

Sistemas operativos: Gestión de procesos

TEMA 16

ABACUS NT

Oposiciones 2021

Índice

- 1. Introducción**
- 2. Procesos**
 - 2.1. Definición de proceso
 - 2.2. Tabla de Procesos
 - 2.3. Jerarquía de procesos
 - 2.4. Estados del proceso
 - 2.5. Concurrencia y cambio de contexto
 - 2.5.1. Concurrencia o multiprogramación
 - 2.6. Hilos
- 3. Interacción entre procesos**
 - 3.1. Competencia entre Procesos
 - 3.1.1. Condiciones de competencia
 - 3.1.2. La sección crítica
 - 3.1.3. Métodos de exclusión mutua
 - 3.2. Comunicación entre procesos
 - 3.2.1. Memoria Compartida
 - 3.2.2. Intercambio de mensajes
 - 3.2.3. Buzones y puertos
 - 3.2.4. Conductos (pipes)
 - 3.2.5. Llamadas a procedimientos remotos (RPC)
 - 3.3. Interbloqueo e Inanición
 - 3.3.1. Condiciones de Hoffman
 - 3.3.2. Prevención del interbloqueo
- 4. Planificación de Procesos**
 - 4.1. Objetivo de la planificación de procesos
 - 4.1.1. Planificación garantizada
 - 4.2. Algoritmos de Planificación
 - 4.2.1. Primero en llegar, primero en servir (FCFS)
 - 4.2.2. Round Robin (RR)
 - 4.2.3. Planificación con prioridad
 - 4.2.4. Primero el más corto (SJF)
 - 4.2.5. Menor tiempo restante (SRT)
 - 4.2.6. Colas con diferente nivel de prioridad (colas multinivel)
 - 4.2.7. Colas de diferente nivel de prioridad realimentadas
 - 4.3. Algoritmos de planificación en multiprocesadores
 - 4.4. Implementación de la planificación de procesos en los sistemas operativos actuales
 - 4.5. Gestión de procesos en Android
 - 4.5.1. Tipos de procesos en Android.
 - 4.5.2. Planificación de procesos
- 5. Conclusión.**
 - 5.1. Relación del tema con el currículo
- 6. Bibliografía**

1. Introducción

Un ordenador sin sistema operativo es un cacharro inútil. Más de un neófito con su recién adquirida máquina ha vuelto a la tienda a reclamar que “no hace nada” por no haber sido aún instalado.

Y es que la interacción máquina-usuario se produce precisamente mediante la interfaz del sistema. El sistema operativo controla todos los recursos de la computadora y proporciona la base sobre la cual pueden escribirse los programas de aplicación.

Una de las principales funciones del sistema operativo es ocultar toda esta complejidad y proporcionar al programador y al usuario un conjunto más conveniente de instrucciones y programas con el cual trabajar.

Por encima del sistema operativo está el resto del software de sistema. Aquí se encuentran el intérprete de comandos (shell), compiladores, editores y programas de aplicación independientes similares a ellos.

La principal función de los sistemas operativos (SSOO) según el autor **William Stallings**, en su libro Sistemas operativos (2005), es **“controlar la ejecución de programas y aplicaciones y actuar como interfaz entre las aplicaciones y el hardware del computador.”**

En este tema vamos a ver cómo lleva a cabo la gestión de procesos el sistema operativo, cómo los planifica, qué dificultades se presentan en esta tarea y cómo se puede solucionar. Por último veremos como un sistema operativo actual, tal como **Android**, lleva a cabo la gestión de procesos.

2. Procesos

2.1. Definición de proceso

El proceso se puede definir como un programa en ejecución y, de una forma más precisa, como la unidad de procesamiento gestionada por el sistema operativo. Todos los programas cuya ejecución solicitan los usuarios lo hacen en forma de procesos, de ahí la importancia de conocerlos en detalle.

La idea clave es que un proceso es una entidad dinámica.

En este tema se verán las características de los procesos y las acciones que debe llevar a cabo el Sistema Operativo para realizar su gestión.

