptive = 0 // Modo expropiativo desactivado* *siPreemptive = 1 // Modo expropiativo activado*

Acción	Habilitar modo expropiativo
Interfaz	sched_setPreemptive()
Transiciones	16, 17, 18, 20, 21, 22, 23, 25
Comportamiento	Cambiar el valor de la variable global *siPreemptive* a 1, donde: *siPreemptive = 0 // Modo expropiativo desactivado* *siPreemptive = 1 // Modo expropiativo activado*

Acción	Unificar cola de procesos listos
Interfaz	sched_queueMode(enum MODO)
Transiciones	6, 7, 8, 9, 10 (con parámetro UN) 1, 12, 17, 22, 27 (con parámetro PR)
Comportamiento	Reordenar la cola de procesos listos, ubicando primeros a los procesos que cumplan el criterio pasado por parámetro: - Para *UN*, se crea una única lista de prioridad máxima con todos los procesos listos para ejecución. - Para PR, se crea una lista por prioridad, ubicando a los procesos listos según su prioridad

Acción	Definir modo de evaluación de duración de quantum
Interfaz	sched_quantumMode(enum MODO)
Transiciones	11, 12, 13, 14, 15 (con parámetro UN) 2, 7, 18, 23, 28 (con parámetro PR)

Comportamiento	Cambiar el valor de la variable global *siQuantumMode* a , donde: *siQuantumMode = UN // Se utiliza sólo el quantum de la prioridad 0* *siQuantumMode = PR // Se utiliza cuantum por procesos según lo definido para cada prioridad*

4.3.3. TABLA DE TRANSICIONES

Habiendo analizado las acciones necesarias para realizar las transiciones para el cambio de modo, y habiendo definido las interfaces para las rutinas que estas necesitan, es posible elaborar una tabla que sirva de base para la aplicación de reglas de decisión.

Regla		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Modo Actual	RR	x	x	x	x	x																									
	RRPR						x	x	x	x	x																				
	RRQV											x	x	x	x	x														
	FCFS																x	x	x	x	x									
	SJFS																					x	x	x	x	x				
	BTS																										x	x	x	x	x
Acción	setbasealg(BTS)					x					x					x					x					x				
	setbasealg(RR)																										x	x	x	x	x
	initPriorities()	x	x															x	x				x	x			x	x		
	initQTime()		x					x											x					x				x		
	initReorder(CTIME)			x					x					x											x				x	
	initReorder(PTIME)				x					x					x					x										x
	setNonPreemptive()			x	x					x	x			x	x														x	x
	setPreemptive()																x	x	x		x	x	x	x		x				
	queueMode(UN)						x	x	x	x	x																			
	queueMode(PR)	x											x					x					x					x		
	quantumMode(UN)											x	x	x	x	x														
	quantumMode(PR)		x					x											x					x					x		

[Tabla 4.1] Tabla de relación entre modos, transiciones, y acciones.

En la [Tabla 4.1], es posible relacionar fácilmente qué acciones pertenecen a cada transición y modo de implementación. Utilizando la nomenclatura propia de las tablas de decisión, las transiciones pueden ser consideradas reglas, y dispuestas en columnas, mientras que las acciones se disponen en filas. Luego, cada ‘x’ marca la correspondencia entre una regla y sus acciones.

Dado que las acciones a ejecutar para efectuar una transición entre un modo y otro no se ven alteradas por el contexto ni el uso del sistema operativo, se las puede considerar como invariantes. Esto permitirá abstraer la complejidad la tabla original, por una que contenga cada grupo de acciones específicas codificadas, por ejemplo, con una letra. Dicha simplificación es realizada por dos motivos: aprovechar de mejor manera la flexibilidad que proveen las tablas de decisión convencionales y, por otro lado, mejorar la facilidad de lectura de las posteriores tablas de decisión. Nótese que las reglas con conjuntos similares de acciones se encuentran codificadas con la misma letra. En la implementación sobre SODIUM, las letras utilizadas no tienen ningún significado específico; sólo se programan el conjunto de funciones que ellas representan.

Regla	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Modo Actual	RR	RR	RR	RR	RR	PR	PR	PR	PR	PR	QV	QV	QV	QV	QV	FC	FC	FC	FC	FC	SJ	SJ	SJ	SJ	SJ	BT	BT	BT	BT	BT
Acción	a	b	c	d	e	f	g	h	i	j	k	l	m	n	ñ	o	p	q	r	s	t	p	q	u	s	v	w	x	y	z

[Tabla 4.3] Tabla de relación entre modos, transiciones, y acciones.

En la [Tabla 4.3] puede observarse la simplificación de la [Tabla 4.1] original, utilizando la codificación en letras. No obstante estar utilizando una notación similar, esta tabla no representa todavía una tabla de decisión propiamente dicha, ya que sólo indica las acciones necesarias para llevar a cabo una transición. Solo luego de analizar los criterios bajo los cuales consideramos que el kernel de SODIUM debe llevar a cabo cada transición, será posible agregar las condiciones y los eventos que determinen esa capacidad de decisión.

4.3.4. TRANSICIONES ENTRE MODOS

Es posible establecer una correlación entre el modo actual, el modo al que se cambia, y la acción efectuada. Esta correlación se mantiene inalterada ya que es determinada durante el

diseño del aspecto reconfigurable. Para este caso, podemos basarnos en la [Tabla 4.3] para elaborar la [Tabla 4.4] que contiene dicha correlación.

		Modo Destino					
		RR	PR	QV	FC	SJ	BT
Modo Actual	RR	0	a	b	c	d	e
	PR	f	0	g	h	i	j
	QV	k	l	0	m	n	ñ
	FC	o	p	q	0	r	s
	SJ	t	p	q	u	0	s
	BT	v	w	x	y	z	0

[Tabla 4.4] Tabla de correlación entre modo origen, destino, y acción a tomar.

La [Tabla 4.4] será utilizada en la [Sección 6.3.2] como base para crear nuevas reglas en las que haga falta decidir qué acción tomar en base a los modos envueltos en la transición. La acción *0* indica que no es necesario realizar ninguna acción.

En el siguiente capítulo se plantean los criterios para definir la forma inicial de la tabla de decisión para el planificador de procesos de SODIUM, que formará la base para la construcción del tipo de tabla de decisión adaptativa.

5. CRITERIOS PARA LA TOMA DE DECISIONES

Para lograr que la [Tabla 4.3] con la forma transición-modo-acción provea al sistema operativo de una capacidad de decidir de manera autónoma si es necesario realizar una transición de modo, es preciso definir dos nuevos elementos clave: condiciones, que indiquen cuales reglas –generalidad de lo que nos referimos como transición– deben ejecutarse en un momento dado; y eventos, que nos indiquen cuándo deben evaluarse las condiciones.

Los eventos en sí mismos también pueden ser considerados condiciones, ya que si estos no se cumplen, no se evalúan las condiciones propiamente dichas, ni se ejecutarán las acciones a ellas asociadas. Sin embargo, desde el punto de vista del sistema operativo, la distinción es importante, ya que nos indicará cuáles son los acontecimientos luego de los cuáles debe evaluarse la tabla de decisión.

El conjunto transición-modo-acción-condición-evento nos permitirá establecer el qué, el cómo, y el cuándo de los cambios de modo. En este capítulo se establecen los criterios utilizados para determinar las condiciones y eventos a tener en cuenta, y sus relaciones con cada transición para el planificador de procesos reconfigurable de SODIUM.

5.1. CATEGORÍAS DE APLICACIÓN

La razón de que existan tantas propuestas diferentes para el diseño de algoritmos para planificar procesos es que cada uno de ellos responde mejor para cierto tipo de aplicaciones. La decisión final de implementar uno en particular para los sistemas operativos convencionales deriva directamente del análisis de cuáles serán las aplicaciones que los

usuarios ejecutarán. Por ejemplo, para un sistema operativo para uso general, se deben implementar aquellos algoritmos que respondan de mejor manera en promedio para todo tipo de situaciones; por otro lado, un sistema operativo específico para servidores de procesamiento intensivo en lotes, debería implementar algoritmos que permitan la rápida finalización de tareas sin interrupciones y en el orden en que llegan.

El propósito de construir un planificador reconfigurable para SODIUM que pueda tomar decisiones autónomamente es que reconozca los perfiles de aplicación siendo ejecutadas, y cambie su modo de ejecución para permitirles una mejor ejecución, tal como lo haría un sistema operativo especializado.

Por lo general, las aplicaciones de usuario pueden ser definidas dentro de un grupo de software según sea su objetivo. A su vez, cada uno de estos grupos puede ser categorizado respecto a las exigencias de servicios del sistema operativo y recursos de hardware. En la [Tabla 5.1] se enumeran cada una de estas categorías, los grupos que la componen.

Categoría	Grupo	Aplicaciones
Aplicaciones de Interacción Intensiva	Juegos	Unreal Tournament Need For Speed Call Of Duty
	Consola de comandos	Bourne-again Shell Korn Shell SODIUM Shell
	Interfaces gráficas de usuario	Gnome X Window System Ubuntu GUI
	Simuladores	EASA FFS SIMCOM NADS MiniSim
	Herramientas de Oficina	Microsoft Excel Microsoft Word Bloc de Notas
Aplicaciones Multimedia	Reproductores de Audio y Video	GOM Player Windows Media Player Winamp
	Editores de Audio y Video	Adobe Premiere Sony Vegas Music Maker
	Renderizadores de 3D	3D Studio

		AutoCad
Aplicaciones de E/S Intensiva	Programas de Transferencia de datos	cp dd
	Grabadores de CD/DVD/Blu-Ray	Nero CDBurnerXP Roxio
	Motores de Base de Datos	Microsoft SQL Server MySql Oracle DB
Aplicaciones de Internet	Navegadores Web	Mozilla Firefox Opera Google Chrome
	Programas de red	Ping Trace Telnet SSH
Aplicaciones de Procesamiento Intensivo	Compiladores	gcc nvcc mini-gw
	Benchmarks	Super-Pi 3DMark
	Uso Científico	BLAS HPCC ATLAS
Aplicaciones de Sistema	Mantenimiento	Desfragmentador Verificador de Discos Copia de Seguridad Liberador de Espacio
	Servicios	Manejador de cola de impresión Comunicación Inter-Procesos

[Tabla 5.1] Tabla de categorías de aplicaciones.

Si bien cada aplicación en particular tiene sus necesidades puntuales de procesamiento, entrada-salida, e interacción, podemos establecer que, en general, dentro del grupo al que pertenecen, estas características permanecen similares. Con menor precisión aún es posible establecer que los grupos de programas pertenezcan estrictamente a la categoría en la que están englobados. Por ejemplo, existen juegos con alta demanda de recursos multimedia, o de uso de red.

Como es imposible tener en cuenta la demanda particular, ya sea de cada programa o cada grupo, es necesario abstraer el trabajo de definir una configuración inicial de la tabla de decisión teniendo en cuenta sólo las categorías de manera general.

Luego, podemos definir, para cada categoría, cuáles son los criterios particulares de planificación más importantes a ser considerados. Estos criterios estarán definidos en base al valor relativo que se espera conseguir según las métricas expuestas en el [Anexo III]. Este análisis, reflejado en la [Tabla 5.2], será la base para establecer las condiciones de la tabla de decisión inicial de SODIUM. La mayor parte de los criterios especificados fueron obtenidos de [Silberschatz – 2012].

Categoría	Criterios	Métricas
Aplicaciones de Interacción Intensiva (II)[5]	El tiempo total de ejecución de un proceso no es importante en los sistemas interactivos. En estos sistemas es necesaria un tiempo de respuesta rápido en promedio, y de varianza reducida.	Tiempo de Respuesta Tiempo de Espera
Aplicaciones Multimedia (M)	Las aplicaciones multimedia requieren el procesamiento continuo de grandes cantidades de datos, que deben ser emitidos a través de los dispositivos. No debe sufrir retrasos dado que esto afecta la continuidad de la salida.	Tiempo de Espera Uso de CPU
Aplicaciones de E/S Intensiva (ES)	Se espera que se finalicen la mayor cantidad de procesos encargados de transferencia en la menor cantidad de tiempo.	Tasa de Finalización Tiempo de Respuesta
Aplicaciones de Internet (WEB)	Las aplicaciones de Internet representan procesos que deben tener un buen balance entre tiempo de respuesta y una distribución equitativa de la ejecución. Como deben pasar tiempo de ejecución solicitando operaciones de red, es posible que su tiempo efectivo de servicio sea reducido.	Tiempo efectivo de servicio Tiempo de Respuesta
Aplicaciones de Procesamiento Intensivo (P)	Las aplicaciones de procesamiento intensivo, deben ejecutar procesos no-interactivos, apuntados al resultado, y con altos tiempos de ejecución. Intentar garantizar un menor tiempo total de ejecución es de gran importancia.	Tiempo de Ciclo de Vida Tasa de Finalización
Aplicaciones de Sistema (S)	Las aplicaciones propias de sistema, funcionan como procesos sin utilidad propiamente dicha para el usuario, por lo que es importante que ocupen el menor tiempo posible de procesamiento.	Overhead

[Tabla 5.2] Tabla de características importantes según categoría de aplicación.

[5] Se utilizarán estas abreviaciones en las siguientes tablas para referenciar a las categorías de aplicación.

5.1.1. DETERMINANDO LA CATEGORÍA DE UN PROCESO

El problema principal para el uso de categorías de aplicaciones es que no existe una forma directa y segura de categorizar a un proceso en pleno funcionamiento. Además, como en esta investigación se postula que la toma de decisiones debe realizarla el sistema operativo sin intervención de los programadores o usuarios, no es de esperarse que éstos le indiquen a qué categoría pertenece la aplicación que ejecutará. Por lo tanto, es necesario formular algún método de inferir la categoría de un proceso en base a su comportamiento, aunque sea de manera aproximada.

Afortunadamente, existen trabajos existentes enfocados a resolver este problema. [Varghese – 2007] propone una técnica de clasificación de procesos basada en el análisis cuantitativo de los syscalls utilizados. Si bien su trabajo se enfoca a la evaluación de posibles amenazas a la seguridad, es posible utilizar la misma metodología para clasificar procesos en base al comportamiento descripto en la [Tabla 5.2].

En el sistema operativo Solaris [McDougall – 2007], se mantienen estructuras para cada proceso que registran, no sólo la cantidad y clasificación de syscalls para cada proceso, sino también la cantidad de bytes transmitidos de entrada/salida, y los fallos de página. Implementar esto en SODIUM permitirá contar con la información suficiente para determinar su categoría.

En esta investigación se implementa el enfoque de [Varghese – 2007], pero ampliando el rango de datos a evaluar tomando como base las estructuras definidas para cada proceso en Solaris.

5.2. SELECCIÓN DEL ALGORITMO DE PLANIFICACIÓN

Si las necesidades de los usuarios sólo requirieran la ejecución de una única aplicación mono-proceso en todo momento, no habría necesidad de utilizar un planificador. La única –y obvia– decisión sería cederle por completo la ejecución del procesador, sin interrupciones de tiempo de ningún tipo, o bien utilizar el planificador no expropiativo con menor overhead posible. Sin embargo, en la computación actual, este caso imaginario está muy lejos de ser el real.

Si los usuarios precisaran ejecutar varias aplicaciones de manera simultánea, pero éstas pertenecieran todas a la misma categoría de aplicaciones, podríamos decidir qué planificador utilizar en cada momento en base a cuál satisface mejor los criterios establecidos en la [Tabla 5.2]. Por ejemplo, si supiéramos con pura certeza que el usuario ejecutará sólo aplicaciones de interacción intensiva, bastaría con hacer que el planificador de procesos de SODIUM decida utilizar un algoritmo orientado a tal fin de manera permanente.

En el uso real de las computadoras, y también el uso que se pretende dársele a SODIUM, el planificador debe estar preparado para los casos generales; debe ser capaz de responder de la mejor manera ante la ejecución de todo tipo de aplicaciones de diferentes categorías de manera concurrente. Dada esta perspectiva, sólo sería plausible el uso de un planificador generalizado, tal como los implementados en Solaris 10 [McDougall – 2007], o Windows Vista [Russinovich – 2009].

Para la creación de la tabla de decisión inicial para determinar qué algoritmo implementar en cada momento se utilizará una solución que unifica cada uno de estos tres casos:

Caso I. Cuando exista tan solo un proceso presente en la cola de listos, se utilizará un algoritmo de planificación que le permita obtener el máximo uso del procesador con las menores interrupciones posibles.

Caso II. Cuando existan varios procesos de una única categoría presentes en la cola de listos, se utilizará el algoritmo de planificación que mejor se adapte a los criterios establecidos en la [Tabla 5.2]. En la siguiente sección determinaremos qué algoritmos se adaptan mejor para cada caso en base a un análisis de su funcionamiento, y su relación con las métricas listadas en el [Anexo III].

Caso III. Cuando existan varios procesos de diferentes categorías al mismo tiempo presentes en la cola de listos, se implementará el algoritmo de mayor generalidad posible. Este algoritmo será determinado de la misma manera que en los anteriores dos puntos: determinando cuál de ellos respeta de mejor manera la generalidad de los criterios; o, mejor dicho, cuál de ellos produce menos problemas de gravedad a la menor cantidad de categorías.

Puede decirse que la selección de estos tres casos simplifica en exceso el posible espectro combinatorio entre diferentes categorías. Es decir, dos categorías diferentes pueden compartir criterios que pueden ser satisfechos de mejor manera con un planificador específico, que con el de mayor generalidad. Existen, sin embargo, razones para la simplificación:

- En la selección de los criterios de planificación para cada categoría de aplicaciones ya supone una cierta imprecisión. Como ya se ha dicho, pueden existir aplicaciones específicas, que no reflejen perfectamente la necesidad de criterios de su propia categoría. En esta investigación damos por supuesto que, aún con esta imprecisión, la selección del planificador según estos criterios será satisfactoria en la mayoría de los casos. Intentar determinar criterios a priori para combinaciones de diferentes categorías supondría un grado de imprecisión que difícilmente responda de manera satisfactoria a la excesiva varianza en necesidades de las aplicaciones.

- Un análisis exhaustivo de todas las combinaciones sería excesivamente trabajoso. La combinatoria para 6 categorías diferentes sería de factorial(6), es decir 720 combinaciones diferentes a evaluar.

Es posible, sin embargo, salvar la simplificación si se implementa algún mecanismo que permita obtener, de manera automatizada, cuales son los planificadores que mejor responderán en cada caso. Esta forma de aprovechar al máximo el potencial de una tabla de decisión representa el núcleo de esta investigación, y será analizada en profundidad en el siguiente capítulo. Por lo pronto, la utilización de únicamente 3 casos generalizados facilita el análisis manual para determinar la forma inicial de la tabla de decisión estática, que servirá de base de conocimiento para la aplicación de tablas de decisión adaptativas.

5.3. RELACIÓN ENTRE CRITERIOS Y ALGORITMOS DE PLANIFICACIÓN

Para entender qué relación tiene un algoritmo de planificación con cada uno de los criterios importantes para cada categoría, debe analizarse su relación respecto a las métricas de

planificación utilizadas. En otras palabras, cada algoritmo produce un resultado positivo para un cierto tipo de métricas, mientras que su impacto en otras puede ser insignificante. Un algoritmo incluso puede producir que ciertas métricas indiquen valores inaceptables, en caso de estar planificándose aplicaciones que la consideran necesaria.

Analizando la relación de cada algoritmo respecto a su efecto sobre cada métrica será posible determinar fácilmente qué criterios favorece, y luego a sobre cuales categorías de aplicación conviene aplicarlos. Este análisis fue realizado mediante pruebas de control cuyo diseño se especifica en la [Sección 8.3]; y sus resultados, en la [Tabla 8.2].

A partir de los resultados de dichas pruebas de control puede observarse que, por ejemplo, el algoritmo FCFS obtiene los mejores resultados para las métricas de overhead y tiempo de ciclo de vida, mientras que el modo RRQV consigue los mejores resultados para uso del procesador, tiempo de espera, y tiempo de respuesta. Esta información será útil para determinar, en última instancia, qué algoritmos responderán mejor para cada escenario de ejecución y sus categorías.

5.4. RELACIÓN ENTRE CATEGORÍAS DE APLICACIONES Y ALGORITMOS

A partir de lo obtenido en la tabla [Tabla 8.2], es posible elaborar pruebas sobre escenarios de ejecución reales en los que se mida la eficiencia de cada algoritmo para cada categoría de aplicación. Las pruebas para la medición de la eficiencia y sus resultados se encuentran en la [Sección 8.4].

Con los datos obtenidos en la [Tabla 8.4] es posible ahora definir de manera estática qué decisiones para cada uno de los casos descriptos previamente en la [Sección 5.2], utilizando el mejor algoritmo para cada categoría:

Caso I. Cuando exista un solo proceso listo, conviene utilizar el algoritmo FCFS, ya que tiene el menor nivel de overhead, y además, al ser no expropiativo, reduce el tiempo de ciclo de vida del proceso.

Caso II. Cuando existan diversos procesos listos de una misma categoría, se utilizará el algoritmo de mejor resultado para cada categoría: para aplicaciones de interacción intensiva (II), resulta equivalente utilizar RRPR o RRQV, pero definimos RRQV ya que no requiere tener en cuenta el problema de la inanición; para aplicaciones multimedia (M), utilizaremos RRPR; para aplicaciones de E/S intensiva (ES), utilizaremos RRQV; para aplicaciones de Internet (WEB), utilizaremos BTS; para aplicaciones de procesamiento intensivo (P), utilizaremos SJFS; y para aplicaciones de sistema (S), utilizaremos FCFS.

Caso III. Cuando existan diversos procesos listos de diversas categorías, utilizaremos el algoritmo con menor conflictividad, y que responda relativamente bien ante todos los casos. El algoritmo más apropiado en este caso es el de Best-Time-Of-Service

Puede observarse que, dados los resultados obtenidos durante las pruebas de control, el algoritmo de Round-Robin básico (RR) no resulta mejor que los otros en ninguno de los casos analizados, pero, aún así, es mejor que otros en el promedio de los casos. Esto se debe a que los algoritmos especializados RRPR y RRQV que de él surgen, permiten especializar su comportamiento para todas situaciones particulares en las que un algoritmo de este tipo es ideal. Debido a esto, las reglas que contemplen transiciones al modo RR no contarán con

ninguna condición a satisfacer en la tabla de decisión. Se conservan, no obstante, para ser consideradas por los dispositivos adaptativos a implementar en los siguientes capítulos.

Habiendo resuelto las acciones a tomar para cada uno de los casos definidos, se pueden obtener las condiciones a ser agregadas a la [Tabla 4.3], como uno de los elementos necesarios –todavía falta definir los eventos para su evaluación– para convertirla en una tabla de decisión estática. Para lograr esto, asociaremos las condiciones evaluadas para cada caso con las reglas que determinan la transición hacia el algoritmo correspondiente.

En la [Tabla 5.5] se encuentra la asociación condiciones-modo-reglas que será anexada a la [Tabla 4.3] al final de este capítulo. Dado que el conjunto de condiciones describe el universo completo de posibilidades, y que ninguna de ellas contiene intersecciones, es posible definir, en cada caso, qué planificador se debe utilizar.

No basta sólo con conocer el planificador a utilizar para que el kernel de SODIUM pueda seleccionar una regla de transición específica. Sin embargo, al contar con el modo actual, es posible determinar qué única regla cumple con la transición del modo actual hacia modo a utilizar. En este sentido, el modo actual también sirve como condición a evaluar para seleccionar una regla.

Regla		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Modo Actual		RR	RR	RR	RR	RR	PR	PR	PR	PR	PR	QV	QV	QV	QV	QV	FC	FC	FC	FC	FC	SJ	SJ	SJ	SJ	SJ	BT	BT	BT	BT	BT
Caso Actual	Caso I			x					x					x											x					x	
	II		x					x											x					x					x		
	M	x											x					x					x				x			
	ES		x					x											x					x					x		
	WEB					x					x					x					x					x					
	P				x					x					x					x										x
	S			x					x					x										x					x	
	Caso III					x					x					x					x					x					

[Tabla 5.5] Tabla de condiciones a evaluar para cada regla.