2.2. Tabla de Procesos

El sistema operativo mantiene una tabla de procesos, dentro de la que se almacena un BCP (Bloque de Control de Proceso) por cada proceso. Por razones de eficiencia, la tabla se

construye generalmente como una estructura estática, con un determinado número de BCP, todos ellos del mismo tamaño.

El BCP contiene la información básica del proceso, en la que cabe destacar:

- Información de Identificación: Identificador del proceso, del proceso padre, identificador del usuario y del grupo.
- Estado del procesador: Contiene los valores en los que fue interrumpido el proceso, o los valores iniciales si aún no ha entrado en ejecución
- Información de Control del Proceso: En esta sección se incluye diversa información entre la que cabe destacar:
 - Información de planificación y estado:
 - Estado del proceso
 - Evento que espera el proceso bloqueado
 - Prioridad
 - Otra información de planificación
 - Descripción de los segmentos/páginas de memoria asignadas
 - Recursos asignados (Archivos, puertos de comunicación, etc)
- Punteros: para encolar procesos
- Comunicación entre procesos: señales, mensajes, etc

2.3. Jerarquía de procesos

Los procesos no se encuentran aislados, sino que se pueden organizar en diferentes estructuras, normalmente jerárquica, así por ejemplo en GNU/Linux, los procesos pueden crear otros procesos hijos usando la llamada al sistema **FORK**, obteniéndose de esta manera un árbol de procesos. A su vez, los procesos tendrán uno o varios hilos, los cuales se pueden definir como la unidad básica de ejecución. Todo esto, debe ser teniendo en cuenta por los SSOO para obtener una gestión eficiente de los procesos.

2.4. Estados del proceso

Un proceso puede encontrarse en tres estados diferentes:

- **En Ejecución:** En este estado se encuentra el proceso que está siendo ejecutado por el procesador. El estado del proceso residirá en los registros del procesador.
- **Bloqueado:** Un proceso bloqueado está esperando a que ocurra un evento y no puede seguir ejecutándose hasta que termine el evento. Una situación típica de proceso bloqueado se produce cuando el proceso solicita una operación de entrada/salida. Hasta que no termina esta operación, el proceso queda bloqueado.
- **Listo:** Un proceso está listo para ejecutar cuando puede entrar en fase de procesamiento. Dado que puede haber varios procesos en este estado, una de las tareas del sistema operativo será seleccionar aquél que debe pasar a ejecución. El módulo del sistema operativo que toma esta decisión se denomina planificador (scheduler).

Las posibles transiciones entre estados son las siguientes:



- **Transición 1**: De ejecutándose a bloqueado: el programa que está en ejecución necesita algún recurso, señal, datos, etc para poder seguir ejecutándose.
- **Transición 2**: De ejecutándose a listo: El proceso ha utilizado el tiempo asignado de CPU y tiene que dejar paso al siguiente proceso
- **Transición 3**: De listo a ejecutándose: Al proceso le llega una nueva disposición de tiempo de la CPU
- **Transición 4**: De bloqueado a Listo: El recurso que necesitaba el proceso ha sido liberado y está disponible, por lo que sólo necesita pasar a ejecución.

2.5. Concurrencia y cambio de contexto

Un cambio de contexto consiste en la ejecución de una rutina perteneciente al núcleo del sistema operativo multitarea de una computadora, cuyo propósito es parar la ejecución de un hilo o proceso para dar paso a la ejecución de otro distinto.

2.5.1. Concurrencia o multiprogramación

En principio, una computadora que dispone de un único microprocesador solamente puede ejecutar un proceso al mismo tiempo. No es posible ejecutar otro proceso hasta que ha finalizado el anterior.

No obstante, es posible alternar el uso de la CPU entre varios procesos durante cortos períodos de tiempo. Esto se denomina ejecución concurrente o multiprogramación.

Además, durante la ejecución de un proceso existen muchos tiempos muertos donde no es necesario el uso de la CPU. Se trata de los momentos en los que el programa está esperando a que finalice una operación de entrada/salida, por ejemplo, una lectura desde el disco duro. Estos tiempos muertos podrían aprovecharse para ejecutar otro programa.