Por ejemplo, si estamos en presencia de un caso de ejecución de tipo WEB, sabemos que deberemos cambiar el modo hacia BTS. Teniendo el conjunto de reglas {5, 10, 15, 20, 25} de transición hacia BTS, y sabiendo que actualmente estamos ejecutando planificando actualmente en modo RR, podemos determinar que sólo la regla 5 cumple con ambas condiciones.

5.5. EVENTOS

Otro factor importante para lograr que SODIUM pueda tomar decisiones autónomamente sobre qué modo implementar en cada aspecto de su configuración es el que define en qué momentos deben evaluarse las condiciones, y finalmente ejecutar o no una de las reglas de transición. Se define como *evento* a todo suceso propio de la ejecución del sistema operativo que pueda ser registrado y al que pueda anexársele la ejecución de la evaluación del a tabla de decisión.

En el sentido más general, pueden definirse eventos en todos los manejadores del sistema. Estos pueden incluir: manejadores de interrupciones, de excepciones, de llamadas al sistema, de cambio de contexto, o de callgates. En el marco particular de esta investigación nos interesa definir nuestros eventos sobre sucesos que influyan en la planificación de los procesos; otros aspectos del sistema operativo pueden precisar eventos completamente diferentes.

Los eventos que utilizaremos son aquellos luego de los cuales alguna de las condiciones de tipo “Caso Actual” de la [Tabla 5.5] haya podido cambiar. Se precisará crear eventos luego de que se creen y finalicen procesos, pero también, como se tiene en cuenta también aquellos que ocasionen un cambio en el estado del proceso.

Se creó en el kernel SODIUM la función global *vFnVerificarPlanificador()*, que agregada al final de cada función de kernel que contenga un evento definido. Su objetivo será verificar la tabla de condiciones, dados el modo y el caso actuales, en busca de reglas de transición que deban cumplirse, bajo las nuevas condiciones determinadas por el evento.

5.5.1. CREACIÓN / FINALIZACIÓN DE PROCESOS

En SODIUM, la creación y finalización de procesos se realiza respetando el estándar POSIX para interfaces de sistema –syscalls y programas–.

La única forma que un proceso tiene para crear otro es duplicándose a si mismo mediante una llamada al syscall *fork()*. El manejador del sycall *fork()* crea los segmentos en memoria, los registros en la GDT, y copia el contenido de memoria y registros del proceso llamador al nuevo, generando una copia idéntica. La función encargada de realizar esta tarea en SODIUM se llama *int iFnDuplicarProceso(unsigned int uiProcPadre)*, por lo que se le agregará una llamada a *vFnVerificarPlanificador()* antes del *return* final.

Existen varias maneras que tiene un proceso para ser finalizado. Por un lado, los syscalls *exit()* y *join()* solicitan la finalización del proceso actual, mientras que otros como *kill()* pueden solicitar la finalización de otros procesos. En todos los casos, la función encargada de liberarlos de memoria y pasarlos a estado *terminado* es *int iFnEliminarProceso(unsigned int uiProceso)*, por lo que se agregará una llamada a *vFnVerificarPlanificador()* también antes de su finalización.

5.5.2. CAMBIOS DE ESTADO

Existen muchas circunstancias que pueden cambiar el estado de un proceso. En esta investigación sólo nos limitaremos a monitorear aquellos que provocan cambios del proceso de el estado *ejecutando* al estado *esperando* y de *esperando* a *listo*.

Específicamente en SODIUM, las únicas circunstancias que han sido implementadas hasta el momento que causan que un proceso pase de un estado de *listo* son: la operación de lectura de dispositivos *read()*, la de espera de otros procesos *waitpid()*, la de espera de tiempo *sleep()*, y la de solicitud de semáforos *sempost()*. Se han agregado llamadas a *vFnVerificarPlanificador()* al final las funciones de atención de dichos syscalls.

Los eventos que pueden provocar el cambio del estado *esperando* al de *listo* en SODIUM, son los que responden a las solicitudes de los syscalls que ocasionaron la espera. Se han agregado llamadas a la función de verificación al final de cada una de ellas.

5.5.3. RELACIÓN ENTRE EVENTOS Y ALGORITMOS DE PLANIFICACIÓN

No todos los algoritmos de planificación pueden verificar todos los eventos. FCFS y SJFS no pueden evaluar cambios de estado ya que son algoritmos no expropiativos. Intentar cambiar de algoritmo en el medio de la ejecución de un proceso bajo FCFS o SJFS implicaría violar su funcionamiento, y por lo tanto, anular sus beneficios. Tampoco son sensibles a la creación de un proceso, ya que éstos se encolarán según el criterio, pero no afectarán la ejecución del actual.

Como los eventos a evaluar serán únicamente dependientes del Modo Actual, y no del modo al que transicionamos –como sí son las acciones a realizar– es posible codificar funciones en SODIUM que automáticamente monitoreen los eventos necesarios según el algoritmo en ejecución. De esta manera, no hará falta incluir a los eventos como condiciones de la tabla de decisión final.

5.6. TABLA DE DECISIÓN CONVENCIONAL

A partir de una simple fusión de las tablas de condiciones obtenidas a lo largo de este capítulo, sumadas a las acciones y reglas para cada transición obtenidas en el anterior, es posible obtener la tabla de decisión[6] para el planificador de procesos de SODIUM. El resultado de dicha fusión entre las tablas [Tabla 5.5], y [Tabla 4.3], puede observarse a continuación, en la tabla [Tabla 5.6].

Una vez implementada, la tabla de decisión estática le permitirá a SODIUM contar los criterios necesarios para decidir qué planificador a utilizar. Luego de cada evento, al ejecutar la función *vFnVerificarPlanificador()*, se verificará si alguna de las 30 reglas de transición cumple con la combinación de modo-evento-caso, y si alguna lo cumple –sólo pueden cumplirse las condiciones para una única regla a la vez–, se llevan a cabo las acciones asociadas a esa regla.

Regla			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Condiciones	Modo Actual		RR	RR	RR	RR	RR	PR	PR	PR	PR	PR	QV	QV	QV	QV	QV	FC	FC	FC	FC	FC	SJ	SJ	SJ	SJ	SJ	BT	BT	BT	BT	BT
	Caso Actual	Caso I			x					x					x											x					x	
		II		x					x											x					x					x		
		M	x										x						x					x					x			
		ES		x					x											x					x					x		
		WEB					x					x					x					x					x					
		P				x					x					x					x											x
		S			x					x					x										x					x		
		Caso III					x					x					x					x					x					
Acción		func()	a	b	c	d	e	f	g	h	i	j	k	l	m	n	ñ	o	p	q	r	s	t	p	q	u	s	v	w	x	y	z

[Tabla 5.6] Tabla de decisión convencional del planificador de procesos de SODIUM

La tabla de decisión convencional se mantendrá inalterada durante la ejecución del sistema operativo ya que ni éste, ni los usuarios –aunque esta podría ser una opción válida– podrán cambiarla. Esto presenta un potencial problema para manejar casos excepcionales. Si alguna de las decisiones para casos particulares (casos de tipo II) resultara ser peor que la del

[6] También definida como Tabla de Decisión Convencional en la bibliografía y abreviada como TDC.

caso del algoritmo para el caso general (casos de tipo III), se haría necesario aplicar un cambio en la tabla.

Puede presentarse también el problema opuesto, si algunas de las situaciones particulares, en las que ejecutan procesos de diferentes categorías, pudiera ser mejor manejada por un algoritmo particular –en lugar del general–, se estaría perdiendo la potencialidad de proveer un mejor servicio.

Para enfrentar a estos dos problemas, frutos de la inmutabilidad de la tabla de decisión convencional, es que se propone en esta investigación aplicar técnicas adaptativas que permitan modificar la combinación reglas/condiciones, especializándola progresivamente en base a las métricas obtenidas. Este último aporte es realizado mediante la adición de reglas adaptativas dedicadas exclusivamente a la modificación de la tabla de decisión subyacente.

6. APLICACIÓN DE UN DISPOSITIVO ADAPTATIVO

6.1. PUNTO DE PARTIDA

Como se ha mencionado, la [Tabla 5.6] de decisión convencional obtenida al final del capítulo anterior carece de la posibilidad de ser modificada para salvar situaciones excepcionales en que se pierda rendimiento para el usuario, o aprovechar al máximo casos particulares manejados de manera particular.

Sabemos, por la experiencia del uso de tecnologías adaptativas en otros campos de la ciencia o los negocios para la toma de decisiones analizados en la [Sección 2.1.3], que éstas pueden ser aplicadas sobre tablas de decisión convencionales existentes, para obtener tablas de decisión adaptativas.

Las tablas de decisión adaptativas son una de las posibles implementaciones de lo que se denominan *dispositivos adaptativos*. La siguiente es una definición de *dispositivo adaptativo* extraída de [Neto – 2001]:

"Un dispositivo formal es denominado adaptativo cuando su comportamiento cambia dinámicamente, en directa respuesta a sus estímulos de entrada, sin interferencia de agentes externos, ni siquiera de sus usuarios. Para lograr esta capacidad, los dispositivos adaptativos deben ser auto-modificables. En otras palabras, cada posible cambio en el comportamiento debe ser completamente conocido en cualquier paso de su operación en el cual los cambios tienen lugar. Por lo tanto, los dispositivos adaptativos deben ser capaces de detectar todas las situaciones que puedan determinar posibles modificaciones y reaccionar adecuadamente imponiendo los cambios correspondientes al comportamiento del dispositivo."

Partiendo de la definición, sabemos que la tabla de decisión adaptativa que vayamos a construir a partir de la estática con la que contamos, deberá contener mecanismos dentro de sí misma que le permitan modificar su propio comportamiento. A su vez, estos mecanismos deberán procurar que las modificaciones se realicen en base a los estímulos de entrada, y con el fin de adaptar su comportamiento a una mejor experiencia para los usuarios.

Se espera que la aplicación de adaptatividad sobre la tabla de decisión convencional inicial produzca una *convergencia de comportamiento* que logrará, dada una cantidad finita de pasos, aproximarse al ideal para el usuario en particular.

Desde el punto de vista del comportamiento, la tabla de decisión convencional puede ser vista como una foto inmóvil que describe las reglas de transición y sus condiciones a partir del primer momento, y no evoluciona en base a los posibles estímulos que puedan suceder. Sin embargo, se hace necesario contar con ella ya que cumple tres funciones importantes:

- Permite tener en cuenta el conocimiento del experto, y de la experiencia empírica anterior. El comportamiento inicial definido por la tabla de decisión convencional elaborada por los expertos asegura, dentro de lo posible, un comienzo aceptable, y un buen punto de partida para comenzar la convergencia.

- Define el comportamiento del aspecto – en nuestro caso, el planificador – al momento de inicial la ejecución. Esto representa un aporte esencial, ya que de no existir, la convergencia hacia un comportamiento ideal demoraría muchísimo más, durante lo cual, el rendimiento podría resultar inaceptable.

- Si se implementa un sistema de resguardo y restauración utilizando archivos de configuración, se puede tomar el comportamiento de la tabla de decisión adaptativa de

la ejecución anterior y utilizarla como tabla de decisión inicial en la siguiente. Esto permitiría una evolución constante del comportamiento del aspecto, sin necesidad de comenzar de nuevo en cada ejecución.

Los pasos a seguir para convertir la tabla de decisión con la que contamos hacia su versión adaptativa serán:

- Analizar los componentes de una tabla de decisión adaptativa.
- Relacionar los componentes con los elementos de la planificación de procesos con los que contamos.
- Elaborar la tabla de decisión adaptativa para el planificador de procesos reconfigurable de SODIUM.

6.2. FORMULACIÓN DE LAS TABLAS DE DECISIÓN ADAPTATIVAS

6.2.1. TABLA DE DECISIÓN CONVENCIONAL SUBYACENTE

Las tablas de decisión adaptativa (TDA) se construyen en base a una tabla de decisión convencional subyacente. En su formulación original de [Hughes – 1968], las tablas de decisión convencionales (TDC) se definen de la siguiente manera:

$$TDC = (CT, R, CV, t_0, AT, A)$$

Donde:

- CT representa el conjunto de todas las configuraciones posibles de la tabla de decisión; El conjunto finito de reglas de decisión que define es $R = \{r_i\ ,\ 1 \leq i \leq n\}$
- R representa cada una de las reglas $r_i = (c_{ij}, a_{ik}) \in R$. Donde c_{ij} Representa un valor para la condición C_j de la regla r_i, y a_{ik} representa un valor para la acción A_k respecto a la regla r_i

- CV representa el conjunto finito de los valores c_{ij} válidos para las condiciones C_i
- t_0 representa la configuración inicial de la tabla de decisión
- AT representa el conjunto de configuraciones aceptadas para la tabla de decisión
- A representa el conjunto finito de acciones.

En la [Tabla 6.1] se puede observar una construcción de dicha especificación en forma tabular. Dicha tabla sirve como un modelo para el armado de nuevas tablas de decisión según la formulación de [Hughes – 1968]. Puede observarse que la tabla de decisión [Tabla 5.6], obtenida al final del capítulo anterior, cuenta con este mismo formato y cumple con la especificación de una tabla de decisión convencional.

Tabla de Decisión Convencional	Regla		r_1	r_2	…	r_k
	Condiciones	C_1	c_{11}	c_{12}	…	c_{1k}
		…	…	…	…	…
		C_n	c_{n1}	c_{n2}	…	c_{nk}
	Acciones	A_1	a_{11}	a_{12}	…	a_{1k}
		…	…	…	…	…
		A_m	a_{m1}	a_{m2}	…	a_{mk}

[Tabla 6.1] Formato de una Tabla de decisión convencional según la definición

6.2.1. DEFINICIÓN FORMAL DE UNA TABLA DE DECISIÓN ADAPTATIVA

De acuerdo con [Stange – 2011], una versión adaptativa de una tabla de decisión convencional puede ser obtenida adicionándosele a ella filas en las que se incluyan funciones adaptativas. Además, a cada columna que representa una regla simple se le debe adicionar una llamada para cada función adaptativa asociada a su ejecución en particular.

Cada vez que una regla adaptativa es aplicada, una acción adaptativa es invocada, permitiendo que se realicen cambios en el conjunto de reglas subyacente. Cuando una acción adaptativa es ejecutada, el conjunto de reglas resulta modificado y, en consecuencia, resulta

cambiada la cantidad de columnas de la tabla, o sus valores para cada fila. La cantidad de filas no sufren cambios y se mantienen inalteradas.

En su tesis doctoral, [Tchemra – 2009], se basa en la definición original de las tablas de decisión adaptativas (TDA) construidas sobre tablas de decisión convencionales, y elabora una definición formal de la siguiente manera:

$$TDA = (TDN, CA)$$

Donde:

- TDN representa la tabla de decisión convencional subyacente.
- CA representa todo el mecanismo adaptativo que se le anexa a TD

El mecanismo adaptativo CA perteneciente a TDA se encuentra definido de la siguiente manera:

$$CA = (BA, AA, R)$$

Donde:

- AA - *after actions* - representa el conjunto de funciones adaptativas (FA) a ser ejecutadas después de las reglas no adaptativas de la tabla, pertenecientes a la tabla de decisión convencional subyacente.
- BB - *before actions* - representa el conjunto de funciones adaptativas (FA) a ser ejecutadas antes de las reglas no adaptativas de la tabla, pertenecientes a la tabla de decisión convencional subyacente.
- R representa el conjunto de reglas y sus valores, ampliando la definición inicial para indicar cuales funciones adaptativas se deben ejecutar junto a cada una de ellas.

Las funciones adaptativas (FA) están, a su vez, definidas de la siguiente manera:

$$FA = (nombre, P, V, G, BA, AD, AA)$$

Donde:

- $nombre$ es la identificación utilizada para referenciar a la función adaptativa.
- P representa la lista de parámetros formales que serán utilizados para referenciar valores pasados a la función adaptativa cuando es llamada.
- V (opcional), representa una lista de variables que son utilizadas para almacenar valores resultantes de la aplicación de la función adaptativa.
- G (opcional), es una lista de generadores de nombres utilizados para referenciar nuevos valores, cuando una función adaptativa es activada.
- AA (opcional) se utiliza para indicar una llamada a una acción adaptativa anterior.
- BB (opcional) se utiliza para indicar una llamada a una acción adaptativa posterior.
- AD representa el cuerpo de la función adaptativa y está compuesta por un conjunto de acciones adaptativas elementares de consulta, inclusión, y exclusión que modifican el conjunto de reglas actual de la tabla de decisión.

6.2.2. COMPONENTES DE UNA TABLA DE DECISIÓN ADAPTATIVA

En la [Tabla 6.2] se puede observar la estructura general de una tabla de decisión adaptativa. Las partes sombreadas indican las filas y columnas pertenecientes a la tabla de decisión convencional subyacente, y las partes sin sombra representan los mecanismos adaptativos (CA) agregados.

<table>
<tr><td></td><td colspan="3">Columnas de Funciones Adaptativas</td><td>Columnas de Reglas</td></tr>
<tr><td>Fila de Cabecera</td><td colspan="4">Valores de cabecera (tags)</td></tr>
<tr><td>Filas de Condiciones</td><td rowspan="3">Declaración de Función Adaptativa 1 (acciones)</td><td rowspan="3">Declaración de Función Adaptativa … (acciones)</td><td rowspan="3">Declaración de Función Adaptativa n (acciones)</td><td>Valores de las condiciones</td></tr>
<tr><td>Filas de Acciones</td><td>Acciones a ser aplicadas</td></tr>
<tr><td>Camada adaptativa (nombre, funciones, y parámetros)</td><td>Funciones adaptativas a ser ejecutadas y sus parámetros</td></tr>
</table>

[Tabla 6.2] Formato de una Tabla de decisión adaptativa. En gris se marca la parte compuesta por la tabla de decisión convencional subyacente.

Se puede observar que la tabla de decisión adaptativa agrega varios componentes a la tabla subyacente. A continuación se detalla la función de cada uno de ellos:

En la fila de cabecera se definen los tipos de cada columna que componen la tabla. El rótulo, o *tag*, es el valor que toma la fila de cabecera para cada columna. Dicho rótulo puede contar con uno de los siguientes valores posibles { H, ¿, +, - }, bajo las columnas de funciones adaptativas, para definir su comportamiento; o uno de los valores { S, R, E } bajo las columnas de reglas, para definir su comportamiento.

- **H**, indica una columna de cabecera de especificación de una f unción adaptativa. En cada celda de esta columna, junto a las líneas de funciones adaptativas, debe ser definida con alguno de los siguientes catacteres:
    - **B** – *before* -, indica que la función debe ser ejecutada antes de la regla no adaptativa subyacente
    - **A** – *after* -, indica que la función debe ser ejecutada después de la regla no adaptativa subyacente
    - **P** indica que esta fila corresponde a un parámetro de una función adaptativa.

  - **V** indica que se trata de una variable de una función adaptativa.
  - **G** indica que se trata de un generador de nombres.
- **?**, **+**, y **–** representan, respectivamente, las columnas de las funciones adaptativas elementares de consulta, creación, y eliminación de reglas de la tabla de decisión subyacente. En cada columna de este tipo, las celdas deben contener:
  - En las celdas de las condiciones, los valores específicos que serán evaluados.
  - En las celdas de las acciones, las acciones que serán tomadas por las reglas de decisión a agregar. Para consultas o eliminación, no debe definirse nada.
  - En las celdas de acciones adaptativas, se indican aquellas acciones adaptativas que serán ejecutadas por la función correspondiente a la columna.
- **S**, indica que la columna de la regla correspondiente es la inicial de la ejecución de la tabla de decisión. Sólo puede definirse una sola columna como inicial.
- **R**, indica que una columna corresponde a una regla normal de transición.
- **E**, corresponde a una regla de estado final que finaliza la ejecución de la tabla.

6.2.3. EJEMPLO DE FUNCIONAMIENTO

En la [Tabla 6.3] se muestra un ejemplo de una tabla de decisión adaptativa funcional simple. El objetivo de esta tabla es crear nuevas reglas toda vez que no se encuentren transiciones existentes para una condición.

El resultado –salida en pantalla– esperado para la tabla del ejemplo es “1, 2, 3, 4, 5, 6, 7, 8”. Sin embargo, puede observarse que sólo las reglas 3, 4, 5 y 6 utilizan la función print(V), encargada de imprimir el contenido de la variable V. La salida esperada de sólo ejecutar dichas reglas sería sólo de “2, 5, 6, 8”. Esto sucede debido a que faltan reglas que impriman los números faltantes 1, 3, 4, y 7.

		Declaración de cada función				Reglas						
		Ad1		Ad2		0	1	2	3	4	5	6
		H	?	H	+	S	R	R	R	R	R	E
Condiciones	V =				p1				2	5	6	8
	Estado =							ND				
Acciones	V:=				p1+1	1			3	6	7	
	Estado :=		D				ND					
	Print(V)				x				x	x	x	x
	Finalizar()											x
Funciones Adaptativas	Ad1	A			x			x	x	x	x	
	Ad2			B				x				
	p1			P				V				

[Tabla 6.3] Estado inicial de la tabla de decisión adaptativa del ejemplo

Las funciones adaptativas del ejemplo se ejecutarán coordinadamente para crear nuevas reglas que suplan la ausencia de los números faltantes cuando detecten la falta de reglas. Para analizar su funcionamiento, a continuación se detalla paso a paso como logra la salida esperada.

1 Paso)

La condición inicial de la tabla y las variables del sistema son:

Variable	Valor
Estado	
V	
Salida	

	Ad1		Ad2		0	1	2	3	4	5	6
	H	?	H	+	S	R	R	R	R	R	E
V =				p1				2	5	6	8
Estado =							ND				
V:=				p1+1	1			3	6	7	
Estado :=		D				ND					
Print(V)				x				x	x	x	x
Finalizar()											x
Ad1	A			x			x	x	x	x	
Ad2			B				x				
p1			P				V				

Al ejecutarse la regla inicial (0), V toma el valor 1. El resultado de la aplicación de esa regla resulta en los siguientes valores (se sombrean los cambios realizados):

Variable	Valor
Estado	
V	1
Salida	

2 Paso)

Ahora, la única regla de la tabla de decisión convencional subyacente que obedece a las condiciones impuestas es 1, ya que es la única que no cuenta con condiciones para su ejecución. Su único objetivo será definir la variable estado al valor ND (No Definido)

Variable	Valor
Estado	ND
V	1
Salida	

3 Paso)

Las dos reglas que aceptan las condiciones dadas en este punto son la 1 y la 2. La regla 1 asignará el valor "ND" a la variable Estado. Como ya tenía el valor ND asignado, no cambiará.

La regla 2, por otro lado, ejecutará la función adaptativa Ad2, antes de sus propias acciones, ya que está definida como B (before), pasándole como parámetro p1 el valor de V, que en este caso es 1. Ad2, cuenta con una única acción de tipo insertar (+) que creará una nueva regla (de número 6) que tome por condiciones que V valga lo especificado en p1, en este caso 1; y que de su ejecución V sea redefinida como p1 + 1 – es decir, 2 –. Además, la nueva regla, deberá también llamar previamente a su ejecución a la función adaptativa Ad1, especificada como A (after), y luego imprimir en pantalla el número correspondiente a V.

Luego, ejecutará la función adaptativa Ad1 que tiene como única acción definir la variable estado como D (definido). El resultado de la ejecución de ambas es el siguiente:

Variable	Valor
Estado	D
V	1
Salida	

	Ad1		Ad2		0	1	2	3	4	5	6	7
	H	?	H	+	S	R	R	R	R	R	E	R
V =				p1				2	5	6	8	1
Estado =							ND					
V:=				p1+1	1			3	6	7		2
Estado :=		D				ND						
Print(V)				x				x	x	x	x	x
Finalizar()											x	
Ad1	A			x			x	x	x	x		x
Ad2			B				x					
p1			P				V					

4 Paso)

En este punto se ejecutará nuevamente la regla 1, definiendo a Estado como ND.