Gracias a las interrupciones generadas por el propio ordenador, es posible expulsar a un programa en ejecución para dar paso al sistema operativo. Cuando esto ocurre, el sistema operativo ejecuta inmediatamente la rutina de cambio de contexto. Esta rutina realiza las siguientes operaciones en el orden indicado:

1. Salvar el estado del programa que se estaba ejecutando. El estado, también denominado contexto, consiste en los valores de todos los registros del microprocesador. Se copian en la memoria principal.

2. Seleccionar otro programa para ejecutar. Entre todos los programas que estén preparados para ejecutarse, la rutina selecciona uno de ellos siguiendo algún algoritmo equitativo.
3. Restaurar el estado del programa seleccionado. Para ello, se toma el estado previamente copiado en la memoria principal y se vuelca en los registros del microprocesador.
4. Ejecutar el programa seleccionado. La rutina termina su ejecución saltando a la instrucción que estaba pendiente de ejecutar en el programa seleccionado.

La ejecución concurrente también es aplicable a computadoras Multiprocesador. En este caso, se llevan a cabo cambios de contexto en cada microprocesador de manera independiente.

2.6. Hilos

Los hilos son en realidad subprocessos. Cada hilo se ejecuta secuencialmente y tiene su propio contador de programa y una pila para llevar un registro de posición, pero comparte el resto de recursos asignados al proceso (archivos abiertos, procesos hijos, cronómetros, etc) con los otros hilos del proceso.

Una ventaja del uso de hilos es que la comunicación entre los hilos es muy rápida. Además, es más rápido crear otro hilo que otro proceso.

Una tarea posee recursos, pero no posee capacidad de ejecución, debe tener al menos un hilo de ejecución. Un proceso clásico sería equivalente a una tarea con un hilo en ejecución.

Empleando hilos podemos tener varias unidades de ejecución activas en una misma aplicación. Por ejemplo, en un procesador de textos, un hilo recoge las pulsaciones del teclado y visualiza el texto y otro hilo realiza las tareas de corrector ortográfico. Otro ejemplo son los sistemas cliente-servidor, en los que el servidor genera un hilo para atender a cada proceso cliente, esto sucede en chats o servidores de páginas Web.

Como ventajas, los cambios de contexto entre hilos de la misma tarea son mucho más rápidos que entre procesos, además, en sistemas multiprocesador los distintos hilos de una aplicación pueden ser ejecutados en paralelo en distintos procesadores.

3. Interacción entre procesos

Dos o más procesos que se ejecuten simultáneamente se dicen concurrentes.

Dichos procesos pueden ser independientes o cooperantes.

- Un proceso independiente es aquél que se ejecuta sin requerir ayuda o actividad de otros procesos. Esto no significa que su ejecución no dependa en absoluto de los demás, ya que pueden darse condiciones de competencia por los distintos recursos disponibles.

- Un proceso Cooperante puede requerir de mecanismos de sincronización y comunicación entre sí para conseguir algún fin; los procesos pueden formar parte de una misma jerarquía de procesos: Un proceso padre puede crear subprocessos hijos y un mismo subprocesso puede tener varios hilos de ejecución que comparten parte de la memoria asignada.

Tanto si los procesos son independientes o cooperantes, pueden producirse una serie de interacciones entre ellos. Estas interacciones pueden ser de dos tipos:

- Interacciones motivadas porque los procesos comparten o compiten por el acceso a recursos físicos o lógicos.
- Interacciones motivadas porque los procesos se comunican y sincronizan entre sí para alcanzar un objetivo común.

Estos dos tipos de interacciones obligan al sistema operativo a incluir mecanismos y servicios que permitan la comunicación y la sincronización entre procesos.

3.1. Competencia entre Procesos

3.1.1. Condiciones de competencia

En los sistemas multitarea, se van a presentar situaciones en las que varios procesos intentan acceder a un mismo recurso. A esta situación se le denomina condición de competencia, y para su gestión va a requerir de la existencia de mecanismos de sincronización entre procesos.