La otra regla que puede ejecutar, dado el valor de V = 1 es la que se acaba de crear, es decir la número 7. Sus acciones serán: imprimir en pantalla el valor de la variable V, y cambiar el valor de la variable V a 2. Esta regla va a ejecutar también la acción adaptativa Ad1, de ejecución tipo A (after), luego de realizar sus propias acciones convencionales. La función adaptativa Ad1, tiene la simple función de cambiar el valor de la variable Estado a D. Esto anula lo hecho por la regla 1.

Como no se ha agregado ni quitado ninguna regla, la tabla de decisión se ha mantenido inalterada. El resultado de estas acciones sobre las variables es el siguiente:

Variable	Valor
Estado	D
V	2
Salida	1

5 Paso)

En este punto se ejecutará nuevamente la regla 1, definiendo a Estado como ND.

La otra regla que puede ejecutar, dado el valor de V = 2 es la número 3. Sus acciones serán similares a la del anterior paso, quedando la tabla de variables de la siguiente forma:

Variable	Valor
Estado	D
V	3
Salida	1, 2

6 Paso)

Nuevamente nos encontramos con una regla faltante para la condición de V = 3, que sea capaz de imprimir "3".

En este caso, la única regla que podrá ejecutar, dadas las condiciones, es la regla 1. Se repite exactamente lo explicado en los pasos 2 y 3. De esta manera se altera la tabla de decisión para incluir una nueva regla que contemple la condición V = 3.

Variable	Valor
Estado	D
V	3
Salida	1, 2

	Ad1		Ad2		0	1	2	3	4	5	6	7	8
	H	?	H	+	S	R	R	R	R	R	E	R	R
V =				p1				2	5	6	8	1	3
Estado =							ND						
V:=				p1+1	1			3	6	7		2	4
Estado :=		D				ND							
Print(V)				x				x	x	x	x	x	x
Finalizar()											x		
Ad1	A			x			x	x	x	x		x	x
Ad2			B				x						
p1			P				V						

7 Paso) A partir de la nueva regla 8 agregada, se podrá imprimir el número 3 en pantalla, repitiendo los pasos 4 y 5. El resultado de esa ejecución será:

Variable	Valor
Estado	D
V	4
Salida	1, 2, 3

8 Paso) Se repiten lo mismo especificado en los pasos 6 y 7, para crear y ejecutar la regla que permite imprimir el número 4 en pantalla. El resultado de esa ejecución dará por resultado:

Variable	Valor
Estado	D
V	5
Salida	1, 2, 3, 4

	Ad1		Ad2		0	1	2	3	4	5	6	7	8	9
	H	?	H	+	S	R	R	R	R	R	E	R	R	R
V =				p1				2	5	6	8	1	3	4
Estado =							ND							
V:=				p1+1	1			3	6	7		2	4	5
Estado :=		D				ND								
Print(V)				x				x	x	x	x	x	x	x
Finalizar()											x			
Ad1	A			x			x	x	x	x		x	x	x
Ad2			B				x							
p1			P				V							

9 Paso) Ya que las reglas para contemplar los valores de V = 5 y 6 ya estaban presentes en la tabla de decisión subyacente, su ejecución será similar a la de los pasos 4 y 5 que dará por resultado un cambio únicamente en la tabla de variables:

Variable	Valor
Estado	D
V	7
Salida	1, 2, 3, 4, 5, 6

10 Paso) En este punto, como no existe una regla para contemplar la condición de V = 7, se repiten los pasos 2 y 3 para crearla, y el paso 4 para ejecutarla, resultando en:

	Ad1		Ad2		0	1	2	3	4	5	6	7	8	9	10
	H	?	H	+	S	R	R	R	R	R	E	R	R	R	R
V =				p1				2	5	6	8	1	3	4	7
Estado =							ND								
V:=				p1+1	1			3	6	7		2	4	5	8
Estado :=		D				ND									
Print(V)				x				x	x	x	x	x	x	x	x
Finalizar()											x				
Ad1	A			x			x	x	x	x		x	x	x	x
Ad2			B				x								
p1			P				V								

Variable	Valor
Estado	D
V	8
Salida	1, 2, 3, 4, 5, 6, 7

11 Paso) En este punto, las únicas dos reglas que pueden ejecutar son 1 y 6. La regla 1 cambiará el valor de estado a ND, mientras que la regla 6 imprimirá el número 8 en pantalla, y finalizará la ejecución de reglas. El resultado final será:

Variable	Valor
Estado	ND
V	8
Salida	1, 2, 3, 4, 5, 6, 7, 8

6.3. APLICACIÓN DE TDA EN EL PLANIFICADOR DE PROCESOS DE SODIUM

6.3.1. TABLA DE DECISIÓN CONVENCIONAL SUBYACENTE

La evaluación de condiciones en el caso de la [Tabla 5.6] obedece a una lógica del tipo *OR Inclusivo*. Esto significa que, basta que sólo uno de los casos actuales válidos para cada regla se cumpla. Por ejemplo, para la regla 2, se ejecutará si estamos en un caso actual de categoría II, y también si estamos en un caso de categoría ES. Sin embargo, la definición formal de las tablas de decisión que deberemos utilizar para lograr un aprendizaje que pueda contemplar todos los casos particulares generalizados en el caso III, requiere que todas las

condiciones tengan un comportamiento del tipo AND. Para esto será necesario agregar reglas duplicadas que cumplan una de las dos condiciones, que será eliminada de la regla original.

Esto obliga a corregir la tabla [Tabla 5.6], debido a que las reglas 2, 3, 7, 8, 13, 18, 23, 24, 28, y 29 cuentan, cada una, con dos condiciones de caso actual que las satisfacen. Para esto es necesario elaborar una nueva [Tabla 6.4] con reglas adicionales duplicadas de aquellas que contengan una de sus condiciones y así poder eliminar las dobles condiciones de las originales.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Modo Actual	RR	RR	RR	RR	RR	PR	PR	PR	PR	PR	QV	QV	QV	QV	QV	FC	FC	FC	FC	FC	SJ	SJ	SJ	SJ	SJ	BT	BT	BT	BT	BT	RR	RR	PR	PR	QV	FC	SJ	SJ	BT	BT
Caso I			x					x					x												x				x												
II		x					x											x					x						x											
M	x											x					x					x						x											
ES																														x		x			x	x		x	
WEB					x					x					x					x					x															
P				x				x					x						x											x										
S																															x		x	x			x		x
Caso III					x					x					x					x					x															
func()	a	b	c	d	e	f	g	h	i	j	k	l	m	n	ñ	o	p	q	r	s	t	p	q	u	s	v	w	x	y	z	b	c	g	h	m	q	q	u	x	y

[Tabla 6.4] Tabla de decisión convencional corregida del planificador de SODIUM

Puede observarse que en la [Tabla 6.4], que la regla 31 es, en realidad, una copia de la antigua regla 2, en la que cada una conserva únicamente una condición a satisfacer para ser ejecutada. De la misma manera se operó con las nuevas reglas 32, 33, 34, 35, 36, 37, 38, 39, y 40.

Dado que, de aquí en más, podremos contemplar la existencia de reglas complejas – es decir, que contemplen la combinación de diversas condiciones de modo actual y tipo de aplicaciones –, la inclusión del caso III para generalizarlas ya no será necesaria. Su remoción ocasionará que algunas reglas, en particular 6, 11, 16, 21, y 26, no cuenten con condiciones para cumplir, y se vuelvan espurias. De la misma manera, el caso I, puede ser obviado del

estudio, ya que difícilmente requiera mayor análisis, y su exclusión permite una mayor facilidad para los siguientes desarrollos. Las reglas que sólo ejecutan bajo el caso I – estas son 3, 8, 13, 22, y 26 – también serán removidas.

Dado que los espacios en blanco indican una relación indiferente con la condición, se hará necesario también indicar explícitamente qué condiciones no deben cumplirse para una regla en particular. Para esto se agregará el símbolo "-" que indica que la condición debe ser falsa para que la regla pueda ejecutar. En la [Tabla 6.5] se refina lo elaborado en la [Tabla 6.4] incluyendo estos cambios, y con la numeración corregida.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Modo Actual	RR	RR	RR	RR	PR	PR	PR	QV	QV	QV	FC	FC	FC	FC	SJ	SJ	SJ	SJ	BT	BT	BT	RR	RR	PR	PR	QV	FC	SJ	SJ	BT	BT
II	-	x	-	-	x	-	-	-	-	-	-	x	-	-	-	x	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-
M	x	-	-	-	-	-	-	x	-	-	x	-	-	-	x	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-
ES	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	x	-	-	x	x	-	x	-	-
WEB	-	-	-	x	-	-	x	-	-	x	-	-	-	x	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-
P	-	-	x	-	-	x	-	-	x	-	-	-	x	-	-	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-
S	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	-	x	x	-	-	x	-	x	-
func()	a	b	d	e	g	i	j	l	n	ñ	p	q	r	s	p	q	s	w	x	z	b	c	g	h	m	q	q	u	x	y	

[Tabla 6.5] Tabla de decisión convencional subyacente a utilizar

6.3.2. MECANISMO ADAPTATIVO A IMPLEMENTAR

Puede observarse que el caso de estudio posee muchas similitudes con el caso del ejemplo de la [Tabla 6.3]:

- En el ejemplo, la tabla de decisión subyacente contaba con reglas existentes que permitían contemplar algunas de las combinaciones de condiciones posibles, y que no debían ser modificadas. Sucede lo mismo en el caso de la tabla de decisión del planificador de SODIUM, donde existen reglas definidas para los casos en que ejecutan varios procesos de una misma categoría. Estas reglas definen el conocimiento empírico, y, por lo pronto, no deberán ser cambiadas.

- En el caso del ejemplo, se debió implementar un mecanismo adaptativo que detectara cuando, dada una condición específica, no existiera ninguna regla que pudiera ejecutar, y entonces la agregara de forma automáticamente. Si bien, en el caso del ejemplo, el procedimiento utilizado fue relativamente simple, es posible utilizar un mecanismo similar para agregar reglas en el caso del planificador de SODIUM, dado casos combinados en los que ninguna regla de la tabla subyacente pueda ejecutar.

Por lo dicho, es entonces posible ampliar la [Tabla 6.5] de decisión convencional para el planificador de SODIUM, con el mismo mecanismo utilizado en el ejemplo de la [Tabla 6.3].

En la [Tabla 6.6] se implementa un mecanismo de inserción de reglas, en las que se contempla únicamente, por lo pronto, transiciones al modo de planificación BTS. Esta tabla de decisión adaptativa tiene el potencial de recrear por completo el caso III, con un comportamiento equivalente al de la tabla [Tabla 6.4]. Por cuestión de espacio, se excluyen de la representación algunas de las reglas de la tabla de decisión subyacente.

		Declaración de cada función adaptativa									Reglas													
		Ad1		Ad2							0	1	2	3	…	26	27	28	29	30	31	32	33
		H	?	H	+	+	+	+	+	+	S	R	R	R	R	R	R	R	R	R	R	R	E
Condiciones	Modo Actual				RR	PR	QV	FC	SJ	BT		RR	RR	RR	…	FC	SJ	SJ	BT	BT			
	Estado														…							N	
	II				C_1	C_1	C_1	C_1	C_1	C_1		-	x	-	…	-	-	-	-	-			
	M				C_2	C_2	C_2	C_2	C_2	C_2		x	-	-	…	-	-	-	-	-			
	ES				C_3	C_3	C_3	C_3	C_3	C_3		-	-	-	…	x	x	-	x	-			
	WEB				C_4	C_4	C_4	C_4	C_4	C_4		-	-	-	…	-	-	-	-	-			
	P				C_5	C_5	C_5	C_5	C_5	C_5		-	-	x	…	-	-	-	-	-			
	S				C_6	C_6	C_6	C_6	C_6	C_6		-	-	-	…	-	-	x	-	x			
Acciones	func()				$\$_m$	$\$_m$	$\$_m$	$\$_m$	$\$_m$	$\$_m$	0	a	b	d	…	q	q	u	x	y			
	Estado=		D												…						N		
Funciones Adaptativas	Ad1	A			x	x	x	x	x	x		x	x	x	x	x	x	x	x	x		x	
	Ad2			B																		x	

[Tabla 6.6] Tabla de decisión adaptativa del planificador de SODIUM

Tal como en el ejemplo de la [Tabla 6.3], la regla 31 ejecutará en todos los pasos para definir el estado como No Determinado (N). En caso de que exista alguna regla de transición existente para las condiciones actuales –que ejecutan la función adaptativa Ad2–, el estado pasará a Determinado (D). Si ninguna regla cumple con las condiciones actuales, el estado permanecerá en (N), lo que disparará la ejecución de la regla 32.

Las reglas 0 y 33 no cumplen ninguna función específica más que la de cumplir con la formulación original que exige la existencia de una regla de comienzo y otra de final.

La regla 32 ejecuta la función adaptativa Ad1 que agregará 6 nuevas reglas según el siguiente criterio:

- Las condiciones de las reglas serán las mismas que las que describen las categorías de aplicación de usuario actuales ($C_1 \ldots C_6$).
- Cada una de las reglas tendrá un modo actual diferente a las demás. De esta manera se contemplan todos los posibles modos desde los cuales se pueden efectuar transiciones dadas las categorías en ejecución.
- La función de transición ejecutar estará dada por la relación $\$_m$ derivada de la Tabla [Tabla 4.4]. $\$m$ toma el modo actual, y el nuevo modo al que la regla transiciona para obtener la acción a ejecutar. La función de transición 0, tomada cuando el modo de destino es el mismo que el actual, no efectúa ninguna acción.

6.3.3. FUNCIONAMIENTO DE LA TABLA DE DECISIÓN ADAPTATIVA

La lógica implementada en la [Tabla 6.6] provee un mecanismo inicial simple de aprendizaje sobre el uso de las aplicaciones por parte del usuario. Cuando detecta una situación de ejecución que no está contemplada en la tabla de decisión subyacente, agrega las reglas necesarias tal que:

- La primera vez que se detecta el nuevo conjunto de condiciones, éste debe ser planificado con el modo actual, ya que se asume que el modo actual parte de una situación de ejecución ideal para sí mismo, y las nuevas condiciones difícilmente la alteren críticamente. Por lo tanto, debe agregarse una regla de acción nula (0) para suplir la condición y continuar con el mismo modo.
- Se utiliza dicha primera ejecución para establecer un perfil de usuario. Dicho perfil indica que cuando se repita la condición, siempre debe utilizarse el modo determinado. Por lo tanto, se agregan otras 5 reglas que contemplan cualquier modo actual, y transicionan hacia el modo determinado en la primera ejecución.

El siguiente ([Tabla 6.7]) es un ejemplo en el cual se está ejecutando en modo FCFS, con procesos únicamente de categoría de procesamiento intensivo (P). Al crearse un nuevo proceso y detectar que es de categoría (M), se agregan nuevas reglas que transicionan a FCFS dadas las condiciones simultáneas (P) y (M). Las nuevas reglas – 34, 35, 36, 37, 38, y 39 – se muestran sombreadas.

	Declaración de cada función adaptativa									Reglas																					
	Ad1		Ad2							0	1	2	3	…	26	27	28	29	30	31	32	33	34	35	36	37	38	39
	H	?	H	+	+	+	+	+	+	S	R	R	R	R	R	R	R	R	R	R	R	E	R	R	R	R	R	R
Modo Actual				RR	PR	QV	FC	SJ	BT		RR	RR	RR	…	FC	SJ	SJ	BT	BT				RR	PR	QV	FC	SJ	BT
Estado														…							N								
II				C_1	C_1	C_1	C_1	C_1	C_1		-	x	-	…	-	-	-	-	-				-	-	-	-	-	-
M				C_2	C_2	C_2	C_2	C_2	C_2		x	-	-	…	-	-	-	-	-				x	x	x	x	x	x
ES				C_3	C_3	C_3	C_3	C_3	C_3		-	-	-	…	x	x	-	x	-				-	-	-	-	-	-
WEB				C_4	C_4	C_4	C_4	C_4	C_4		-	-	-	…	-	-	-	-	-				-	-	-	-	-	-
P				C_5	C_5	C_5	C_5	C_5	C_5		-	-	x	…	-	-	-	-	-				-	-	-	-	-	-
S				C_6	C_6	C_6	C_6	C_6	C_6		-	-	-	…	-	-	x	-	x				x	x	x	x	x	x
func()				$\$_m$	$\$_m$	$\$_m$	$\$_m$	$\$_m$	$\$_m$	0	a	b	d	…	q	q	u	x	y				c	h	m	0	u	y
Estado=		D												…						N									
Ad1	A			x	x	x	x	x	x		x	x	x	x	x	x	x	x	x		x		x	x	x	x	x	x
Ad2			B																		x								

[Tabla 6.7] Ejemplo de nuevas reglas al detectar una condición (M) + (S) no contemplada.

Si bien este criterio presenta una capacidad de adaptación acotada, sí permite establecer un perfil de planificación diferente para cada usuario o comportamiento, en el que se detecta de manera trivial qué prioridades tiene cada categoría de aplicaciones. Por ejemplo, dado un usuario que ejecute frecuentemente procesos de aplicación científica, y eventualmente ejecuta uno de aplicación multimedia, se seguirá priorizando el uso de planificadores tipo FCFS que benefician a la primera categoría.

Este criterio cuenta con la limitación de que, una vez determinada las transiciones para todas las condiciones posibles, ya no es posible modificarla y converge hacia una tabla de decisión convencional. Luego, pierde la capacidad de adaptarse a nuevos escenarios de ejecución, a menos que sea reiniciada al estado inicial. Esto provoca que la capacidad de la tabla esté enfocada especialmente para casos específicos, y no permita contemplar casos de ejecución donde estén presentes muchas categorías de aplicaciones de usuario. Como aún esta tabla no desarrolla todo el potencial que permiten los mecanismos adaptativos, la denominaremos Tabla de Decisión Adaptativa Simple (TDAS), a fin de diferenciarla de la versión extendida que obtendremos en el siguiente capítulo.

7. EXTENSIÓN PARA DECISIONES MULTI-CRITERIO

7.1. EXTENSIÓN DE LA TABLA DE DECISIÓN ADAPTATIVA

La [Tabla 6.6] de decisión adaptativa obtenida en el anterior capítulo permite la creación de nuevas reglas que contemplan condiciones no incluidas en la tabla de decisión convencional subyacente, agregando conocimiento a partir del aprendizaje sobre el uso que los usuarios le dan a las aplicaciones. Sin embargo, como se ha mencionado, presenta dificultades a la hora de evaluar múltiples criterios de adaptación.

Hasta este punto, el único criterio con el que se definen las nuevas reglas es el de mantener el mismo planificador actual, dado que se estima que el cambio de crear un nuevo proceso con categoría de aplicación es menor ante los n-posibles procesos de categoría única actuales. Sin embargo, esta simple heurística no contempla el uso de indicadores directos de rendimiento como lo son las métricas de planificación de procesos, ni tampoco presenta un mecanismo para alterar las reglas ya existentes en base a éstas.

Por lo tanto, se hace necesario extender la definición de la tabla de decisión adaptativa para incluir funciones adaptativas específicas cuyo objetivo sea el de decidir cuál planificador utilizar en base a la relación entre las categorías de procesos y las métricas mantenidas. En este sentido, en la tesis doctoral de [Tchemra – 2009] se presenta la definición de *tabla de decisión extendida* cuyo elemento central constituye un proceso de toma de decisión en la que intervienen múltiples criterios, considerados simultáneamente.

Las tablas de decisión adaptativas extendidas tienen por objetivo apoyar, por medio de una formulación expresiva, problemas en los cuales existen reglas conocidas, pero insuficientes como información para la toma de decisión ([Tchemra – 2010]). En ésta se

utilizan técnicas basadas en algoritmos clásicos de métodos de decisión multi-criterio que son aplicadas en forma de funciones adaptativas.

Una tabla de decisión adaptativa extendida se construye agregando una camada de funciones auxiliares a una tabla de decisión adaptativa subyacente. Estas nuevas funciones tienen como único objetivo definir valores para las variables utilizadas como datos de entrada para las funciones adaptativas de la tabla de decisión adaptativa subyacente.

Para el usuario o decisor – en nuestro caso, el kernel de SODIUM – la tabla de decisión adaptativa extendida permite, por ejemplo: proveer condiciones complejas para que el decisor exprese sus preferencias con respecto a los datos disponibles del problema de forma consistente; analizar los criterios impuestos a las reglas del problema y sus combinaciones, además de presentar alternativas de solución para situaciones no previstas anteriormente; e interactuar con el sistema para obtener soluciones viables.

7.1.1. DEFINICIÓN FORMAL DE UNA TABLA DE DECISIÓN ADAPTATIVA EXTENDIDA

Como se ha dicho, las tablas de decisión adaptativa extendidas ($TDAE$) son una ampliación de la formulación de las tablas de decisión adaptativas (TDA) convencionales presentadas anteriormente, en las que:

$$TDAE = (TDA, FM, M)$$

Donde:

- TDA representa la tabla de decisión adaptativa subyacente.
- FM representa el conjunto de funciones auxiliares que establecen valores a ser utilizados por el resto de las funciones adaptativas de la TDAsubyacente.

- M es cualquier método de decisión multi-criterio que desee adoptar el decisor. Se lo incluye como elemento definible para que la formulación de la $TDAE$ resulte genérica respecto al algoritmo de decisión.

En la definición original de la $TDAE$, se hace énfasis en que Mdebe ser implementado por el usuario mediante el uso de un lenguaje específico a ser compilado automáticamente para formar nuevas reglas. En nuestro caso, dicho método estará ya construido dentro del kernel, y no se proveerán interfaces de usuario para cambiarlo o alterar su funcionamiento.

Si bien las funciones auxiliares FM pueden ser implementadas como funciones adaptativas, debido a su complejidad, exigirán un repertorio más amplio de operaciones primitivas que las disponibles en el formalismo de la TDA subyacente. Estas funciones operan como interfaz entre el conjunto de reglas adaptativas y el resto de la tabla, efectuando operaciones que serían demasiado extensas si se expresan en forma de funciones adaptativas convencionales. Esto confiere a estas funciones la capacidad de calcular y alterar los valores de variables utilizados para la evaluación de condiciones que determinan la ejecución de las reglas.

Es importante destacar que las funciones FM deben ser ejecutadas como acciones del tipo B (before), dado que el funcionamiento de las reglas adaptativas contenidas en $TDAE$ depende de los valores calculados por ellas.

En la [Tabla 7.1] se presenta un esquema del formato general de una $TDAE$.

<table>
<tr><td></td><td colspan="3">Columnas de Funciones Adaptativas</td><td>Columnas de Reglas</td></tr>
<tr><td>Fila de Cabecera</td><td colspan="4">Valores de cabecera (tags)</td></tr>
<tr><td>Filas de Condiciones</td><td rowspan="4">Declaración de Función Adaptativa 1 (acciones)</td><td rowspan="4">Declaración de Función Adaptativa … (acciones)</td><td rowspan="4">Declaración de Función Adaptativa n (acciones)</td><td>Valores de las condiciones</td></tr>
<tr><td>Filas de Acciones</td><td>Acciones a ser aplicadas</td></tr>
<tr><td>Funciones Auxiliares (FM)</td><td>Llamados a las funciones auxiliares</td></tr>
<tr><td>Camada adaptativa (nombre, funciones, y parámetros)</td><td>Funciones adaptativas a ser ejecutadas y sus parámetros</td></tr>
</table>

[Tabla 7.1] Formato de una $TDAE$. En gris se marca la parte compuesta por la TDA convencional subyacente.

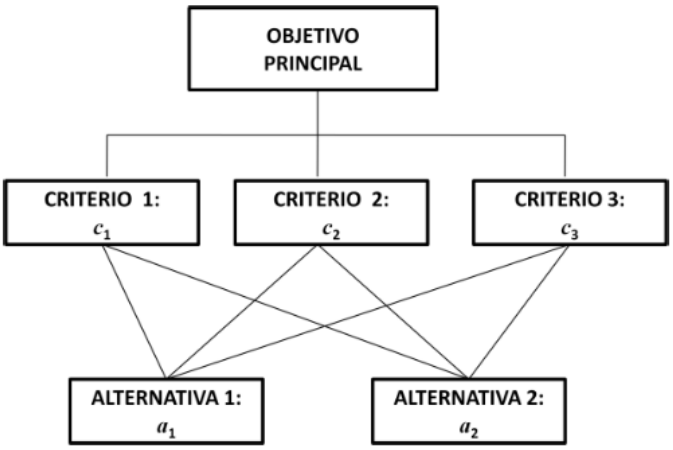
7.2. ELABORACIÓN DE LA TABLA DE DECISIÓN ADAPTATIVA EXTENDIDA

Los pasos para la construcción de una $TDAE$ se dividen en tres módulos. Cada módulo cuenta con un objetivo específico.

7.2.1. MÓDULO I

El módulo I consiste en un único paso de identificación de criterios y alternativas de decisión del problema en cuestión, y la relación entre ellos. Esta relación se define de forma binaria por la cual una alternativa puede verse afectada o no por un criterio. La [Figura 7.1] presenta un esquema de un posible mapa de relaciones de un problema con 3 criterios y 2 alternativas de solución Donde:

- C es el conjunto de criterios $C = \{c_1, c_2, c_3\}$
- A es el conjunto de alternativas $A = \{a_1, a_2\}$



[Figura 7.1] Estructura jerárquica del mapa de relaciones entre criterios y alternativas.

A partir de la relación entre los componentes del conjunto C de criterios y el conjunto A de alternativas, es posible establecer una tabla de relación con reglas que indiquen con 'S' o 'N' para cada combinación criterio-alternativa si dicho criterio tiene peso para la elección o no de dicha alternativa. En la [Tabla 7.2] se define la tabla de relación inicial t_0 para el ejemplo.

	r_1	r_2	r_3	r_4
Criterio 1	S	S	S	N
Criterio 2	S	N	N	N
Criterio 3	N	N	S	S
Alternativa 1	X	X		
Alternativa 2			X	X

[Tabla 7.2] Tabla inicial t_0 de preferencias para relaciones de criterios-alternativas.

La relación entre los criterios y las alternativas obtenidas en la [Tabla 7.2] representa un conjunto inicial R de relaciones incompleto, ya que no contempla decisiones para combinaciones de criterios que no determinen una alternativa de manera directa. Para dichos

casos, los siguientes módulos definen métodos que utilizan el peso relativo de cada criterio para determinar la alternativa de solución a utilizar.

7.2.2. MÓDULO II

Este módulo permite que el decisor compare el peso relativo de cada criterio en pares, ingresando valores que reflejen su propia preferencia, indicando qué importancia tiene cada criterio c_i respecto a otro c_j utilizando la escala fundamental de [Saaty – 1991] presentada en la [Tabla 7.3].

Escala	Significado
1	Ambos criterios son igualmente importantes
3	Preferencia moderada por un criterio respecto a otro
5	Preferencia Fuerte. Se favorece a un criterio en relación a otro
7	Preferencia Muy fuerte
9	Preferencia Extrema
2,4,6,8	Valores intermedios

[Tabla 7.3] Escala fundamental de Saaty

Como primer paso para la elaboración de este módulo, se debe construir una tabla de comparación entre criterios para evaluar la importancia de uno respecto al otro, utilizando los valores de la escala de Saaty para establecer dichos valores. La [Tabla 7.4] contiene valores para el caso del ejemplo tomado de [Tchemra – 2010].

	Criterio 1	Criterio 2	Criterio 3
Criterio 1	1	4	3
Criterio 2	1/4	1	2
Criterio 3	1/3	1/2	1

[Tabla 7.4] Matriz de preferencias entre pares de criterios

Los siguientes pasos requieren normalizar a 1 las filas y columnas de la matriz y verificar la consistencia de los valores relativos obtenidos.

En este módulo también se evalúa el peso relativo de cada criterio respecto a las diferentes alternativas. Para esto se utiliza la escala de la misma manera que en la comparación entre pares de criterios. La [Tabla 7.5] contiene valores de ejemplo de importancia relativa entre pares de alternativas para cada criterio.

c_1	a_1	a_2
a_1	1	8
a_2	1/8	1

c_2	a_1	a_2
a_1	1	6
a_2	1/6	1

c_3	a_1	a_2
a_1	1	4
a_2	1/4	1

[Tabla 7.5] Importancia relativa entre cada criterio y pares de alternativas

Una vez obtenido las importancias relativas normalizadas entre los criterios, y entre alternativas para cada criterio, es posible construir una matriz Z de desempeño de las alternativas en función de los criterios. Utilizando los valores del ejemplo, dicha matriz tendría la forma mostrada en la [Tabla 7.6].

	c_1	c_2	c_3
a_1	0.89	0.86	0.80
a_2	0.11	0.14	0.20

[Tabla 7.6] Matriz Z de desempeño de las alternativas respecto a los criterios

Es posible ahora determinar qué soluciones permiten un mejor desempeño al tener en cuenta uno, o una combinación de criterios simultáneamente. En el caso del ejemplo, la solución a_1 tenderá a ser la elegida en todos los casos ya que otorga mejor desempeño para todos los criterios. Sin embargo, para casos más complejos como el del planificador de procesos de SODIUM, la matriz Z de desempeño presentará una complejidad mayor, y sus resultados dependerán de los criterios a tener en cuenta.

7.2.3. MÓDULO III

En esta etapa se implementan las funciones adaptativas y acciones adaptativas elementales de cada función para el mecanismo que agrega reglas no previstas al comienzo del proceso de decisión.

Para la construcción del mecanismo adaptativo de la $TDAE$ es preciso haber determinado el método M de decisión multi-criterio por parte del decisor. En base a él se crean las funciones auxiliares FM que serán agregadas a la TDAsubyacente. Estas funciones auxiliares se asocian a las acciones del tipo adición (+) existentes para determinar las mejores alternativas para cada nueva regla, teniendo en cuenta:

- Los ejemplos de la tabla de decisión subyacente.
- Los pesos relativos de los criterios, ordenados de acuerdo a su importancia relativa
- Los valores globales a_x de las alternativas, también ordenados.

7.3. EXTENSIÓN PARA EL PLANIFICADOR DE PROCESOS DE SODIUM

En esta sección se presenta el desarrollo de los tres módulos para la elaboración $TDAE$ para el planificador de procesos reconfigurable de SODIUM, utilizando los elementos para su análisis ya vistos.

7.3.1. MÓDULO I

Para esta etapa de identificación de criterios y alternativas podemos recurrir directamente a la [Tabla 8.4] de la que tomaremos como conjunto de criterios C a las métricas de evaluación de rendimiento de los algoritmos planificadores, y como conjunto de alternativas A a los diferentes modos de planificación:

- $C = \{\%cpu, \#_f, t_{cv}, t_w, t_r, t_s, o_v\}$

- $A = \{RR, RRPR, RRQV, FCFS, SJFS, BTS\}$

Los elementos del conjunto A serán los candidatos a ser utilizados como modo de planificación para las nuevas reglas a agregar, a diferencia del método utilizado en la TDAobtenida en el capítulo anterior en el que se mantenía el modo actual.

7.3.2. MÓDULO II

Como se ha visto, para obtener la matriz Z de desempeño entre combinaciones de criterios y alternativas, es preciso obtener las relaciones entre cada par de criterios y la relación para cada criterio entre pares de alternativas.

La relación entre alternativas para cada criterio del planificador de procesos puede ser obtenido directamente del análisis de la tabla [Tabla 8.2]. Esto puede ser realizado traduciendo la nomenclatura utilizada (+, 0, -, y x) para utilizar la escala fundamental de Saaty donde:

- + equivale a una preferencia moderada (3)
- 0 equivale a una preferencia indiferente (1)
- - equivale a una no preferencia moderada (1/3)
- x equivale a una no preferencia extrema (1/9) en todos los casos

Multiplicando los pares de valores para cada combinación, es posible obtener el valor comparativo entre las diferentes alternativas para cada criterio. La forma de la tabla comparativa de peso entre pares de criterios dependerá de qué categorías de procesos estén en la cola de listos. Se utilizará una combinatoria que decida en función a la cantidad de procesos listos por categoría.

El peso que cada criterio tiene sobre cada categoría ya esta expresado en la tabla la [Tabla 5.2], por lo que sólo bastará realizar la misma traducción a la escala fundamental de Saaty. En la [Tabla 7.9] se muestran los valores de preferencia de cada criterio sobre cada categoría.

	Preferencia del Criterio (métrica)						
Categoría	%cpu	$\#_f$	t_{cv}	t_w	t_r	t_s	o_v
(II)	1	1	1	3	3	1	1
(M)	3	1	1	3	1	1	1
(ES)	1	3	1	1	3	1	1
(WEB)	1	1	1	1	3	3	1
(P)	1	3	3	1	1	1	1
(S)	1	1	1	1	1	1	3

[Tabla 7.9] Valor de preferencia de criterios de cada categoría

La importancia de cada criterio variará según la cantidad de procesos de cada categoría estén ejecutando. Por ejemplo, para un escenario donde se ejecuten 2 procesos de categoría (II), y 1 de categoría (S), la importancia de las métricas t_w y t_r será de 6; la de la métrica o_v será de 3; y la del resto será de 0. Es posible, entonces, elaborar una tabla de preferencia relativa entre criterios. La [Tabla 7.10] contiene los valores que resultarían del ejemplo recién citado.

	Preferencia del Criterio (métrica)						
Criterios	%cpu	$\#_f$	t_{cv}	t_w	t_r	t_s	o_v
%cpu	1	1	1	6	6	1	3
$\#_f$	1	1	1	6	6	1	3
t_{cv}	1	1	1	6	6	1	3
t_w	1/6	1/6	1/6	1	1	1/6	3
t_r	1/6	1/6	1/6	1	1	1/6	3
t_s	1	1	1	6	6	1	3
o_v	1/3	1/3	1/3	1/3	1/3	1/3	1

[Tabla 7.10] Valor de preferencia entre pares de criterios del ejemplo

Se hace evidente que la forma de la [Tabla 7.10] variará constantemente según la cantidad de procesos exista en cada momento y las categorías a las que éstos pertenezcan. Esto permitirá determinar nuevas reglas mucho más fieles al escenario de ejecución actual,

para lidiar con los escenarios de ejecución aún no contemplados. La matriz Z de desempeño de las alternativas respecto a los criterios se obtendría de multiplicar la tabla de preferencias entre pares de criterios con cada tabla de rendimiento comparativo de criterios; para el caso del ejemplo, la [Tabla 7.11] ilustra la tabla resultante ya normalizada.

	Preferencia del Criterio (métrica)							Preferencia Total
Categoría	%cpu	$\#_f$	t_{cv}	t_w	t_r	t_s	o_v	
Multiplicador	x 0	x 0	x 0	x 6	x 6	x 0	x 3	
RR				5,20	5,33		2.53	13.06
RRPR				5.28	5.41		2.56	13.25
RRQV				6.00	6.00		0.00	12.00
FCFS				5.94	5.95		3.00	14.89
SJFS				1.43	2.16		2.99	6.57
BTS				5.94	4.63		2.84	13.42

[Tabla 7.11] Matriz Z de desempeño para el caso del ejemplo

Pude verse en la [Tabla 7.11] que para el caso de ejemplo con dos procesos de categoría (II) y uno de categoría (S), se da una preferencia mayor para el modo de planificación FCFS.

La matriz Z se regenerará completamente en base a la cantidad de procesos por categoría listos para ejecutar toda vez que alguna función adaptativa llame a la función *z_gen()*. Se utilizará la función z_get() para obtener el modo de funcionamiento de mayor preferencia de la matriz Z actual.

7.3.3. MÓDULO III

En esta etapa se deben explicitar las funciones de agregado de nuevas reglas para las condiciones no contemplado en el momento en que se detectan. Como esto ya fue conseguido en la *TDA* representada en la [Tabla 6.6], sólo bastará agregar los llamados a las funciones *z_gen()* y *z_get()* para establecer el valor de la variable *m* que representará el modo al que se deberá transicionar.

Vale recordar que el valor para la variable *m* en el TDAde la [Tabla 6.6], contenía directamente el modo actual de planificación; esto será reemplazado ahora por el valor de preferencia obtenido por la matriz Z.

En la [Tabla 7.12] se puede observar la forma final de la $TDAE$ del planificador de procesos de SODIUM que será utilizada en las pruebas del siguiente capítulo. En verde se indican los elementos agregados.

		Declaración de cada función adaptativa									Reglas													
		Ad1		Ad2							0	1	2	3	…	26	27	28	29	30	31	32	33
		H	?	H	+	+	+	+	+	+	S	R	R	R	R	R	R	R	R	R	R	R	E
Condiciones	Modo Actual				RR	PR	QV	FC	SJ	BT		RR	RR	RR	…	FC	SJ	SJ	BT	BT			
	Estado														…							N	
	II				C_1	C_1	C_1	C_1	C_1	C_1		-	x	-	…	-	-	-	-	-			
	M				C_2	C_2	C_2	C_2	C_2	C_2		x	-	-	…	-	-	-	-	-			
	ES				C_3	C_3	C_3	C_3	C_3	C_3		-	-	-	…	x	x	-	x	-			
	WEB				C_4	C_4	C_4	C_4	C_4	C_4		-	-	-	…	-	-	-	-	-			
	P				C_5	C_5	C_5	C_5	C_5	C_5		-	-	x	…	-	-	-	-	-			
	S				C_6	C_6	C_6	C_6	C_6	C_6		-	-	-	…	-	-	x	-	x			
Acciones Auxiliares	z_gen()																					x	
	z_get()																					m	
Acciones	func()				$\$_m$	$\$_m$	$\$_m$	$\$_m$	$\$_m$	$\$_m$	0	a	b	d	…	q	q	u	x	y			
	Estado=		D												…						N		
Funciones Adaptativas	Ad1	A			x	x	x	x	x	x		x	x	x	x	x	x	x	x	x		x	
	Ad2			B																		x	
Variables	m			V																		x	

[Tabla 7.12] Tabla de decisión adaptativa extendida del planificador de SODIUM

La tabla de decisión adaptativa extendida obtenida en [Tabla 7.12], junto a las funciones *z_gen()* y *z_get()* son el producto final de la propuesta presentada en esta investigación para la aplicación de tecnologías adaptativas en la toma de decisiones para sistemas operativos reconfigurables.

Durante el siguiente capítulo se explicarán las pruebas a las que serán sometidos los tres enfoques para la toma de decisión de modo de planificación de procesos obtenidos en el

transcurso de la investigación: utilizando la tabla de decisión convencional obtenida en [Tabla 5.6], la tabla de decisión adaptativa obtenida en la [Tabla 6.6], y la tabla de decisión adaptativa extendida obtenida en [Tabla 7.12]. Luego, se compararán los resultados para poder definir cuál ofrece mejores resultados respecto a las métricas que se tendrán en cuenta.

8. VALIDACIÓN

8.1. PLAN DE PRUEBAS

El objetivo de las pruebas a realizar será el de medir la eficiencia de la decisión sobre qué modo (algoritmo) utilizar para la planificación de procesos de cada una de las estrategias expuestas en este trabajo. Para ello, se elaborarán tres versiones diferentes del planificador de procesos de SODIUM, cada una implementando una de las siguientes estrategias:

- Utilizando la [Tabla 5.6] de decisión convencional estática (TDC).
- Utilizando la [Tabla 6.6] de decisión adaptativa simple (TDAS).
- Utilizando la [Tabla 7.12] de decisión adaptativa extendida (TDAE)

Como ya se ha analizado en la [Sección 5.2], existen 720 posibles escenarios de ejecución diferentes. Sin embargo, no bastaría ejecutar aisladamente cada escenario para efectuar un análisis exhaustivo del comportamiento de cada versión dado que los resultados también dependen del orden en que cada escenario se presente luego del anterior.

Para obtener un panorama estadísticamente aceptable sin requerir un conjunto de pruebas exhaustiva para todos los posibles caminos, se definirán seis caminos diferentes que servirán de muestra para representar el conjunto completo de escenarios. Cada uno de estos caminos será recorrido por las tres versiones de la misma forma, garantizando una igualdad de condiciones. Los seis resultados de eficiencia para cada recorrido/versión serán finalmente promediados.

Para la obtención de un resultado cuantitativo único indicativo de la eficiencia se utilizará el método indicado en la [Sección 8.4]. Para ello hará falta definir mediciones de control utilizando los resultados de las métricas para cada algoritmo de planificación sobre

escenarios de ejecución de categoría única. Dichos resultados servirán para definir escalas de eficiencia que serán fácilmente comparables con otros.

8.2. AMBIENTE DE PRUEBAS

Todas las pruebas se realizarán creando y ejecutando procesos en el sistema operativo SODIUM. Dichos procesos ocuparán todos inicialmente el mismo tamaño en memoria, y ejecutarán cada uno, un programa correspondiente a su categoría. Cada uno de estos programa representa a cada categoría según lo especificado en la [Tabla 8.1].

Programa	Categoría	Comportamiento
casoII.bin	Aplicaciones de Interacción Intensiva	Ejecución iterativa del syscall read() para leer caracteres ingresados por teclado, y write() para imprimir dicho carácter en pantalla.
casoM.bin	Aplicaciones Multimedia	500 iteraciones de una lectura de 512 bytes de un dispositivo, ejecución de división sobre todos los elementos, impresión en pantalla de la suma de las divisiones.
casoES.bin	Aplicaciones de E/S Intensiva	100 iteraciones de lectura de 100kb de un dispositivo, y escritura de 100kb sobre otro.
casoWEB.bin	Aplicaciones de Internet	200 iteraciones de ejecución del syscall read(), una solicitud de lectura sobre una interfaz de red.
casoP.bin	Aplicaciones de Procesamiento Intensivo	64 millones de operaciones de división.
casoS.bin	Aplicaciones de Sistema	Espera activa durante 15 segundos de ejecución efectiva

[Tabla 8.1] Tabla de categorías de aplicaciones.

Como se explicó en la [Sección 5.2], el kernel de SODIUM no utilizará ningún tipo de información explícita para conocer la categoría de aplicación que se está ejecutando ni utilizará los parámetros pasados a los programas para determinar su categoría, sino que analizará su comportamiento en base al uso de syscalls, y el tiempo de procesamiento entre ellas explicitado en la [Tabla 8.1]. Por lo tanto, se espera que los comportamientos ofrecidos por los programas durante todas las pruebas que se realizarán sean reconocidos perfectamente.

La única excepción será la detección de aplicaciones de sistema, ya que el kernel será el responsable de crear sus procesos, y conocerá su categoría a priori.

Si bien los comportamientos explicitados en la [Tabla 8.1] intentan representar de manera general a todas las posibles aplicaciones que puedan existir de cada categoría, resultaría válido discutir si su nivel de precisión resulta suficiente. Sin embargo, aún si se demostrara que no realizan la mejor representación general posible para cada escenario particular, esto no afecta a lo que se intenta demostrar en esta investigación. Lo que se busca, en cambio, es determinar si los mecanismos de toma de decisión a evaluar permiten definir correctamente los modos de planificación para cada escenario según las reglas y escenarios establecidos, aún si estos no fueran por si mismos suficientes para representar la generalidad de los casos reales.

Todos los escenarios de ejecución serán lanzados en forma de en lote desde el proceso inicial llamado *init.bin*. Se modificará su comportamiento original – iniciar la línea de comandos – por cada escenario de ejecución determinado por cada prueba. De esta manera, se permitirá automatizar las pruebas, y además se elimina la necesidad de contar con una línea de comando para crear cada proceso. Esto resulta indispensable, ya que la línea de comando es por sí misma un proceso de interacción intensiva, lo que afectará a escenarios de prueba que no impliquen la existencia de un proceso de dicha categoría.

8.3. PRUEBAS DE CONTROL

En esta investigación utilizamos pruebas de control para obtener los mejores/peores resultados para cada estadística para definir una escala unificada de 0 a 1, para las pruebas generales. Esta escala nos permitirá sumar, consolidar, y comparar resultados a través de diferentes métricas.

Se obtendrán los valores para cada métrica, de la ejecución de 2 procesos por cada categoría de aplicación, para comparar el rendimiento de los 6 algoritmos de planificación. La siguiente [Tabla 8.2] ilustra los mejores y peores resultados obtenidos por cada algoritmo:

Resultados de Métricas							
Algoritmo	%cpu	$\#_f$	t_{cv}	t_w	t_r	t_s	o_v
Mejores Resultados							
RR	85	0.34	580	67	0.66	1.45	3829
RRPR	98	0.74	510	61	0.60	0.10	3718
RRQV	99	0.73	540	12	0.12	0.10	12073
FCFS	69	0.31	118	16	0.16	0.21	2280
SJFS	52	0.50	125	325	3.24	1.13	2328
BTS	97	0.73	525	16	1.23	46	2789
Peores Resultados							
RR	85	0.23	616	406	3.22	0.48	3829
RRPR	99	0.23	617	409	1.89	0.05	3718
RRQV	95	0.21	554	402	1.22	0.10	12073
FCFS	69	0.21	739	407	1.41	0.10	2280
SJFS	52	0.21	982	423	4.99	0.37	2328
BTS	97	0.22	540	373	1.24	42	2789

[Tabla 8.2] Tabla de mejores y peores resultados de métricas para cada algoritmo

Con los resultados obtenidos en la [Tabla 8.2], podemos extraer el mejor y el peor resultado obtenido para cada métrica. Con ellos podemos definir los límites con los que comparar otros resultados en base a una fórmula de regresión lineal entre los mejores y peores valores para cada métrica. En la [Tabla 8.3] puede verse cómo se evaluarán los resultados para cada métrica:

Métricas	Peor Resultado	Puntaje	Mejor Resultado	Puntaje	Fórmula
%cpu	52	0	99	1	$f(\%_{cpu}) = \frac{\%_{cpu} - 52}{47}$
$\#_f$	0.23	0	0.74	1	$f(\#_f) = \frac{\#_f - 0.23}{0.51}$
t_{cv}	982	0	118	1	$f(t_{cv}) = \frac{-t_{cv} + 982}{864}$
t_w	423	0	12	1	$f(t_w) = \frac{-t_w + 423}{411}$

t_r	4.99	0	0.12	1	$f(t_r) = \frac{-t_r + 4.99}{4.87}$
t_s	0.05	0	46	1	$f(t_s) = \frac{t_s}{46}$
o_v	12073	0	2280	1	$f(o_v) = \frac{-o_v + 12073}{9793}$

[Tabla 8.3] Formulas para normalización de puntajes comparativos entre métricas

Resultados menores que el peor resultado de la tabla [Tabla 8.3], o mayores que el mejor resultado, se normalizan a 0, y 1 respectivamente.

8.4. MEDICIÓN DE EFICIENCIA

Para medir la eficiencia de un algoritmo de planificación respecto a un escenario de ejecución en particular, obtendremos un puntaje normalizado que permita comparar su desempeño respecto a otras combinaciones algoritmo/escenario.

Para el cálculo de la eficiencia e de un algoritmo de planificación respecto a un escenario específico se utilizarán los siguientes elementos:

- $N_{categoría}$ Indica la cantidad de procesos de una categoría en particular para el escenario evaluado. Por ejemplo, N_P indica la cantidad de procesos de categoría procesamiento intensivo.
- Relevancia de cada métrica según la categoría de aplicación según la tabla [Tabla 5.2]. Por ejemplo, para la categoría de procesamiento intensivo (P), sólo son relevantes las métricas Tiempo de Ciclo de Vida (t_{cv}) y Tasa de Finalización ($\#_f$).
- Las mediciones normalizadas para cada métrica obtenidas según las fórmulas de la [Tabla 8.3]. Por ejemplo, $f(\%_{cpu})$ es la medición obtenida para la métrica Uso de CPU ($\%_{cpu}$).