Un problema típico de competencia se produce cuando dos procesos manipulan simultáneamente una misma variable o registro, obteniendo valores totalmente inesperados que van a depender de los instantes de actuación de cada uno.

3.1.2. La sección crítica

Para evitar las condiciones de competencia se necesita la **exclusión mutua** que es una forma de garantizar que, mientras un proceso esté usando un recurso compartido, los otros procesos no podrán hacerlo.

La selección de las operaciones primitivas adecuadas para garantizar la exclusión mútua es una decisión primordial en el diseño del sistema operativo.

La parte del programa que accede al recurso compartido se denomina **sección crítica**.

Para resolver el problema de la sección crítica es necesario un mecanismo de sincronización que permita a los procesos cooperar entre ellos sin problemas. Este mecanismo debe proteger el código de la sección crítica.

Cualquier solución que se utilice para resolver este problema debe cumplir los requisitos siguientes:

- **Exclusión mutua:** Si un proceso está ejecutando código de la sección crítica, ningún otro proceso lo podrá hacer.

- **Progreso:** La decisión sobre qué proceso debe entrar en la sección crítica debe efectuarse en tiempo finito y solamente sobre los procesos que deseen entrar.
- **Espera acotada:** Debe existir un límite en el tiempo que un proceso espera la ejecución de su sección crítica tras su solicitud y mientras otros procesos entran en ella.

3.1.3. Métodos de exclusión mutua

Las soluciones para garantizar la exclusión mutua consisten en que dos procesos no pueden ejecutar su sección crítica simultáneamente dando lugar a condiciones de competencia.

Los mecanismos que permiten esto son:

A. Prohibición de las interrupciones

Dado que incluso la alternancia entre procesos es controlada por interrupciones, un posible método para llevar a cabo la exclusión mutua podría ser que el proceso que entra en una región crítica las desactive todas. Este método no es admisible ya que no es correcto que un proceso desactive todas las interrupciones ya que tendría el control completo del sistema y además si el sistema tuviera más de una CPU la desactivación correspondería sólo a una de ellas.

B. Cerrojos

Podría pensarse que consultar el valor de una variable cerrojo antes de acceder a una sección crítica y modificar su valor para indicar que está ocupada solucionaría el problema. Pero no es así ya que el propio cerrojo presentaría condiciones de competencia.

C. Algoritmo de Peterson

Antes de acceder a su región crítica los procesos tienen que llamar al procedimiento entrarEnSeccionCritica pasando como parámetro su número de proceso. Si el otro proceso está en su sección crítica esperará hasta que la abandone para entrar, en caso contrario entrará inmediatamente. Al final de su sección crítica, cada proceso debe llamar a salirRegionCritica.

Esta solución funciona, pero presenta el problema de realizar espera activa.

D. Dormir y Despertar

Intenta solucionar el problema de la espera activa ampliando el número de estados de un proceso. Para ello se definen dos primitivas, dormir (sleep) y despertar (wakeUp). La primera es una llamada al sistema que provoca la suspensión de quien efectúa la llamada. La segunda requiere de un parámetro que es el identificador del proceso que se desea despertar.

Este sistema puede dar lugar a condiciones de competencia. El problema radica en la pérdida de una llamada a la primitiva wakeUp() cuando el proceso destinatario ha hecho la llamada sleep() pero aún no está dormido, por lo que no la recibe.

E. Semáforos

Un semáforo es una variable especial (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos compartidos. Fueron inventados por Edsger Dijkstra en 1965 y se usaron por primera vez en el sistema operativo THEOS.

Los semáforos sólo pueden ser manipulados por las primitivas P y V que tienen su origen en el idioma holandés. "Verhogen" significa incrementar y "Proberen" probar, aunque Dijkstra usó la palabra inventada prolaag [1], que es una combinación de probeer te verlagen (intentar decrementar).

El valor del semáforo es el número de unidades del recurso que están disponibles (si sólo hay un recurso, se utiliza un "semáforo binario" cuyo valor inicial es 1).