Utilizando estos elementos, la fórmula para calcular el puntaje de eficiencia bruta (e_{bruta}) de un algoritmo para un escenario en particular es:

$$e_{bruta} = N_{II} * [f(\mathrm{t_w}) + f(\mathrm{t_r})] + N_m * [f(\mathrm{t_w}) + f(\%_{cpu})] + N_{ES} * [f(\#_\mathrm{f}) + f(\mathrm{t_r})]$$

$$+N_{WEB} * [f(\mathrm{t_s}) + f(\mathrm{t_r})] + N_P * [f(\mathrm{t_{cv}}) + f(\#_\mathrm{f})] + N_S * 2f(\mathrm{o_v})$$

Para calcular la eficiencia neta (e) de manera independiente de la cantidad de procesos y de categorías faltantes, es preciso dividirla por la cantidad total de procesos (N_{total}) utilizada:

$$e = \frac{e_{bruta}}{2 * N_{total}}$$

También se la divide a la mitad, ya que se utilizan dos métricas por categoría, para que el resultado de la eficiencia neta resulte siempre entre 0 y 1. El resultado numérico obtenido para e permitirá comparar resultado de la ejecución de cualquiera de los algoritmos en cualquier escenario.

8.5. EVALUACIÓN DE ALGORITMOS RESPECTO A CATEGORÍAS

Para poder definir la [Tabla 5.6] de decisión convencional subyacente que forma la base para todos los métodos propuestos, se midió la eficiencia de cada uno de los algoritmos respecto a cada categoría de procesos.

	Categoría						
Algoritmo	II	M	ES	WEB	P	S	Promedio
RR	0.85	0.38	0.29	0.53	0.27	0.84	0.52
RRPR	0.85	0.53	0.85	0.53	0.27	0.85	0.64
RRQV	0.99	0.52	0.88	0.64	0.24	0.01	0.54
FCFS	0.86	0.34	0.39	0.57	0.50	0.99	0.60
SJFS	0.07	0.07	0.01	0.17	0.73	0.95	0.33
BTS	0.98	0.49	0.79	0.67	0.26	0.94	0.68

[Tabla 8.4] Eficiencia de cada algoritmo respecto a cada categoría

Se utilizaron los mismos resultados obtenidos para las pruebas de control de la [Sección 8.2], y utilizando las fórmula para el cálculo de eficiencia neta, discriminando por categoría. En la tabla [Tabla 8.4] se muestran los resultados obtenidos.

8.6. PRUEBAS GENERALES

Para las pruebas generales, se establecieron seis recorridos (denominados 1, 2, 3, 4, 5, y 6). Cada recorrido pasa por un escenario de categoría única, y luego agrega procesos de otras categorías escalonadamente, hasta que llega al escenario donde procesos de todas las categorías se encuentran en ejecución.

Inicialmente, cada recorrido crea dos procesos de una categoría, evalúa las métricas, y crea otros dos nuevos procesos de otra categoría, hasta llegar a los 12 procesos simultáneos, con los que completa todas las categorías. Los caminos de cada recorrido se especifican en la [Tabla 8.5].

Recorrido	Categoría Inicial	Camino				
1	(II)	(S)	(P)	(WEB)	(ES)	(M)
2	(M)	(II)	(S)	(P)	(WEB)	(ES)
3	(ES)	(M)	(II)	(S)	(P)	(WEB)
4	(WEB)	(ES)	(M)	(II)	(S)	(P)
5	(P)	(WEB)	(ES)	(M)	(II)	(S)
6	(S)	(P)	(WEB)	(ES)	(M)	(II)
Cantidad de Procesos	2	4	6	8	10	12

[Tabla 8.5] Tabla de recorridos para las pruebas generales

Cada versión implementada del planificador de procesos utilizará dichos recorridos pre-establecidos para la evaluación de su eficiencia (e).

Por cada prueba, se obtendrán seis valores de e para cada versión del planificador a probar, uno por cada recorrido. La eficiencia total (e_t) para cada planificador se obtendrá de la sumatoria de cada e obtenido para cada escenario:

$$e_t = \frac{\sum_{i=1}^{6} e_i}{6}$$

La eficiencia total (e_t) de un recorrido en particular dará por resultado un número entre 0 y 1. Un resultado ideal de 1 significará que el algoritmo de planificación elegido por el mecanismo de toma de decisión fue ideal para cada escenario de ejecución evaluado.

Los resultados de las pruebas realizadas sobre los tres métodos de toma de decisión implementados para el planificador de procesos de SODIUM, utilizando la tabla de decisión convencional (TDC), la tabla de decisión adaptativa simple (TDAS), y la tabla de decisión adaptativa extendida (TDAE), para cada uno de los seis recorridos descriptos se muestran en la serie de [Tablas 8.6].

A fin de poder comparar no sólo la eficiencia entre los métodos de toma de decisión mencionados entre sí, sino también conocer su capacidad de seleccionar el mejor planificador para cada caso, se agregan los resultados obtenidos en los casos ideales. El caso ideal de cada recorrido indicará qué eficiencia es la máxima que podría obtenerse y qué algoritmo de planificación se debería haber elegido para obtenerla.

Los casos ideales se obtienen a partir de ejecutar pruebas manuales para cada paso de cada recorrido con todos los algoritmos de planificador disponibles, y seleccionar siempre el de mejor rendimiento. Esto nos permitirá evaluar la capacidad intrínseca de cada método de toma de decisión y establecer finalmente si resultan o no convenientes para la selección de algoritmos de planificación.

Recorrido 1		Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
TDC	Eficiencia	1,00	0,65	0,60	0,62	0,77	0,70
	Algoritmo	RRQV	BTS	BTS	BTS	BTS	BTS
TDAS	Eficiencia	1,00	0,68	0,61	0,63	0,69	0,55
	Algoritmo	RRQV	RRQV	RRQV	RRQV	RRQV	RRQV
TDAE	Eficiencia	1,00	0,69	0,65	0,64	0,77	0,70
	Algoritmo	RRQV	FCFS	FCFS	FCFS	BTS	BTS
Caso Ideal	Eficiencia	1,00	0,69	0,67	0,70	0,77	0,70
	Algoritmo	RRQV	FCFS	RR	RRPR	BTS	BTS
Cantidad de Categorías		1	2	3	4	5	6

Recorrido 2		Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
TDC	Eficiencia	0,88	0,85	0,77	0,59	0,65	0,70
	Algoritmo	RRPR	BTS	BTS	BTS	BTS	BTS
TDAS	Eficiencia	0,88	0,92	0,76	0,58	0,60	0,65
	Algoritmo	RRPR	RRPR	RRPR	RRPR	RRPR	RRPR
TDAE	Eficiencia	0,88	0,86	0,77	0,60	0,65	0,70
	Algoritmo	RRPR	RRQV	BTS	FCFS	BTS	BTS
Caso Ideal	Eficiencia	0,88	0,92	0,77	0,60	0,65	0,70
	Algoritmo	RRPR	RRPR	BTS	FCFS	BTS	BTS
Cantidad de Categorías		1	2	3	4	5	6

Recorrido 3		Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
TDC	Eficiencia	0,90	0,77	0,86	0,84	0,71	0,70
	Algoritmo	RRQV	BTS	BTS	BTS	BTS	BTS
TDAS	Eficiencia	0,90	0,81	0,88	0,74	0,55	0,55
	Algoritmo	RRQV	RRQV	RRQV	RRQV	RRQV	RRQV
TDAE	Eficiencia	0,90	0,81	0,88	0,84	0,71	0,70
	Algoritmo	RRQV	RRQV	RRQV	BTS	BTS	BTS
Caso Ideal	Eficiencia	0,90	0.88	0.91	0,84	0,71	0,70
	Algoritmo	RRQV	RRPR	RRPR	BTS	BTS	BTS
Cantidad de Categorías		1	2	3	4	5	6

Recorrido 4		Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
TDC	Eficiencia	0,65	0,77	0,64	0,73	0,73	0,70
	Algoritmo	BTS	BTS	BTS	BTS	BTS	BTS
TDAS	Eficiencia	0,65	0,77	0,64	0,73	0,73	0,70
	Algoritmo	BTS	BTS	BTS	BTS	BTS	BTS
TDAE	Eficiencia	0,65	0,78	0,64	0,75	0,73	0,70
	Algoritmo	BTS	RRQV	RRQV	RRQV	BTS	BTS
Caso Ideal	Eficiencia	0,65	0,78	0.80	0,75	0.81	0,70
	Algoritmo	BTS	RRQV	RRPR	RRQV	RRPR	BTS
Cantidad de Categorías		1	2	3	4	5	6

Recorrido 5		Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
TDC	Eficiencia	0,74	0,55	0,61	0,55	0,66	0,70
	Algoritmo	SJFS	BTS	BTS	BTS	BTS	BTS
TDAS	Eficiencia	0,74	0,57	0,46	0,29	0,28	0,34
	Algoritmo	SJFS	SJFS	SJFS	SJFS	SJFS	SJFS
TDAE	Eficiencia	0,74	0,63	0,73	0,58	0,70	0,70
	Algoritmo	SJFS	FCFS	RRQV	RRQV	RRQV	BTS
Caso Ideal	Eficiencia	0,74	0.67	0,73	0.60	0,70	0,70
	Algoritmo	SJFS	RR	RRQV	RRPR	RRQV	BTS
Cantidad de Categorías		1	2	3	4	5	6

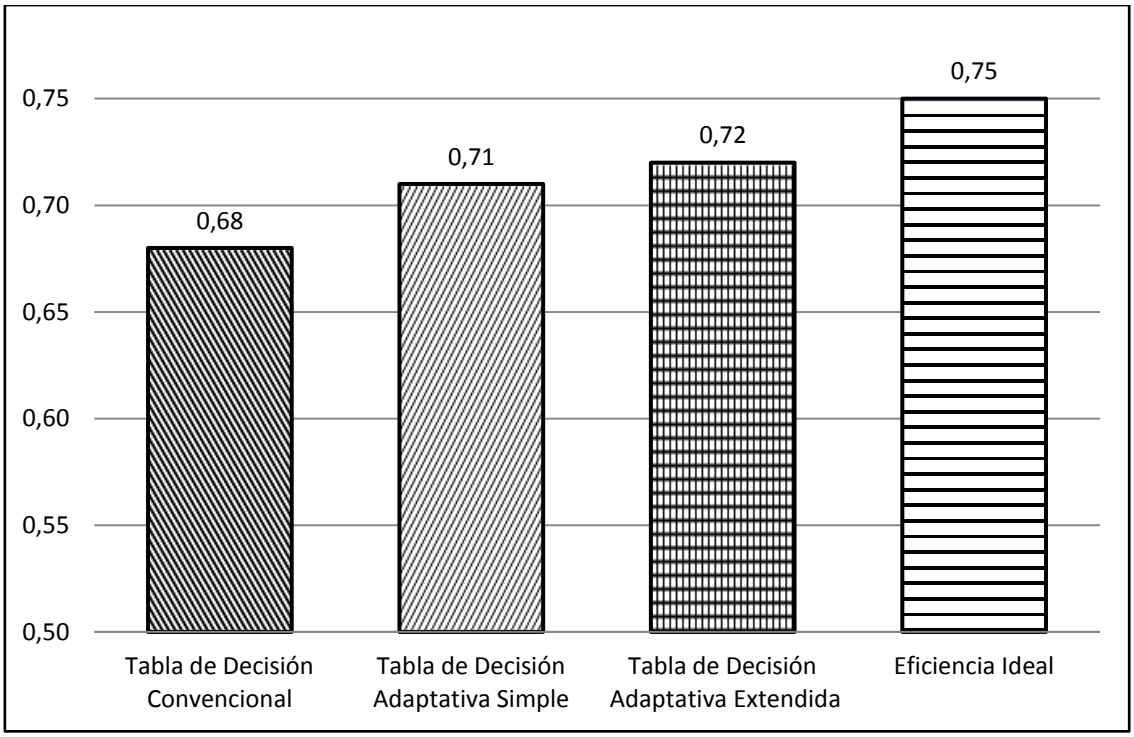
Recorrido 6		Paso 1	Paso 2	Paso 3	Paso 4	Paso 5	Paso 6
TDC	Eficiencia	0,20	0,48	0,51	0,69	0,62	0,70
	Algoritmo	FCFS	BTS	BTS	BTS	BTS	BTS
TDAS	Eficiencia	0,20	0,51	0,55	0,55	0,48	0,48
	Algoritmo	FCFS	FCFS	FCFS	FCFS	FCFS	FCFS
TDAE	Eficiencia	0,20	0,57	0,55	0,69	0,62	0,70
	Algoritmo	FCFS	SJFS	FCFS	BTS	BTS	BTS
Caso Ideal	Eficiencia	0,20	0,57	0.57	0,69	0,62	0,70
	Algoritmo	FCFS	SJFS	RRPR	BTS	BTS	BTS
Cantidad de Categorías		1	2	3	4	5	6

[Tablas 8.6] Resultados para cada uno de los seis recorridos para cada método de toma de decisión implementado.

8.7. INTERPRETACIÓN DE RESULTADOS

Para los escenarios de ejecución en que sólo se presentan procesos de una única categoría de aplicación, los tres métodos obtuvieron los mismos resultados en cada recorrido y también en promedio (eficiencia = 0,73). Esto se debe a que este tipo de escenarios ya está contemplado en las reglas existentes de la TDC sobre la que los tres métodos se basan de igual manera, por lo que utilizan siempre el mismo modo de planificación. Además, este tipo de escenarios obtiene siempre la mayor eficiencia ya que se basa en las heurísticas de la [Tabla 8.4] para escoger el mejor planificador para la categoría en ejecución. La eficiencia ideal también coincide con ellos dado que no existen mejores planificadores para ninguno de los casos que los seleccionados por los métodos de toma de decisión implementados.

Para las pruebas con 2 categorías de aplicación simultáneas, pueden observarse las primeras diferencias en la eficiencia de los tres métodos y su comparación respecto al caso ideal. En la [Figura 8.1] puede observarse que los tres métodos de decisión presentan una eficiencia promedio menor a la ideal.

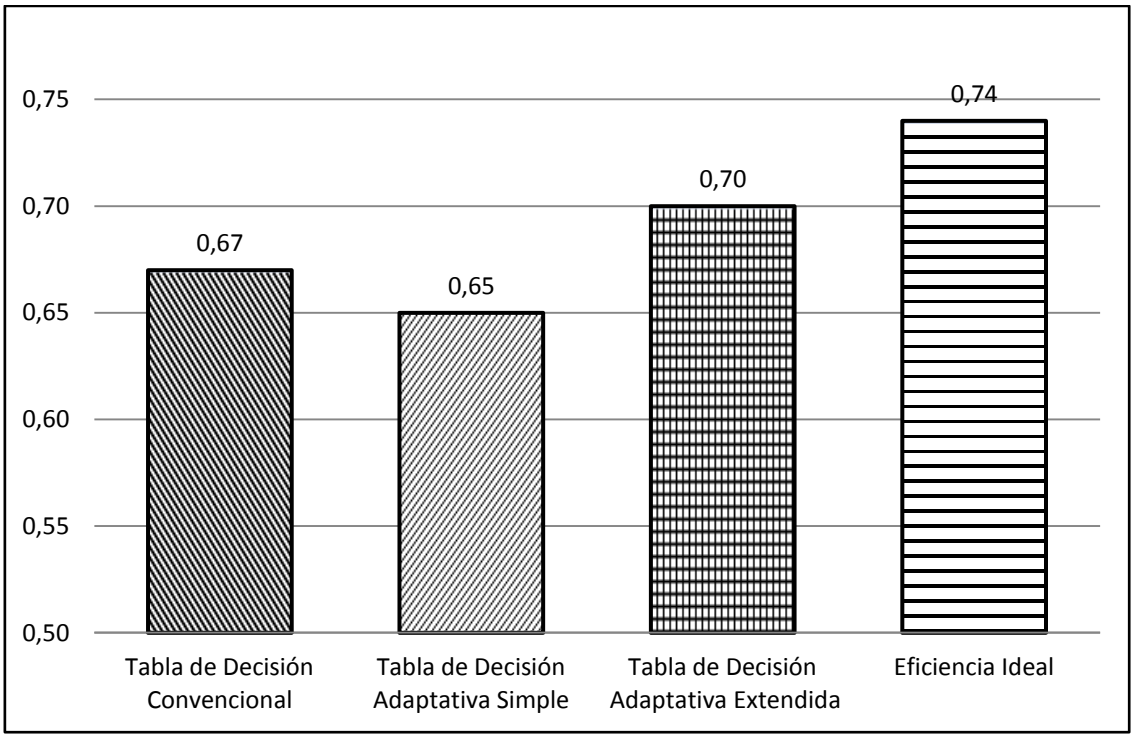


[Figura 8.1] Promedio de eficiencia para pruebas con 2 categorías de aplicación (4 procesos)

En este caso, las tablas de decisión adaptativas presentan un mayor rendimiento respecto a la convencional. Esto confirma que la tendencia de la tabla convencional de utilizar el algoritmo de mayor generalidad apenas se detecta más de una categoría, le resta potencial frente a casos de mayor especificidad. En ese mismo sentido el rendimiento de las TDAS y TDAE consisten en las primeras pruebas de que son capaces de mejorar autónomamente el comportamiento de la TDC subyacente.

Para las pruebas con 3 categorías de aplicación simultáneas puede observarse una mayor diferencia entre los métodos de toma de decisión y el caso ideal. En la [Figura 8.2] se muestra que la diferencia entre los métodos de toma de decisión y el caso ideal se amplía aún

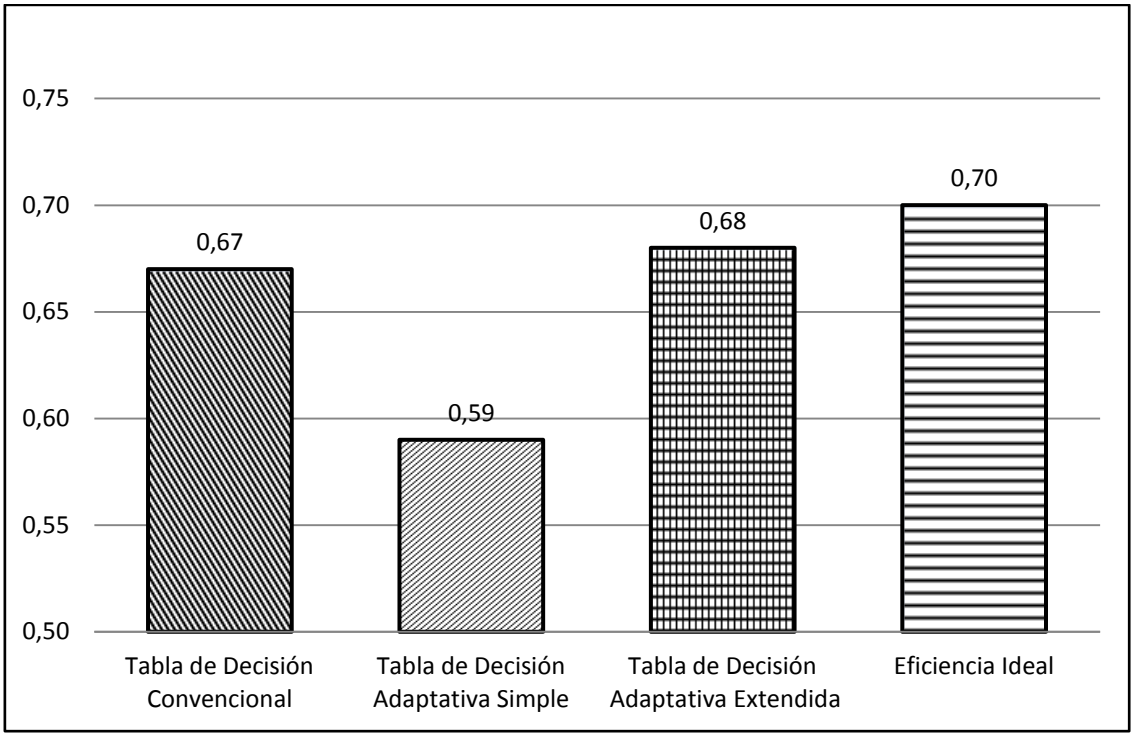
más. Esto sucede debido a que la combinación de métricas para cada categoría no es aún ni tan simple como para determinar con mayor certeza un planificador específico, ni tan diversa como para tender a la selección de un planificador más general (por ejemplo: BTS).



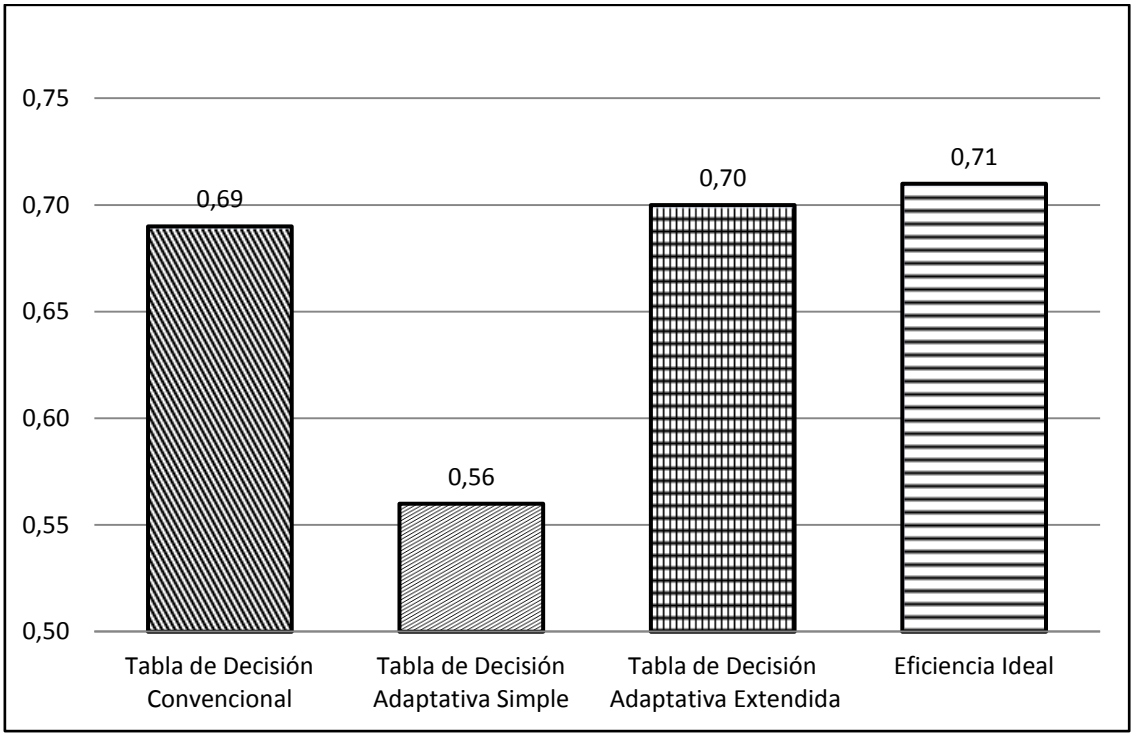
[Figura 8.2] Promedio de eficiencia para pruebas con 3 categorías de aplicación (6 procesos)

A partir de esta cantidad de categorías, la TDAS registra un menor rendimiento que la TDC debido a que, al implementar un mecanismo simple que favorece los casos específicos, pierde capacidad de seleccionar planificadores generales al aproximarse a escenarios de ejecución con mayor diversidad de categorías.

La TDAE, por otro lado, se mantiene aún en el punto más cercano al caso ideal. Esto permite comprobar que su capacidad adaptativa permite obtener buenos resultados tanto en casos específicos – 2 o 3 categorías – como, se verá a continuación, en generales – 4 a 6 categorías –.