Los semáforos sólo pueden ser manipulados usando las siguientes operaciones (éste es el código con espera activa):

```
Inicia (Semáforo s, Entero v{    s = v; }

P (Semáforo s){    if (s>0)          s = s-1;      else        wait(); }

V (Semáforo s){      if (!procesos_bloqueados)      s =
s+1;      else          signal(); }
```

F. Monitores

Un monitor es una construcción de un lenguaje de programación que contiene tanto los datos como los procedimientos necesarios para asignar un recurso compartido.

Para ejecutar una función de asignación de recursos, un proceso debe llamar a una entrada del monitor específica, pero no tiene acceso a sus estructuras internas. Los procesos que desean usar el monitor cuando ya hay un proceso activo deben esperar, el monitor controla automáticamente esta espera. Como la exclusión mutua está garantizada, se evitan los problemas de competencia.

3.2. Comunicación entre procesos

El problema más general que debe resolverse en el tratamiento de los procesos concurrentes, es el de la comunicación, que consiste en proporcionarles mecanismos que les permitan intercambiar información.

3.2.1. Memoria Compartida

Los procesos comparten algunas variables para el intercambio de información. La responsabilidad de proporcionar la comunicación recae sobre los programadores de aplicaciones, el sistema operativo únicamente tiene que ofrecer la memoria compartida.

3.2.2. Intercambio de mensajes

Se utilizan dos primitivas:

enviar (destinatario, mensaje);
recibir(origen, mensaje);

Se trata de llamadas al sistema y no comandos del lenguaje por lo que son accesibles desde muchos entornos de lenguajes de programación.

Un envío con bloqueo debe esperar hasta que el receptor reciba el mensaje constituyendo un ejemplo de comunicación síncrona. Un envío sin bloqueo permite al transmisor continuar con otras tareas aunque el receptor no haya recibido aún el mensaje permitiendo pues que ambos interlocutores se comuniquen asíncronamente. El envío sin bloqueo requiere de buffers para almacenar los mensajes hasta que los reciba el receptor. La comunicación asíncrona aumenta la posibilidad de paralelismo.

En la transmisión de mensajes pueden darse errores en la comunicación y es necesario un protocolo que permita solucionar este problema. También es necesario poder identificar mediante un nombre de forma no ambigua a los procesos destinos y origen. Estos y otros aspectos relativos a la comunicación son tratados en otros temas del temario por lo que no nos extendemos más sobre ellos.

3.2.3. Buzones y puertos

Un buzón consiste en un buffer en el que se almacenan los mensajes enviados antes de ser extraídos por el receptor. En este caso, el transmisor y el receptor especifican el buzón, en lugar del proceso, con el que se van a comunicar.

Un puerto se define como un buzón utilizado por múltiples transmisores y un único receptor. En los sistemas cliente/servidor, cada servidor tiene un puerto asignado por el cual recibe los mensajes de multitud de clientes.

3.2.4. Conductos (pipes)

UNIX introdujo la noción de conductos como un esquema para manejar comunicaciones entre procesos. Un conducto es en esencia un buzón que permite extraer un número específico de bytes a la vez. El conducto no mantiene las fronteras entre mensajes. Si los procesos convienen en leer y escribir mensajes con un tamaño determinado o terminar cada mensaje con un señalador especial la utilización de los conductos no presenta ningún problema.

3.2.5. Llamadas a procedimientos remotos (RPC)

Son un mecanismo estructurado de alto nivel para realizar la comunicación entre procesos en sistemas distribuidos. Con una llamada a un procedimiento remoto, un proceso de un sistema A llama a un procedimiento de un proceso de otro sistema B. El proceso A se bloquea esperando el retorno desde el procedimiento llamado en el sistema remoto y después continúa su ejecución desde el punto que sigue inmediatamente a la llamada.

El procedimiento llamado y el que llama residen en procesos distintos, con espacios de direcciones distintos, por lo cual no existe la noción de variables globales compartidas, por

lo que las RPC transfieren la información estrictamente a través de los parámetros de la llamada.

Un problema de las RPC es que las llamadas por referencia son difíciles de implementar ya que una RPC comunica espacios de direcciones diferentes.

3.3. Interbloqueo e Inanición

El interbloqueo consiste en varios procesos que compiten por recursos en una situación en la que no pueden proseguir debido a que cada uno necesita recursos que tiene otro proceso. Se da inanición cuando un proceso no recibe nunca los recursos necesarios para su ejecución.

3.3.1. Condiciones de Hoffman

Los autores Silberchatz, Jensen y Tanenbaum, coinciden en que para que se produzca un estado de Intebloqueo las cuatro condiciones de Coffman son condiciones necesarias y suficientes y deben producirse simultáneamente. Stallines, mientras tanto indica que las tres primeras condiciones son necesarias, pero no suficientes, y que la cuarta condición ocurre como consecuencia de las otras tres.

1. Mutua exclusión: sólo un proceso a la vez puede usar el recurso
2. Retener y esperar: el proceso retiene los recursos que ya tiene asignados mientras espera por nuevos a adquirir del conjunto de recursos del sistema.
3. No expropiación: el proceso está reteniendo los recursos concedidos y solo puede liberarlos y devolverlos al sistema como resultado de una acción voluntaria de ese proceso. El S.O. no puede obligar a devolverlos.
4. Espera circular: los procesos están esperando mutuamente a que el otro libere el recurso requerido, formando una cadena circular entre dos o más procesos, en la que cada uno de ellos está esperando un recurso que tiene el próximo miembro de la cadena.

3.3.2. Prevención del interbloqueo

Los métodos para tratar los interbloqueos se pueden dividir en dos, aquellos que intentan que no se produzcan y aquellos que actúan una vez que existe el interbloqueo.

a) Evitación del interbloqueo:

Prevención: Lograr que al menos una de las cuatro condiciones no se verifique.

Evitación: pedir información adicional sobre el resto de recursos que va a necesitar el proceso.

b) Solución una vez producido:

Detección: Se ejecuta periódicamente algún mecanismo de detección de interbloqueo.

Recuperación: Se eliminan los procesos implicados en el interbloqueo, bien todos o bien uno a uno hasta que se recupere el sistema.

4. Planificación de Procesos

4.1. Objetivo de la planificación de procesos

El objetivo de la planificación de procesos es el reparto del tiempo de procesador entre los procesos que están en estado de listo. Para ello el sistema operativo dispone de dos módulos diferenciados:

- El **planificador** (scheduler) realiza la función de seleccionar el proceso que entra en estado de ejecución.
- El **despachador** (dispatcher) es el módulo que pone en ejecución el proceso planificado.

Como hemos dicho, el objetivo de la planificación es optimizar el comportamiento del sistema, pero este puede estar más o menos centrado en cada uno de los siguientes objetivos:

- Reparto equitativo del procesador.
- Eficiencia (optimizar el uso del procesador)
- Menor tiempo de respuesta en uso interactivo
- Menor tiempo de espera en ejecución por lotes (batch)
- Mayor número de trabajos por unidad de tiempo
- Cumplir plazos de ejecución en un sistema de tiempo real

La mayoría de estos objetivos son incompatibles entre sí, por lo que hay que centrar la atención en aquél que sea de mayor interés.

4.1.1. Planificación garantizada

Consiste en llegar a un compromiso con el usuario sobre el rendimiento del sistema que se va a poner a su disposición. Por ejemplo, si hay n usuarios conectados dar $1/n$ del tiempo del procesador a cada usuario.

4.2. Algoritmos de Planificación

Cuando un proceso en Ejecución abandona tal estado pasa a Espera o Preparado, dependiendo si deja el procesador voluntaria o involuntariamente. Si los procesos de un sistema nunca dejan la CPU de forma involuntaria, se dice que la política de planificación de CPU es no expulsora o no expropiativa. Por el contrario, si pueden perder la posesión del procesador sin solicitarlo, nos encontramos con una planificación expulsora o expropiativa.