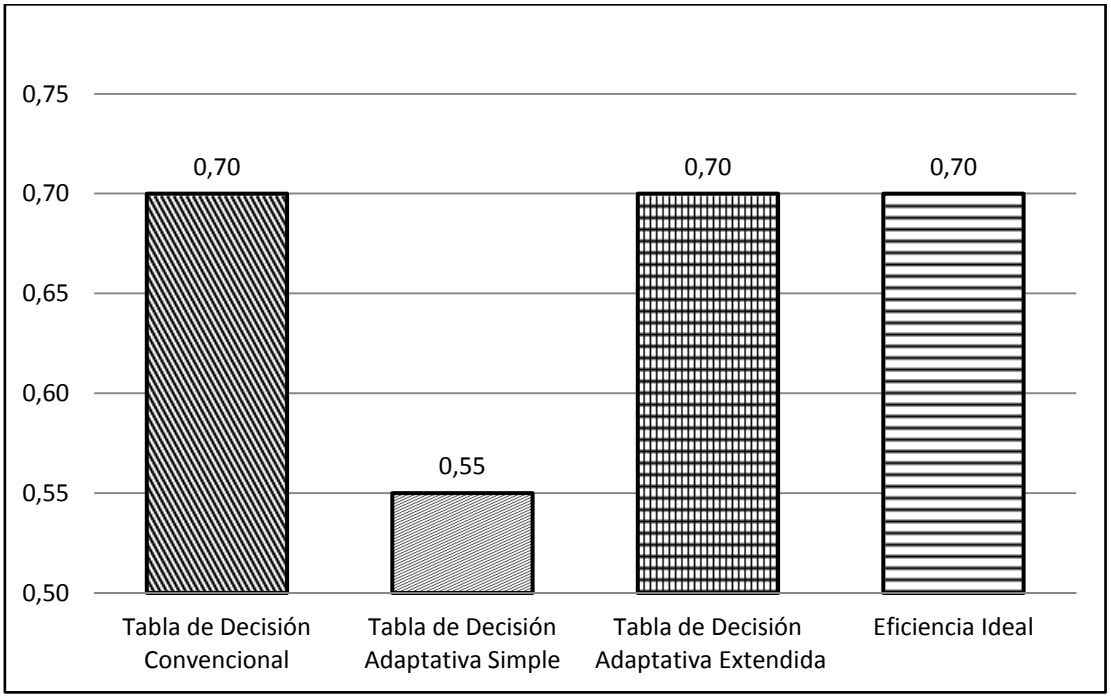
[Figura 8.3] Promedio de eficiencia para pruebas con 4 categorías de aplicación (8 procesos)



[Figura 8.4] Promedio de eficiencia para pruebas con 5 categorías de aplicación (10 procesos)

Puede observarse en la [Figura 8.3] y en la [Figura 8.4] que lo observado para el rendimiento para casos de 3 categorías se mantiene para los casos de 4 y 5 categorías. La TDAS pierde capacidad de establecer algoritmos de planificación generales y ve reducida su eficiencia, mientras que la TDC, al mantener constante su selección de un planificador general, va aproximándose al caso ideal. La TDAE apenas supera el rendimiento de la TDC en esos casos, pero es capaz de detectar casos excepcionales que requieran planificadores específicos y que permitan obtener un mayor rendimiento.

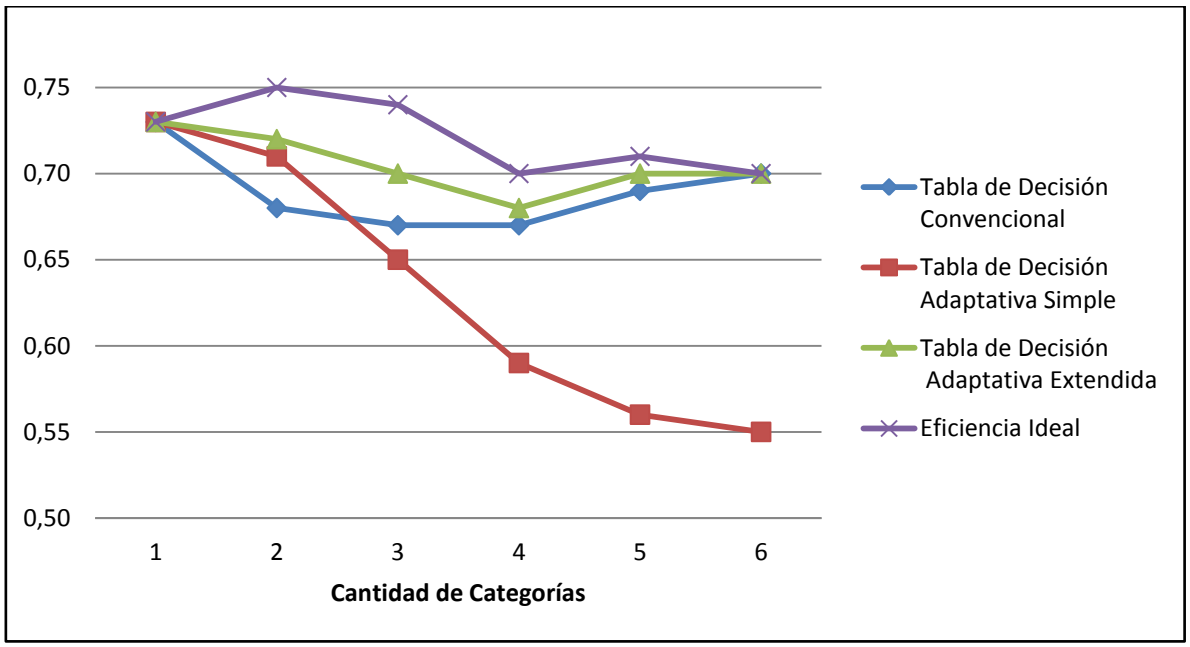
Como puede verse en la [Figura 8.5], para los casos de máxima generalidad con 6 categorías de aplicación, la decisión de elección del planificador también puede ser obtenida a partir de las heurísticas tomadas anteriormente. Tanto TDC y TDAE seleccionan directamente el planificador de mayor generalidad. Sin embargo, como se ha dicho, la TDAS encuentra su punto más bajo de rendimiento debido a su incapacidad de contemplar el uso de planificadores generalizados.



[Figura 8.5] Promedio de eficiencia para pruebas con 6 categorías de aplicación (12 procesos)

En la [Figura 8.6] se muestra un gráfico de evolución para el promedio de eficiencia de los diferentes métodos de toma de decisión implementados respecto al caso ideal. La eficiencia ideal muestra el máximo hipotético que cada método podría obtener.

Puede observarse que la TDAE obtiene los mejores resultados para todos los casos debido a su capacidad de combinar el potencial ofrecido por la TDAS para casos específicos, y el de la TDC para la generalización. Esto demuestra que la TDAE implementada como mecanismo adaptativo para la selección de algoritmos puede obtener resultados cercanos a los ideales para todo tipo de escenarios.



[Figura 8.6] Gráfico de la evolución de la eficiencia entre los diferentes métodos de toma de decisión y el caso ideal.

La decisión a priori de implementar un mecanismo simplista para la TDAS resulta acertada solo considerando la eficiencia obtenida de la TDAE que en base a ella puede ser construida. La TDAS implementa el primer mecanismo básico para el agregado de nuevas

reglas, y la especialización de las decisiones en base al comportamiento de los usuarios. Su bajo rendimiento respecto a TDC para casos más generales no debe ser visto como una degradación de la capacidad de decisión, ya que sirve de base para que los agregados propios de la TDAE corrijan y superen dichos resultados. A pesar de ser un producto intermedio, los resultados obtenidos por la TDAS son igualmente presentados para demostrar las distintas etapas de evolución desde una TDC hacia una más eficiente TDAE.

9. CONCLUSIONES

En el presente trabajo se pudo implementar con éxito un mecanismo para la toma de decisión sobre la configuración de un sistema operativo reconfigurable, mediante la aplicación de tecnologías adaptativas. Las decisiones tomadas por el dispositivo utilizado permitieron obtener una eficiencia cercana a la ideal para el aspecto de selección de planificadores de procesos.

Este enfoque para la toma de decisión en la reconfiguración de los aspectos ofrece una alternativa respecto a aquellas implementadas en los sistemas operativos analizados – tales como Synthetix, SPIN, Kea, y Apertos – en los que se depende de información explícita de la aplicación o el usuario.

Otra de las principales ventajas de haber utilizado este método, es que pudimos contar con información desde el sistema que quizás sería inaccesible para los usuarios y aplicaciones, y pudimos procesarla en tiempo real para obtener decisiones complejas rápidamente. Depender de mecanismos manuales o explícitos para la reconfiguración jamás permitiría utilizar estos datos eficientemente.

Dada la simplicidad en el diseño y representación de las tablas de decisión, la aplicación de tecnologías adaptativas resultó fácil de aplicar sobre un aspecto reconfigurable de SODIUM, y provee un formato entendendible para quien deba modificar su comportamiento en un futuro.

Utilizar este tipo de métodos de toma de decisión automáticos releva a los programadores de aplicaciones de la responsabilidad de manejar la reconfiguración del sistema operativo, lo que conlleva dos ventajas: por un lado, los programadores no están

obligados a conocer los pormenores del sistema operativo y sus complejidades, y aún así se benefician de la reconfiguración; por el otro, se abre las puertas a las ventajas de la reconfiguración a sistemas legados o aplicaciones existentes desarrolladas sin la intención explícita de reconfigurar algún aspecto.

Sin dudas, los sistemas operativos reconfigurables con interfaces explícitas cuyos programadores de aplicaciones dominen al detalle, obtendrán los mejores niveles de eficiencia. Sin embargo, los resultados obtenidos con los mecanismos de decisión adaptativos automáticos sobre la planificación de procesos muestran que la eficiencia se mantiene en niveles altos, a pesar de contar únicamente con información implícita, y con aplicaciones agnósticas de su capacidad reconfigurable.

Puede observarse que, como las tablas de decisión adaptativas son procesadas por la computadora misma, cuanto más complejo sea el sistema al que se lo aplique, mayores beneficios se obtendrán. Como contrapartida, la construcción de un sistema operativo reconfigurable eficiente, y sus mecanismos de toma de decisión requieren un trabajo de diseño e implementación adicionales que, quizás, en algunos ambientes de ejecución pueda resultar excesivo. Por eso, este enfoque debe ser evaluado según su costo/beneficio. A priori, podría definirse que los sistemas operativos más generales o complejos podrían beneficiarse más con mecanismos adaptativos que aquellos muy especializados para ciertos ambientes en particular.

9.1. CONTRIBUCIONES

- La metodología mostrada en este trabajo puede ser aplicada para cualquier aspecto de cualquier sistema operativo comercial o académico. Los diseñadores que la apliquen podrán definir métodos de toma de decisión para la reconfiguración independientes del usuario y las aplicaciones.

- Con este enfoque, es posible desarrollar sistemas operativos que se comporten de manera convencional respecto a los usuarios, pero que se reconfigure de manera automática para mejorar el rendimiento global.

- Los sistemas operativos reconfigurables actuales que implementen mecanismos como los propuestos en este trabajo, podrán mejorar la eficiencia general del sistema, aún si las aplicaciones que se ejecuten son heredadas, o agnósticas de su capacidad reconfigurable. Esto amplía la capacidad de retro-compatibilidad y portabilidad del sistema.

- Se ha contribuido, mediante un ejemplo real, a demostrar el potencial de las tecnologías adaptativas a contribuir en la toma de decisión de problemas complejos y cambiantes de manera autónoma.

9.2. TRABAJOS FUTUROS

Aunque el presente trabajo cumple con presentar un método para la toma de decisiones para aspectos de sistemas operativos reconfigurables, sólo abarca una pequeña parte de los posibles desarrollos que pueden realizarse. Otros aspectos, como la administración de memoria, la administración de entrada/salida, la gestión de la energía, entre otros, podrían ser analizados y sometidos a prueba utilizando mecanismos adaptativos. La obtención de la eficiencia en otros aspectos permitirá tener una mejor idea sobre la eficiencia en general de los mecanismos adaptativos, y conocer mejor cuáles son las áreas en las que mejor y peor se desempeña.

Más allá de que, en el marco de este trabajo, el método de decisión adaptativo ha sido aplicado a un único aspecto de SODIUM, podría ampliarse su uso al resto de los aspectos reconfigurables de este sistema operativo. Incluso, se podrían aplicar tablas de decisión adaptativas para decisiones de mayor granularidad. Por ejemplo, se podrían definir, incluso, las prioridades para procesos según información de uso y comportamiento, utilizando tablas de decisión adaptativas extendidas.

Podría evaluarse, también, la posibilidad de aplicar mecanismos adaptativos a planificadores de procesos con soporte para multi-procesamiento – es decir, con más de un procesador –. La complejidad inherente a la concurrencia entre varios procesos podría requerir una tabla de decisión extendida mucho más compleja, aunque, a su vez, podría arrojar beneficios mucho mayores.

Nuevas investigaciones podrían contemplar alternativas al uso de tablas de decisión adaptativas para la toma de decisión. Existen trabajos que utilizan otros mecanismos, como, por ejemplo en [Alfenas – 2012] se propone el uso de sistemas de Markov adaptativos para la toma de decisiones, y en [Santos – 2012] se propone el uso de autómatas adaptativos. Sería relevante investigar si dichos mecanismos pueden presentar mejor eficiencia o facilidad de implementación en aspectos específicos de los sistemas operativos reconfigurables.

Para los investigadores en el área de sistemas operativos, la capacidad de selección del mejor modo de implementación para un aspecto puede no ser la única aplicación posible de tecnologías adaptativas. Podría evaluarse utilizar dispositivos adaptativos para la configuración interna de cada modo, o la configuración interna del kernel, Estas opciones podrían resultar incluso potencialmente benéficas para sistemas operativos convencionales (no reconfigurables) ya que la toma de decisiones es una necesidad en muchos niveles de su configuración interna.

Fuera del área de sistemas operativos, la tarea de elegir un mejor algoritmo, entre varias alternativas, en base a múltiples criterios, o la necesidad de generar reglas de comportamiento para casos difíciles de contemplar de antemano en problemas de la ingeniería de software, representan un campo ideal para contemplar la aplicación de tecnologías adaptativas, de la misma manera que fueron aplicadas en este trabajo.

Quedará, para desarrollos futuros, incluir en las tablas de decisión adaptativas obtenidas, mecanismos que permitan detectar cuando las nuevas reglas creadas ya no responden de manera eficiente a los nuevos escenarios de ejecución y puedan ser eliminadas y reemplazadas por nuevas reglas actualizadas. Esto requerirá un exhaustivo estudio para establecer los criterios y los eventos que desencadenen la necesidad de corregir el comportamiento de la tabla de decisión.

BIBLIOGRAFÍA

[Alfenas – 2012] D. A. Alfenas, D. P. Shibata, M. R. Pereira-Barretto, J. J. Neto,, "Sistemas de Markov Adaptativos: Formulação e Plataforma de Desenvolvimento". *6° Workshop de Tecnologia Adaptativa*. San Pablo, Brasil, 2012.

[Bershad – 1995] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, S. Eggers, "Extensibility: Safety and Performance in the SPIN Operating System". *5th Symposium on Operating Systems Principles.* ACM, New York, Estados Unidos, 1995.

[Burshteyn – 1990] B. Burshteyn, "Generation and recognition of formal languages by modifiable grammars". *ACM SIGPLAN Notices.* 1990.

[Casas – 2007] N. Casas, M. Cortina, G. De Luca, "Desarrollo de un sistema operativo didáctico". *XIII Congreso Argentino de Ciencia de la Computación*. Chaco, Argentina, 2007.

[Casas – 2008] N. Casas, G. De Luca, M. Cortina, G. Puyo, W. Valiente, "Implementación de distintos tipos de memoria en un sistema operativo didáctico". *XIV Congreso Argentino de Ciencia de la Computación*. La Rioja, Argentina, 2008.

[Casas – 2011] N. Casas, G. De Luca, S. Martin, G. Puyo, W. Valiente, "Gestión de energía en el sistema operativo didáctivo utilizando el modelo APM". *XIII Workshop de Investigadores en Ciencias de la Computación*. Santa Fé, Argentina, 2011.

[Casas – 2012] S. Martin, N. Casas, G. De Luca, J. J. Neto, "Utilización de tecnologías adaptativas para la gestión de la energía de un sistema operativo didáctico". *XIV Workshop de Investigadores en Ciencias de la Computación*. Misiones, Argentina, 2012.

[Cereda – 2009] P. R. M. Cereda, R. A. Gotardo, S. D. Zorzo, "Recomendação de Recursos utilizando Autômato Adaptativo". *3° Workshop de Tecnologia Adaptativa*. Escola Politécnica, USP, San Pablo, Brasil, 2009.

[Chaves – 2012] A. Chaves, P. S. Cugnasca, J. J. Neto, "Adaptative Search with multiple Unmanned Aerial Vehicles (UAVs)". *6° Workshop de Tecnologia Adaptativa*. Escola Politécnica, USP, San Pablo, Brasil, 2012.

[Contier – 2012] A. Contier, D. Padovani, J. J. Neto, "Linguístico: Uma Proposta de Reconhecedor Gramatical Usando Tecnologia Adaptativa". *Memorias del 6° Workshop de Tecnologia Adaptativa*. San Pablo, Brasil, 2012.

[Cowan – 1996] C. Cowan, T. C. Autrey, C. Krasic, C. Pu, J. Walpole, "Fast concurrent dynamic linking for an adaptive operating system". *3rd*

International Conference on Configurable Distributed Systems. Vancouver, Canada, 1996.

[Cozman – 2012] F.G. Cozman, M. R. Pereira-Barretto, W. J. Fuks, "Adaptability in Recommendation Systems: perspectives". *6° Workshop de Tecnologia Adaptativa*. San Pablo, Brasil, 2012.

[Dizeró – 2010] W. J. Dizeró, J. J. Neto, "Uso de adaptatividade na modelagem de cursos para software educacional". *4° Workshop de Tecnologia Adaptativa*. Escola Politécnica, USP, San Pablo, Brasil, 2010.

[Fassino – 2002] J. Fassino, J. Stefani, G. Muller, "Think: A Software Framework for Component-based Operating System Kernels".
USENIX 2002 Annual Conference. California, Estados Unidos, 2002.

[Folliot – 1998] B. Folliot, I. Piumarta, F. Ricardi, "A Dynamically Configurable, Multi- Language Execution Platform". *SIGOPS European Workshop*, 1998.

[Hughes – 1968] M. L. Hughes, R. M. Shank, E. S. Stein, "Decision Tables". *Midi Publications, Management Development Institute, Dviisions of Information.* Pennsylvania, Estados Unidos, 1968.

[Lea – 1995] R. Lea, Y. Yokote, J. Itoh. "Adaptive operating system design using reflection". *Object-Based Parallel and Distributed Computation.* Volumen 1107, Springer Berlin, 1996.

[Martin – 2012] S. Martin, N. Casas, G. De Luca, "Diseño de un sistema operativo reconfigurable para fines didácticos y prácticos". *6° Workshop de Tecnologia Adaptativa*. San Pablo, Brasil, 2012.

[Massalin – 1990] H. Massalin, C. Pu, "Fine-Grain Adaptive Scheduling Using Feedback". *Computing Systems*, Volumen 3, Número 1. 1990.

[McDougall – 2007] R. McDougall, J. Mauro, "Solaris Internals". *Segunda Edición.* Prentice-Hall. California, Estados Unidos, 2007.

[Neto – 1994] J. J. Neto. "Adaptive automata for context-dependent languages" *ACM SIGPLAN Notices*. 1994.

[Neto – 2001] J. J. Neto, "Adaptative rule-driven devices - general formulation and a case study". *Sixth International Conference on Implementation and Application of Automata.* Pretoria, Sudáfrica, 2001.

[Pedrazzi – 2005] T. Pedrazzi, A. Tchemra, R. Rocha, "Adaptive Decision Tables A Case Study of their Application to Decision-Taking Problems". *Adaptive and Natural Computing Algorithms*, Springer. Vienna, Austria, 2005.

[Piumarta – 2000] I. Piumarta, B. Folliot, L. Seinturier, C. Baillarguet, C. Khoury, "Highly Configurable Operating Systems: The VVM Approach". *Workshop on Object Orientation and Operating Systems*, Cannes, Francia, 2000.

[Polakovic – 2007] J. Polakovic, S. Mazaré, J. Stefani, P-C. David, "Experience with implementing safe reconfigurations in component based embedded systems". *10th international ACM SIGSOFT symposium on component-based software engineering*. Boston, Massachusetts, Estados Unidos, 2007.

[Rocha – 2000] R. L. E. Rocha, J. J. Neto, "Autômato adaptativo, limites e complexidade em comparação com máquina de Turing". Congress of Logic Applied to Technology - LAPTEC 2000. San Pablo, 2000.

[Ryckeboer – 2008] H. Ryckeboer, N. Casas, G. De Luca, "Construcción de un Sistema Operativo Didáctico". *X Workshop de Investigadores en Ciencias de la Computación*. La Pampa, Argentina, 2008.

[Russinovich – 2009] M. E. Russinovich, D. A. Solomon, A. Ionescu, "Windows Internals". *5ta Edición*. Microsoft Press. Washington, Estados Unidos, 2009.

[Saaty – 1991] T. L. Saaty, "Método de Análise Hierárquica". McGraw-Hill. San Pablo, Brasil, 1991.

[Santos – 2012] I. Santos, C. Cugnasca, "Autômato Adaptativo para definição autônoma do intervalo de amostragem de dados em Rede de Sensores Sem Fio". *6° Workshop de Tecnologia Adaptativa*. San Pablo, Brasil, 2012.

[Senart – 2002] A. Senart, O. Charra, J. Stefani, "Developing dynamically reconfigurable operating system kernels with the THINK component

architecture". W*orkshop on Engineering Context-aware Object-Oriented Systems and Environments*. Washington, Estados Unidos, 2002.

[Shutt – 1995] J.N Shutt. "Self-modifying finite automata - Power and limitations". *Reporte Técnico*. Worcester Polytechnic Institute, Massachusetts, Estados Unidos, 1995.

[Silberschatz – 2012] A. Silberschatz, P. B. Galvin, G. Gagne. "Operating System Concepts". *8va Edición*. John Wiley & Sons, New Jersey, Estados Unidos, 2012.

[Stange – 2011] R. L. Stange, J. J. Neto, "Aprendizagem Incremental usando Tabelas Decisão Adaptativas". *5° Workshop de Tecnologia Adaptativa*. Escola Politécnica, USP, San Pablo, Brasil, 2011.

[Stonebraker – 1981] M. Stonebraker, "Operating system support for database management". *Communications of the ACM*, 24(7). ACM New York, Estados Unidos, 1981.

[Tchemra – 2008] A. H. Tchemra, "Aplicação da Tecnologia Adaptativa em Sistemas de Tomada de Decisão: Uma Abordagem Estratégica na Seleção de Fornecedores". *Segundo Workshop de Tecnologia Adaptativa*. Escola Politécnica, USP, San Pablo, Brasil, 2008.

[Tchemra – 2009] A. H. Tchemra, "Tabela de decisão adaptativa na tomada de decisões multicritério". *Tesis de doctorado*. Escola Politécnica, USP, San

Pablo, Brasil, 2009.

[Tchemra – 2010] A. H. Tchemra, "Adaptatividade na Tomada de Decisão Multicritério". *4° Workshop de Tecnologia Adaptativa*. Escola Politécnica, USP, San Pablo, Brasil, 2010.

[Varghese – 2007] S. M. Varghese, K. P. Jacob, "Process Profiling Using Frequencies of System Calls". *The Second International Conference on Availability, Reliability and Security (ARES'07)*. Viena, Austria, 2007.

[Vaughan-Nichols – 2009] S. J. Vaughan-Nichols. "The 10 Worst Operating Systems of All Time". *PC World Magazine*, 2009

[Veitch – 1996] A. C. Veitch, N. C. Hutchinson, "Kea – a dynamically extensible and configurable operating system kernel", *3rd International Conference on Configurable Distributed Systems*. Vancouver, Canada, 1996.

[Veitich - 1998] A. C. Veitch, "A Dynamically Reconfigurable and Extensible Operating System" Tesis de doctorado, Dept. Of Computer Science, The University of British Columbia, Canada, 1998.