4.2.1. Primero en llegar, primero en servir (FCFS)

Es de tipo no expulsivo. Los procesos se almacenan en una cola FIFO y son atendidos por orden de llegada. El primer proceso que accede a la CPU se mantiene en ella hasta que se bloquea (por espera de E/S u otro motivo) o termina. Presenta el problema de que los procesos largos hacen esperar a los cortos con lo cual la capacidad de concurrencia del sistema disminuye.

4.2.2. Round Robin (RR)

Los procesos entran en la CPU siguiendo un esquema FIFO, pero con una cantidad de tiempo de CPU limitada denominada quantum. Si el proceso activo sigue en ejecución al final de su quantum es extraído de la CPU por el planificador y el siguiente proceso se apropia del procesador. También se asigna la CPU al siguiente proceso si el proceso activo termina o se bloquea antes del fin del quantum.

Es fundamental el tamaño del quantum. Si es muy pequeño se dedica demasiada CPU al cambio de contexto y si es demasiado grande degenera en FCFS.

4.2.3. Planificación con prioridad

La prioridad es la precedencia de un proceso respecto a los otros procesos que puedan ocupar la CPU. Cada proceso tendrá un nivel de prioridad asignada y el proceso listo con máxima prioridad es el que consigue la CPU.

Las prioridades pueden ser asignadas dinámicamente por el sistema y modificadas a lo largo de la vida del sistema.

4.2.4. Primero el más corto (SJF)

Es un sistema no expulsivo. Concede la CPU de forma ininterrumpida al proceso que necesita la CPU durante menos tiempo. Si dos procesos tienen la misma duración se elige uno mediante FCFS.

Se comporta bien en la planificación a largo plazo de los procesos batch, en los que se conocen los requisitos de tiempo total de CPU. El problema es conocer la estimación de tiempo del resto de procesos (se puede hacer una estimación estadística basada en ejecuciones anteriores).

Una variante denominada prioridad con tasa de respuesta alta, highest response-ratio next (HRN) considera además del tiempo de ejecución el tiempo que el proceso lleva esperando para ser atendido para no hacer esperar indefinidamente a los procesos largos.

4.2.5. Menor tiempo restante (SRT)

Es la versión expulsiva del método SJT. Como en general no es posible conocer anticipadamente el tiempo de ejecución de un proceso se acude a estimaciones del tiempo restante de ejecución, en función de las ejecuciones anteriores del proceso, asignándole la prioridad de manera inversamente proporcional a este valor.

4.2.6. Colas con diferente nivel de prioridad (colas multinivel)

Un esquema usual es agrupar procesos en clases de prioridad y utilizar la planificación por prioridad entre las clases y otro modo de planificación (round robin, SJT, SRT) entre los procesos listos de cada clase.

Por ejemplo, podríamos tener un sistema con cuatro colas de prioridad. Mientras hubiera procesos ejecutables en la cola de mayor prioridad, se ejecutaría cada uno un quantum siguiendo un sistema round robin, y no se ejecutarían de las otras colas. Las otras colas podrían usar el algoritmo RR u otro.

4.2.7. Colas de diferente nivel de prioridad realimentadas

Es una variación de la anterior en la que los procesos no tienen por qué mantenerse en la misma cola hasta que se ejecutan, si no que si llevan mucho tiempo en una cola pueden ir subiendo a las colas de más prioridad.

4.3. Algoritmos de planificación en multiprocesadores

La mayoría de los procesadores actuales, disponen de varios núcleos, y a su vez son multihilo, en estas arquitecturas, el planificador debe decidir qué hilo se ejecuta. Para ello se aplican los siguientes algoritmos:

- **Tiempo compartido:** Los hilos son tratados como elementos **independientes**.
- **Espacio compartido:** Los hilos de un proceso sólo se ejecutan cuando se pueden ejecutar **todos** a la vez. Este algoritmo funciona mejor que el anterior cuando existe una comunicación muy frecuente entre hilos de un mismo proceso.
- **Planificación por pandilla:** Los hilos son agrupados en pandillas de hilos que actuarán como lotes, formados por aquellos que están **estrechamente vinculados**.