[von Eicken – 1992] T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser, "Active Messages: A mechanism for Integrated Communication and Computation". *19th International Symposim on Computer Architecture.* Gold Coast, Australia, 1992.

ANEXO I – ESTRUCTURA DEL PLANIFICADOR DE SODIUM

A-I. 1. ESTRUCTURA GENERAL

El planificador de procesos de SODIUM se encarga de seleccionar el siguiente proceso a ejecutar y efectuar el cambio de contexto para poder cederle la ejecución[7] del procesador. Forma parte del kernel monolítico de SODIUM, y ejecuta en el anillo 0 (supervisor) de privilegio del modo protegido de la arquitectura x86.

Actualmente sólo cuenta con una implementación mono-procesador, es decir, que sólo permite despachar procesos para el procesador principal, y no cuenta con operaciones atómicas para la modificación de los estados de ejecución. De todas maneras, se encuentra en etapa de análisis la adaptación para funcionamiento multi-procesador.

El algoritmo que deberá utilizar el planificador se configura, de manera general, utilizando una variable de entorno declarada en el archivo *include/kernel/system.h* de la siguiente manera:

```
short int siPlanificador;
```

Los valores que puede tomar esta variable están definidos en el archivo */include/kernel/sched.h* utilizando las siguientes cláusulas *#define:*

```
#define FIFO 0
#define RR 1
#define BTS 2
```

[7] En la literatura, en inglés, esta acción de cederle la ejecución del procesador a un proceso se identifica con la palabra '*dispatch',* y el mecanismo encargado de realizarlo se identifica como '*dispatcher'*

```
#define RRQV 3
#define RRPR 4
#define SJFS 5
```

La selección de implementación de estos algoritmos está basada en la descripción realizada por [Silberschatz – 2012] para planificadores de procesos para mono-procesadores y se corresponde de la siguiente manera:

- FCFS (0) – First-Come, First-Served Scheduling
- RR (1) – Round-Robin Scheduling
- BTS (2) – Best Time of Service Scheduling[8]
- RRQV (3) – Round-Robin de Quantum Variable
- RRPR (4) – Round-Robin con Prioridades
- SJFS (5) – Shortest-Job-First Scheduling

Durante la etapa de inicialización de SODIUM, luego del booteo, se define el uso de la planificación con Round-Robin como algoritmo de planificación inicial por defecto. Para lograr esto, se utiliza la siguiente llamada desde la función de inicialización del kernel *main* del archivo */kernel/main.c*:

```
lFnSysSchedSetScheduler(RR);
```

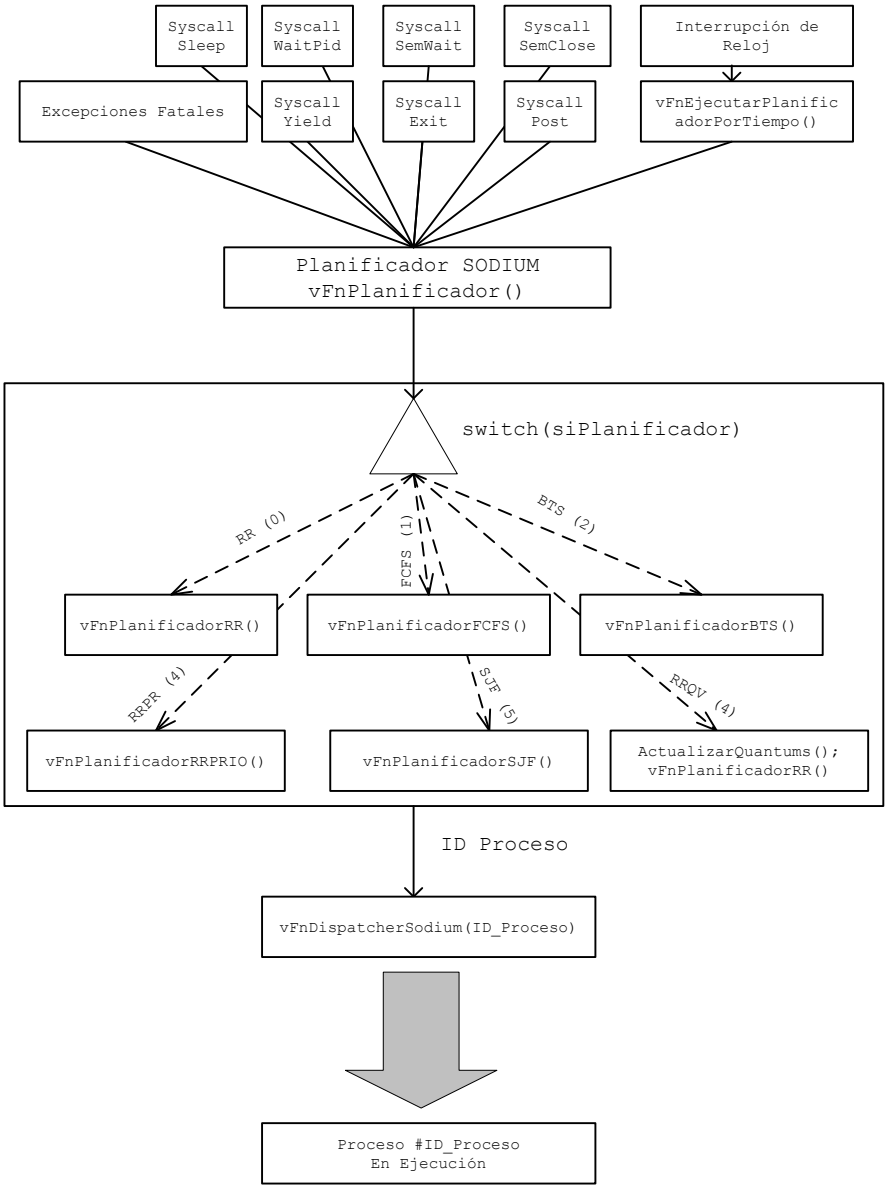
El planificador de procesos de SODIUM es una función de kernel que es llamada desde otros puntos del kernel ante la ocurrencia de distintos eventos. Específicamente, se llama a la función de planificación desde el controlador de la excepción de reloj interno (quantum), si se cumplió con un lapso se quantums mínimo; desde el manejador del syscall

[8] Este algoritmo de planificación no está incluido en la bibliografía citada ya que fue ideado y desarrollado por alumnos de la Universidad Nacional de La Matanza

semclose, sólo si luego de liberarlo es posible reactivar un proceso en espera; desde los manejadores de los syscall semwait y sempost, si el semáforo no está libre; desde el manejador del syscall exit, para terminar el proceso y dejarle la ejecución a otro; desde el manejador del syscall waitpid, para cederle la ejecución a otros procesos mientras se espera la finalización de uno en particular; desde el manejador del syscall yield, para cederle la ejecución a otros proceso, durante un ciclo; desde el manejador del syscall sleep, para cederle la ejecución a otros proceso, mientras se espera un tiempo específico; y desde el manejador de las excepciones fatales, que terminan un proceso y le ceden la ejecución al siguiente.

El llamado al planificador de procesos es llamado siempre desde los puntos indicados, independientemente del algoritmo elegido. La [Figura AI.1] muestra el diagrama de ejecución y despacho de procesos en SODIUM.

Al llamar a la función general *vFnPlanificador()*, es posible abstraer a los demás componentes del sistema del algoritmo de selección de procesos. Luego, es ésta la función encargada de seleccionar el algoritmo indicado por la variable *siPlanificador* mediante un switch. Los algoritmos de planificación tienen el único objetivo de indicar el siguiente proceso a ejecutar y actualizar sus tablas internas. El identificador (ID) de proceso seleccionado es devuelto como un dato de tipo *unsigned int* a *vFnPlanificador()*, que utiliza como parámetro en la llamada a *vFnDispatcherSODIUM(ID_Proceso)*. De ésta manera, se logra abstraer la lógica de selección de procesos del mecanismo del dispatcher propio. Esto permite una mucha mayor facilidad para la programación e intercambio de algoritmos de planificación.



[Figura AI.1] Diagrama de ejecución del planificador de procesos de SODIUM

A-I. 2. FUNCIONES PARA CAMBIOS DE ALGORITMO

El planificador de procesos de SODIUM, en la versión base sobre la cual se realizó el desarrollo de esta investigación, cuenta con una interfaz para el cambio de algoritmo de planificación que es accedida, inicialmente, sólo mediante programas ejecutados desde la

línea de comandos. A continuación se indican los comandos de usuario, las funciones de kernel implementadas para permitir el cambio de algoritmos de planificación.

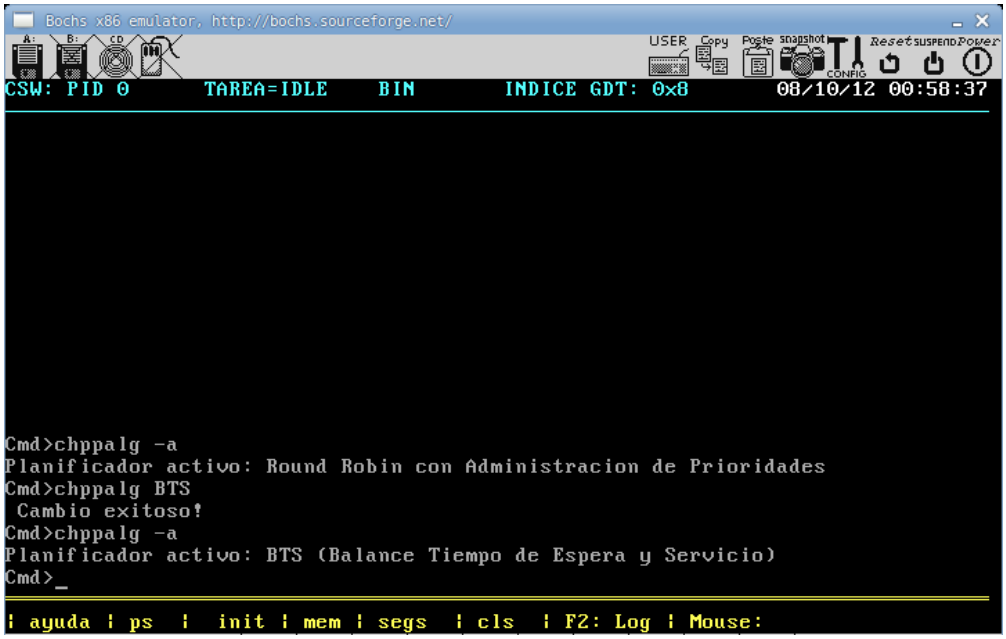
A-I. 3.1. COMANDOS DE USUARIO

Se han desarrollado diversos comandos, compilados como programas de administración de kernel – *chppalg, vcqntm, y chgprio* –, ejecutables desde el entorno de usuario para la administración del planificador de procesos en SODIUM.

A-I. 3.1.1. CHPPALG

El comando *chppalg* permite listar o modificar el algoritmo de planificación actual. El modo de uso es:

`$chppalg -o`	Lista los algoritmos de planificación implementados actualmente en SODIUM
`$chppalg -a`	Indica el algoritmo de planificación de procesos activo en el sistema
`$chppalg <Algoritmo>`	Activa en el sistema el algoritmo de planificación seleccionado. Las opciones disponibles son: - RR: Round Robin - FCFS: First Come First Serve - BTS: Balance Tiempo de Espera y Servicio - RRPRIO: Round Robin con administración de prioridades. - RRQNTM: Round Robin con Quantum Variable. - SJF: Shortest-Job-First
`$chppalg --ayuda`	Muestra la ayuda del comando



[Figura AI.7] Ejemplo de cambio de duración de quantum en SODIUM

En la [Figura AI.7] se muestra una captura de pantalla de un ejemplo de utilización del comando *chppalg* en el que se cambia de un modo de planificación a otro en SODIUM.

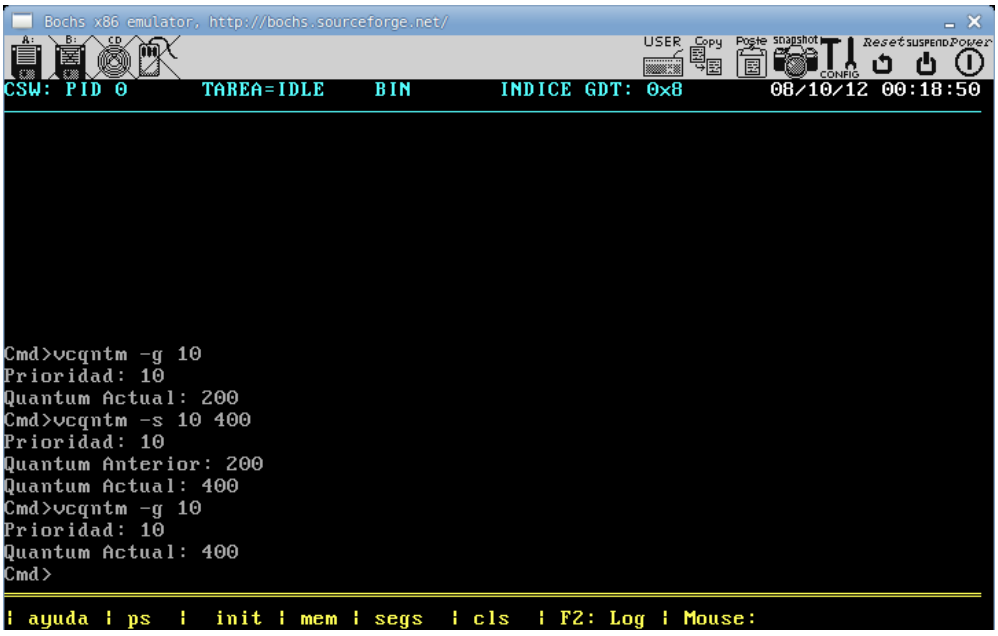
A-I. 3.1.2. VCQNTM

El comando *vcqntm* permite listar o modificar la duración del quantum de un nivel de prioridad en particular a utilizar por el algoritmo de planificación Round-Robin con quantum variable. El modo de uso es:

`$vcqntm`	Devuelve un listado con todos los valores de quantum para cada prioridad de proceso
`$vcqntm -g <valor>`	Devuelve la duración de quantum para la prioridad ingresada como parámetro <*valor*>. El valor de la prioridad debe ser un número entero entre -20 y 19
`$vcqntm <valor> <duración>`	Modifica la duración del quantum para la prioridad ingresada como parámetro <*valor*>, con lo especificado con el parámetro

	<*duración*>. El valor de la prioridad debe ser un número entero entre -20 y 19. El valor del quantum debe ser un número natural mayor a 0.
`$ vcqntm --ayuda`	Muestra la ayuda del comando

En la [Figura AI.8] se muestra una captura de pantalla de un ejemplo de utilización del comando vcqntm en el que se cambia la duración del quantum para el nivel de prioridad 10 en SODIUM.



[Figura AI.8] Ejemplo de cambio de duración de quantum en SODIUM

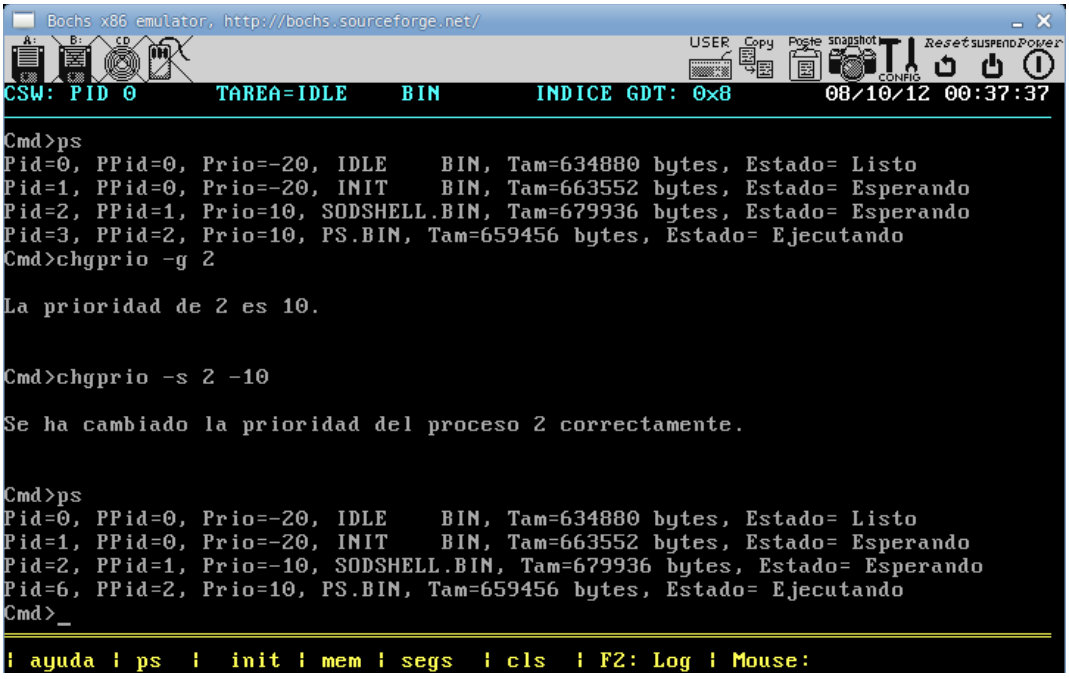
A-I. 3.1.3. CHGPRIO

El comando *chgprio* permite listar o modificar la prioridad asociada a un proceso en SODIUM. El modo de uso es:

`$chgprio -g <PID>`	Devuelve el valor de la prioridad asociada al proceso

	correspondiente al identificador ingresado como el parámetro <*PID*>.
`$chgprio -s <PID> <Prioridad>`	Establece la prioridad ingresada en <*Prioridad*>, para el proceso correspondiente al identificador ingresado como el parámetro <*PID*>.
`$chgprio --ayuda`	Muestra la ayuda del comando

En la [Figura AI.9] se muestra una captura de pantalla de un ejemplo de utilización del comando chgprio en el que se cambia la prioridad del proceso correspondiente a la consola de comandos –*sodshell*– quantum para el nivel de prioridad -10 en SODIUM.



[Figura AI.9] Ejemplo de cambio de prioridad para el shell de SODIUM

A-I. 3.2. FUNCIONES DE KERNEL

Para el cambio y configuración del planificador dentro del kernel de SODIUM, se utilizan las siguientes funciones públicas, las cuales serán utilizadas en esta investigación, que están definidas dentro del archivo */kernel/syscall.h*:

- **int sched_setquantum(int q):** Establece el valor de Quantum en algoritmos de planificación con Quantum variable (Round Robin con Quantum Variable).

- **int sched_getquantum():** Retorna el valor de Quantum actual para os de planificación con Quantums estáticos (RR, BTS y RR con Prioridades).

- **int sched_setvectorquantum (int p, int q):** Establece el valor de Quantum en algoritmos de planificación con Quantum variable

- **int sched_getvectorquantum(int p):** Retorna el valor de quantum para una prioridad determinada, utilizada con algoritmo RR con Quantum Variable.

- **int sched_setscheduler(int p):** Establece el algoritmo de planificación en tiempo de ejecución.

- **int sched_getscheduler():** Retorna el algoritmo de planificación activo en sistema.

- **int sched_yield():** Ejecuta el algoritmo de planificación activo en el sistema.

ANEXO II – ALGORITMOS DE PLANIFICACIÓN DE SODIUM

A-II. 1. DESCRIPCIÓN DE ALGORITMOS DE PLANIFICACIÓN

A continuación se indican detalles de implementación y funcionamiento de los algoritmos de planificación implementados en SODIUM.

A-II.1. 1. ROUND-ROBIN SCHEDULING

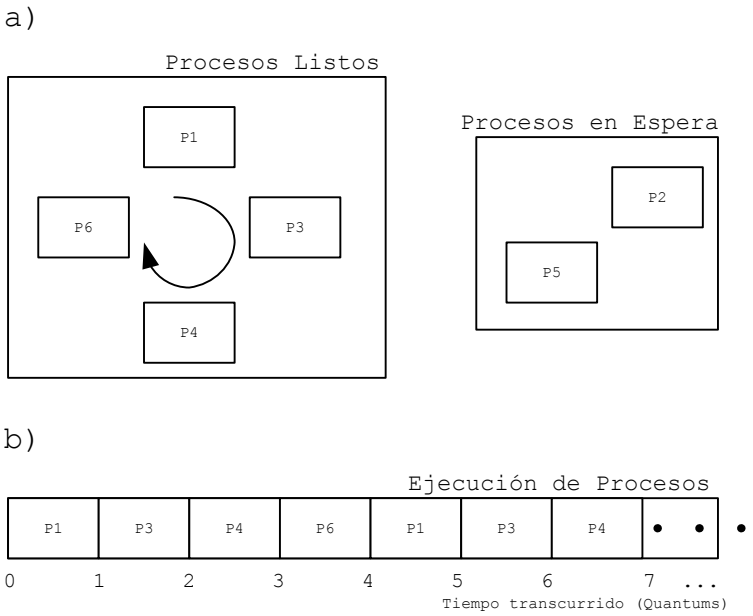
El algoritmo Round-Robin, o distribución en ronda, es un algoritmo expropiativo simple y eficiente para lograr una distribución del uso del procesador entre los diferentes procesos. Se utiliza especialmente para sistemas de tiempo compartido, en el que deben atenderse solicitudes y procesamientos de diferentes usuarios de igual jerarquía.

El algoritmo divide al uso del procesador en pequeñas divisiones de tiempo llamadas *quantums*. La duración de un quantum es definida por el sistema operativo en base a la frecuencia del procesador, y su duración es la de una o más intervalos de interrupciones de reloj. No es posible definir un quantum que dure menos que una interrupción de reloj, ya que el sistema operativo no puede tomar el control de la planificación antes de que el procesador interrumpa la ejecución del proceso. En la versión de Round-Robin implementada en SODIUM, la duración de cada quantum es igual para cada proceso.

La ejecución de un proceso continúa mientras no se haya vencido su quantum de tiempo, por lo que puede continuar ejecutando aún a través de diferentes interrupciones de reloj que no superen el quantum establecido. Una vez que el planificador es llamado y detecta que se ha vencido el quantum de tiempo para el proceso actual, efectúa un cambio de

contexto, resguardando el contexto de ejecución del proceso expropiado en su campo TSS[9] en memoria, y poniendo la del siguiente proceso listo a ejecutar.

En la **[Figura AII.1]** se muestra cómo el algoritmo Round Robin de SODIUM recorre la lista de procesos listos en el orden en que fueron creados – esto lo convierte en una generalización del algoritmo first-come first-serve que se describe más adelante –. El orden de ejecución en procesador es la misma en que la lista fue creada, y es recorrida repetitivamente, dándole a cada proceso un quantum de tiempo equitativo para su ejecución.



[Figura AII.1] a) Recorrido de la lista de procesos listos del algoritmo Round Robin
b) Orden de ejecución de los procesos del algoritmo Round Robin

Para lograr mantener el orden de ejecución asignado por el orden en que han llegado los procesos, en SODIUM se utilizan las siguientes variables para el resguardo del proceso anterior (expropiado), y la asignación del siguiente:

[9] *Task state segment*, o segmento de estado de tarea, es una entrada en la tabla global de descriptores de procesos en la que se le puede indicar al procesador de la familia x86 donde buscar el contexto de ejecución del siguiente proceso a ejecutar, o donde resguardar el que es expropiado.

```
int iIndicePCB;
int iIndicePCBAnterior;
extern unsigned long ulProcActual;
```

Luego de cada ciclo de planificación, la variable *iIndicePCBAnterior* es reemplazada por *iIndicePCB*, que contienen la posición de la entradas de la tabla de PCB de SODIUM, y la variable global *ulProcActual*, se actualiza con el ID de proceso en ejecución, para que pueda ser consultado por otros aspectos del kernel.

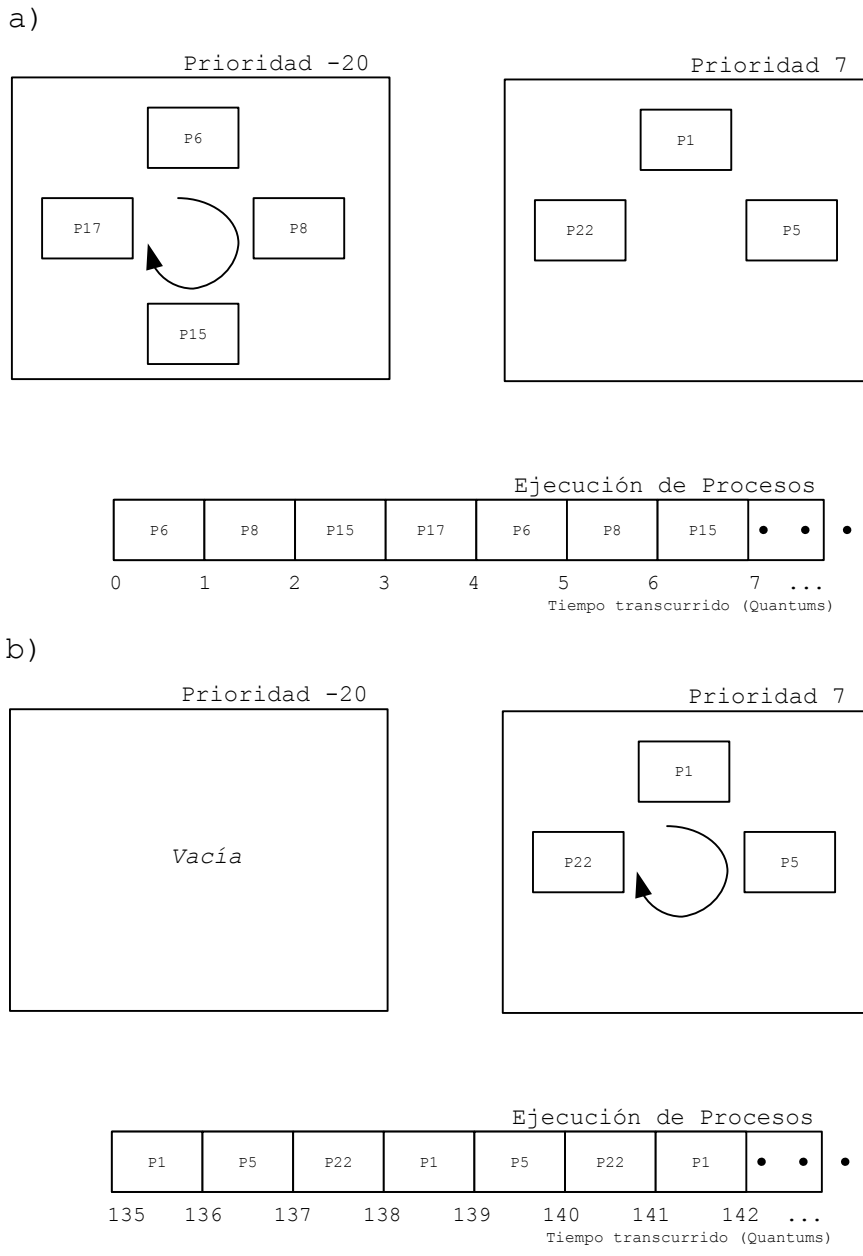
A-II.1. 2. ROUND-ROBIN CON PRIORIDADES

El algoritmo Round-Robin con prioridades, es una variación del algoritmo de Round-Robin original para incluir la posibilidad de definir qué procesos deben tener una frecuencia en el uso del procesador. Esto puede resultar de utilidad para permitir que los procesos propios del sistema operativo, o trabajos de administración y mantenimiento que requieran una ejecución ininterrumpida, puedan hacerlo sin ser interrumpidos por procesos de usuario.

En este algoritmo, la duración de los quantums sigue siendo fija y equitativa para todos los procesos. Sin embargo, la lista –o ronda– de procesos a recorrer no es única, sino que existen múltiples listas, cada una con una prioridad diferente. En SODIUM, se definen un total de 40 listas, que van de un orden de prioridad -20 a +20. En este sentido, procesos listos que se encuentren encolados en una lista de prioridad numérica menor, ejecutarán antes que aquellos que se encuentren en una lista de prioridad numérica mayor.

En la [Figura AII.3] se muestra un ejemplo el comportamiento de este algoritmo en relación a la existencia de procesos encolados en diferentes listas de prioridad. No se recorrerán las listas con menor prioridad (7), hasta que no hayan finalizado todos los procesos encolados en listas de mayor prioridad (-20). Esto podría causar un problema de *inanición*, en la que, si un proceso de mayor prioridad jamás finaliza su ejecución, o se pone en espera, los

procesos de menor prioridad jamás llegan a utilizar el procesador, y permanecerán detenidos indefinidamente.



[Figura AII.3] a) Ejemplo de recorrido de una lista con mayor prioridad primero
b) Luego de que hayan terminado los procesos prioritarios, ejecutan los de menos prioridad

Para solucionar el problema de la inanición, se implementó un mecanismo por el cual, luego de una cantidad de quantums sin ser ejecutado, un proceso aumenta de prioridad en 1

nivel, pasando a la siguiente lista más prioritaria. Al llegar a la lista en la que se encuentra recorriendo actualmente el algoritmo, éste proceso participa de la ronda de ejecución. Sin embargo, una vez que obtiene la ejecución durante su quantum, vuelve a recuperar su prioridad original.

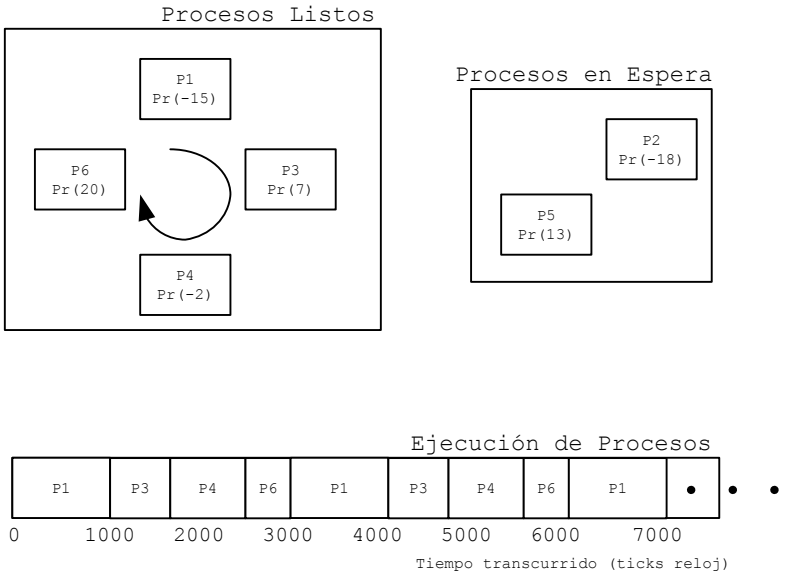
A-II.1. 3. ROUND-ROBIN DE QUANTUM VARIABLE

El algoritmo Round-Robin de Quantum Variable es una alternativa al Round-Robin con prioridades implementado en SODIUM que busca mantener el orden de prioridades de los procesos, pero de manera de evitar el posible exceso de inanición que presentado por el uso de listas de prioridades.

En este algoritmo, la duración de los quantums se puede definir para cada nivel de prioridad en particular. Por lo tanto, un proceso de máxima prioridad contará con un mayor tiempo de ejecución, cuando sea su turno, que uno de menor prioridad. Si las duraciones de cada prioridad no es definida por el usuario, ésta se inicializa, por de parte del kernel, utilizando la siguiente fórmula:

$$Quantum_i = (40 - Prioridad_i) * 20 \text{ ticks}$$

Dada esta fórmula, un proceso con prioridad máxima (-20), por ejemplo, contará con una ejecución de un quantum de 1200 ticks de reloj, donde un proceso de prioridad mínima (20) contará con una ejecución de duración de tan solo 400 ticks de reloj.



[Figura AII.4] Ejemplo de procesos con diferente prioridad. El Proceso 1 cuenta con más del doble de tiempo de ejecución que el Proceso 6 en este ejemplo.

Si bien, utilizando esta primera definición, el algoritmo no presenta inanición, tampoco permite definir procesos de importancia tal que cuya prioridad definan una ejecución casi-exclusiva del procesador. Más aún, dada esta implementación, la razón en porcentaje de uso del procesador entre el más prioritario de los procesos y el menos importante es tan solo de 3 a 1. Para resolver este problema para casos particulares, el algoritmo permite personalizar la duración del quantum particular para cada nivel de prioridad.

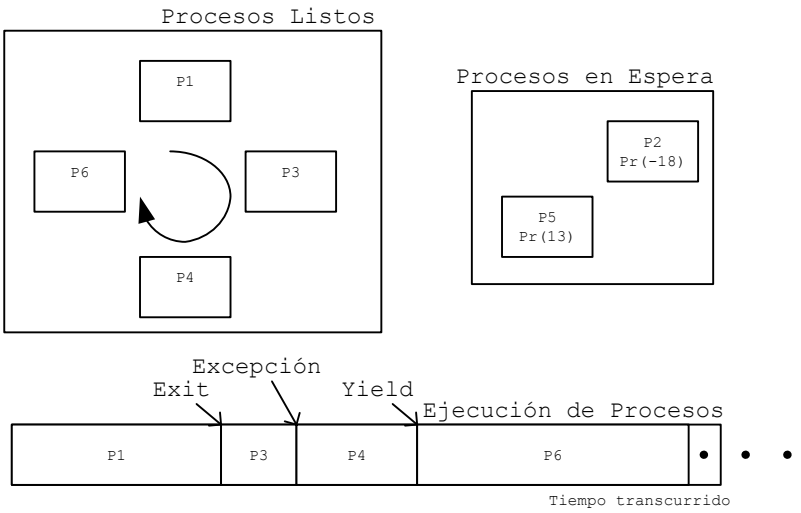
En la [Figura AII.4] se muestra un ejemplo el comportamiento de este algoritmo en relación a la existencia de procesos definidos con diferentes prioridades, utilizando las duraciones de quantum por prioridad iniciales sin modificar.

A-II.1. 4. FIRST-COME, FIRST-SERVED SCHEDULING

El algoritmo First-Come, First-Served –en español, se traduce a "el primero en llegar es el primero en ser atendido"– es el más simple de los algoritmos de planificación. Se trata de un caso particular del algoritmo de Round-Robin original en el que se define un quantum de

duración infinita. De esta manera, el algoritmo jamás expropia a los procesos de su ejecución por cumplirse un lapso de tiempo.

Únicamente cuando vuelve a ejecutarse el planificador por causa de una excepción, o por uno de los syscalls listados, se llama al planificador ordinario de round-robin para que seleccione el siguiente proceso. En este punto, la implementación en SODIUM de este algoritmo varía del presente en la bibliografía, ya que, permite la ejecución de otros procesos antes de la finalización del primer llegado, en caso de que éste pase a quedarse en estado de espera, o ceda su ejecución manualmente. En la [Figura AII.5] se muestra un ejemplo el comportamiento de este algoritmo.



[Figura AII.5] Ejemplo de procesos con el algoritmo de First-Come First-Served

A-II.1. 5. SHORTEST-JOB-FIRST SCHEDULING

El algoritmo de planificación de procesos Shortest-Job-First utilizado para reducir el tiempo de espera para la ejecución de trabajos [10]de largo-plazo. Se plantea establecer el orden de ejecución de los procesos en base, en principio, al tiempo que requerirá su ejecución. Al

[10] Se define *trabajo* como la ejecución completa de un proceso, desde que comienza hasta que termina, independientemente de las interrupciones o la planificación.

posicionar primero a aquellos trabajos que requieren menos tiempo de ejecución, la sumatoria de tiempos de espera de todos los procesos será menor. En oposición, los trabajos más largos serán los que más deberán esperar a la ejecución.

La implementación en SODIUM de este algoritmo es no-expropiativa, es decir, no se establece un límite de tiempo para la ejecución de los procesos, aunque sí permite intercambiar procesos, si el activo cede la ejecución o se pone en espera, en el misma manera que el algoritmo First-Come-First-Served.

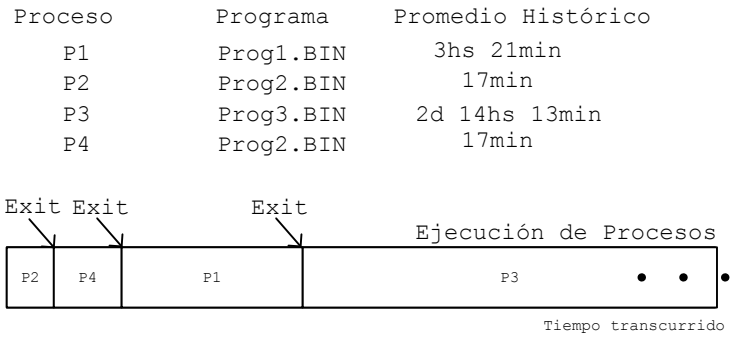
Si bien podría definirse una granularidad más fina para predecir el tamaño de los CPU bursts[11], ajustando en cada cambio la lista de orden de ejecución a un nivel mayor de detalle, este enfoque es impracticable ya que el *overhead*[12] podría superar en costo al beneficio obtenido.

La definición del algoritmo en la bibliografía no indica cómo establecer cuál será el tiempo de ejecución del trabajo. No es posible conocer este dato antes de la ejecución, ya que el tiempo que demandará en ejecutar una tarea no está en función del tamaño del ejecutable, ni en ninguna otra información que pueda obtenerse a priori. La forma implementada en SODIUM para estimar este dato es utilizar datos históricos sobre la ejecución del programa según su nombre de archivo y parámetros de ejecución. En caso de no existir información histórica, se asume el valor promedio entre todos los demás datos históricos como valor inicial. El overhead producido por este enfoque en mínimo, ya que se actualizan las tablas de datos históricos únicamente luego de la finalización de cada proceso.

[11] *CPU Bursts*, se traduce a ráfagas de uso de CPU, marcando el tiempo entre que un proceso comienza a utilizarlo, hasta que lo cede, ya sea por expropiación o syscall.

[12] *Overhead*, tiempo en que el procesador se encuentra ejecutando el kernel, y no procesos de usuario. Se lo considera tiempo perdido y se lo intenta reducir al mínimo posible.

En la [Figura AII.6] se muestra un ejemplo del algoritmo Shortest-Job-First implementado en SODIUM, utilizando datos históricos para definir el posible tiempo de ejecución de un trabajo, asociándolo al programa y los parámetros utilizados. Es importante notar que, si bien el proceso P2, y el proceso P4 ejecutan el mismo programa, cuyo promedio de ejecución es de 17 minutos, pueden tener, durante la nueva ejecución, duraciones diferentes.



[Figura AII.6] Ejemplo de procesos con el algoritmo de Shortest-Job-First en SODIUM.

A-II.1. 6. BEST-TIME-OF-SERVICE SCHEDULING

El algoritmo Best-Time-Of-Service implementado en SODIUM, fue ideado y desarrollado por alumnos de la Universidad Nacional de La Matanza con la finalidad de crear un algoritmo expropiativo que sea capaz de establecer quantums muchos más grandes, reduciendo el overhead por cambios de contexto, pero sin perder el balance de que provee el uso de quantums reducidos.

Al utilizar quantums largos, se penaliza a aquellos procesos que requieren tiempos de espera por acceso a dispositivos o ceden la ejecución con frecuencia. Para compensar esto, el algoritmo consulta la métrica de *tiempo efectivo de servicio* que indica la cantidad de ticks de reloj de ejecución en procesador de cada proceso, y prioriza la ejecución de aquellos relegados respecto a los que pudieron utilizarlo más tiempo.

Otra forma de interpretar el funcionamiento de este algoritmo es que busca compensar el tiempo de quantum perdido al interrumpir su ejecución sin completarlo. El overhead introducido por la comparación entre tiempo de servicio entre los procesos se compensa por el hecho de utilizar quantums de alta duración.

ANEXO III – MÉTRICAS DE PLANIFICACIÓN DE PROCESOS

El planificador de procesos de SODIUM mantiene una serie de datos estadísticos sobre el uso del procesador en general, y también del comportamiento particular de cada proceso. Estos datos se mantienen con fines didácticos, con el motivo de que los alumnos puedan registrarlos y luego consultarlos para comparar los efectos de seleccionar uno u otro algoritmo de planificación.

En el ámbito del sistema operativo, estos datos estadísticos se elaboran a partir de *métricas*, que se utilizan también dentro de los mismos algoritmos de planificación para la toma de decisiones o compensación. Un ejemplo de esto es el algoritmo de Best-Time-Of-Service que mantiene el uso efectivo del procesador de cada proceso para compensar los que menos pudieron utilizarlo. De todas maneras, el acceso a estos datos es público para todo el kernel, y puede ser utilizado, como en el caso de esta investigación, para tomar decisiones de más alto nivel.

Las métricas mantenidas por SODIUM se dividen en generales y particulares de cada proceso. Una descripción de cada una se presenta a continuación:

A-III. 1. MÉTRICAS GENERALES

A-III.1. 1. UTILIZACIÓN DEL PROCESADOR

Una de los criterios para la selección de un buen algoritmo de planificación es lograr que el procesador se encuentre la mayor cantidad de tiempo. La métrica de utilización de procesador

se mantiene para conocer la razón entre el procesamiento efectivo de procesos, por un lado, y el tiempo ocioso o utilizado por el sistema, por el otro.

SODIUM actualiza este dato, tomando lapsos de 5 segundos, utilizando la siguiente fórmula:

$$\%_{cpu} = \frac{\sum_{k}^{n} T_i}{\sum_{0}^{n} T_i}$$

Donde:

- T_i es el tiempo que cada proceso i ha ocupado utilizando el procesador. Se toman en cuenta sólo los procesos de usuario comenzando a partir del índice k. Todos los procesos desde 1 a k, se consideran procesos de sistema.
- T_0 es el tiempo que el planificador despacha al proceso nulo, utilizado como señal para el procesador de que no hay nada que procesar durante el quantum a transcurrir. Es despachado cuando no existe ningún otro proceso listo esperando por ejecutar.

A-III.1. 2. THROUGHPUT (TASA DE FINALIZACIÓN)

Otra forma de evaluar el rendimiento del algoritmo de planificación utilizado es el de *throughput* o tasa de finalización. Este dato se obtiene de calcular la cantidad de trabajos que han finalizado dada una unidad de tiempo relativamente larga.

El problema de utilizar únicamente el porcentaje de utilización del procesador es que, podría estarse ignorando una situación en la cual se produce inanición, o que se pospone demasiado la atención de procesos críticos, o de pequeña duración.

SODIUM actualiza esta métrica cada vez que un proceso finaliza fórmula:

$$\#_f = \frac{\Delta t}{P_f}$$

Donde:

- P_f es la cantidad de procesos completamente ejecutados durante el lapso medido.
- Δt (minutos) es el lapso de tiempo utilizado.

A-III. 2. MÉTRICAS PARTICULARES DEL PROCESO

A-III. 2.1. TURNAROUND TIME (TIEMPO DE CICLO DE VIDA)

La métrica de Turnaround time –o tiempo de ciclo de vida– se utiliza para conocer el tiempo total que le ha tomado al proceso ejecutar un trabajo, desde que es creado hasta que finaliza. Esto incluye los intervalos de tiempo que el proceso espera para ingresar a memoria, esperando en la cola de procesos listos, ejecutando en el procesador, y realizando operaciones de entrada/salida.

SODIUM actualiza el tiempo de ciclo de vida (t_{cv}), para cada proceso, luego de su creación o finalización utilizando la siguiente fórmula:

$$t_{cv} = t_f - t_0$$

Donde:

- t_f es el momento de finalización del proceso.
- t_0 es el momento en que el proceso fue solicitado para su ejecución.

A-III. 2.2. WAITING TIME (TIEMPO DE ESPERA)

La métrica de Waiting time –o tiempo de espera– se utiliza para conocer el tiempo total transcurrido en el que el proceso se ha encontrado esperando en la cola de listos esperando a ser ejecutado. Ya que el algoritmo de planificación no influye en el tiempo que un proceso espera operaciones de entrada/salida, éste tiempo no es relevante para su evaluación, y no se incluye como parte del tiempo de espera.

SODIUM actualiza esta métrica, para cada proceso, luego de que es seleccionado para ejecutar, previamente estableciendo el momento en que fue puesto en la cola de listos. Cada nuevo intervalo de espera se suma para obtener el tiempo total de espera (t_w):

$$t_w = \sum t_e - t_r$$

Donde:

- t_e es el momento en que el proceso entra a la cola de listos.
- t_0 es el momento en que el proceso es despachado para ejecución.

A-III. 2.3. RESPONSE TIME (TIEMPO DE RESPUESTA)

En un sistema interactivo, la métrica de tiempo de ciclo de vida puede no resultar útil, ya que se considera que una rápida respuesta por parte de los procesos es más importante que el tiempo total de procesamiento de estos. A menudo, un proceso puede producir una salida relativamente rápido, y puede continuar produciendo nuevos resultados mientras los producidos se van entregando al usuario.

La métrica de Response Time –o tiempo de respuesta– se utiliza para conocer el tiempo promedio transcurrido en el que se realiza una solicitud, ya sea por dispositivos de entrada, o por el uso de sincronización entre procesos o señales, y ésta es atendida por el

proceso. Para esto, se tiene en cuenta el momento en que el proceso comienza a responder, y no necesariamente el momento en que el resultado está completamente entregado al usuario.

SODIUM actualiza el tiempo promedio de respuesta ($\overline{t_r}$), para cada proceso, luego de que produce la primera impresión en pantalla, escritura sobre un dispositivo, ejecuta un syscall de creación de proceso, o emite una señal, previamente estableciendo el momento en que éste es liberado de la cola de espera para ingresos de teclado, mouse, o es interrumpido por una señal proveniente de otro proceso:

$$\overline{t_r} = \frac{1}{n}\sum_{i=1}^{n} t_{o_i} - t_{s_i}$$

Donde:

- t_o es el momento en que el proceso responde luego de la solicitud.
- t_s es el momento en que el proceso recibe la solicitud.
- n es la cantidad de veces que se recibieron solicitudes. Para n = 0, SODIUM asume que el tiempo de respuesta es inmediato.

A-III. 2.4. TIEMPO EFECTIVO DE SERVICIO

La métrica de tiempo efectivo de servicio se utiliza para conocer el tiempo total de ejecución en procesador que ha pasado un proceso, desde que ha sido creado, sin tener en cuenta las interrupciones que pudo haber sufrido. Esta métrica es utilizada en SODIUM por el algoritmo Best-Time-Of-Service para determinar qué procesos han pasado menos tiempo en uso del procesador.

SODIUM actualiza el tiempo efectivo de servicio total (t_s), para cada proceso, tomando el momento en que el proceso es despachado a ejecución en el procesador, y el momento en que es expropiado de la ejecución, o realiza una solicitud de entrada/salida.

$$t_s = \sum_{i=1}^{n} t_{e_i} - t_{i_i}$$

Donde:

- t_e es el momento en que el proceso es expropiado de la ejecución.
- t_i es el momento en que el proceso inicia la ejecución en el procesador
- n es la cantidad de veces que se despachó el proceso.

A-III. 2.5. OVERHEAD

La métrica de overhead se utiliza para conocer el tiempo de procesador utilizado por el kernel, en el que no se está ejecutando ninguna instrucción o acción solicitada por un proceso.

SODIUM mide el overhead de manera cuantitativa, llevando la cuenta de cuantos cambios de contexto se realizan. Se toma este dato ya que medir en tiempo la cantidad de ciclos de procesador toma cada acción de cambio de contexto influye en gran medida en su resultado.

A mayor cantidad de procesos de sistema, el valor del overhead aumentará sin mejorar el resultado para las demás métricas, por lo que utilizar una medida cuantificada permite fácilmente evaluar su impacto sobre el resto de los procesos.