4.4. Implementación de la planificación de procesos en los sistemas operativos actuales

Algoritmo de GNU/Linux: El algoritmo clasifica los hilos en tres niveles:

- Tiempo real
 - Primer nivel: Se aplica el algoritmo FCFS.
 - Segundo nivel: Se aplica un algoritmo Round Robin, con una cola circular.
- El resto de hilo se consideran de tiempo compartido, ejecutándose en un algoritmo de Round Robin ordenando por prioridades.

Algoritmo de Windows: Utiliza un algoritmo de cola múltiples, de 32 colas, y a su vez cada cola es ordenadas mediante un algoritmo de Round Robin.

4.5. Gestión de procesos en Android

Con frecuencia la documentación de Android se refiere a la gestión de procesos como ciclo **de vida** de Android.

Android implementa la **multitarea** mediante la ejecución de **múltiples hilos** para cada proceso; el proceso más simple consta de un sólo hilo principal que sólo es lanzado en el caso de que no exista ninguna instancia del proceso en ejecución.

De igual forma un proceso puede utilizar múltiples hilos, uno por cada subprocesso. También se da la situación en que un proceso comparte subprocessos con otros, lanzando para ello el sistema los hilos necesarios.

4.5.1. Tipos de procesos en Android.

El tipo de proceso en Android determina el comportamiento del planificador respecto a este: no es lo mismo un proceso “activo” y “visible” cuya finalización por parte del sistema tendría una **experiencia muy negativa** para el usuario, que por ejemplo un servicio del sistema.

Android por tanto, clasifica los procesos como:

- **Activos:** Se están ejecutando y además son requeridos para la interacción con el usuario, bien sean procesos en primer plano o servicios dando soporte a éstos.
- **Visibles:** Un nivel por debajo de Activo, ya que un proceso visible estaría en segundo plano.
- **Servicio:** Procesos que no son visibles de forma directa, pero dan un servicio importante al usuario, por ejemplo mantener una conexión a Internet. Estos procesos no deben ser finalizados a no ser que su mantenimiento suponga una merma para un proceso activo.
- **Segundo Plano:** Procesos en ejecución, pero no visibles, a los que el sistema intentará dotar de recursos siempre que haya disponibles.
- **Vacíos:** Es una forma rápida de inicialización: por ejemplo, un proceso que intenta ser lanzado pero el liberador de memoria está ocupado haciéndole hueco.

4.5.2. Planificación de procesos

La planificación en Android es **expropiativa** lo cual debe ser tenido en cuenta por el programador, ya que si no se definen bien los diferentes componentes de la aplicación esta puede ser finalizada mientras está efectuando una tarea relevante para el sistema o para el usuario.

El algoritmo utilizado por el planificador es **Round-Robin con prioridad** basada en una **jerarquía de importancia** según el tipo de proceso (visto anteriormente) y otros factores:

El sistema mantiene una lista pseudo-LRU (Last Recently Used) para evitar que los procesos se apropien de los recursos, pero ordenada según prioridades. La prioridad de un proceso se puede incrementar si este es un subprocesso de mayor prioridad, es decir, la prioridad de un conjunto de procesos interdependientes se iguala a la prioridad del componente más prioritario del conjunto.

Cuando un proceso pasa al estado de espera o de finalización, este se almacena en caché (zRAM). El sentido de almacenar también los procesos finalizados es acelerar su carga en RAM si se vuelven a solicitar.

Android asegura la respuesta de cualquier app lanzada por el usuario, deteniendo y matando a los procesos que impiden la fluidez y liberando recursos para las aplicaciones más prioritarias.

5. Conclusión.

La interacción entre procesos puede derivar en condiciones de competencia, situaciones en las que la perfecta sincronización determina el resultado.

Para evitar estas condiciones de competencia, se introdujo el concepto de sección crítica, que es una sección de código en la que un proceso realiza algo con el estado compartido y no desea que otros procesos trabajen ahí también. Las secciones críticas proporcionan la exclusión mutua.

Existen varias primitivas de la comunicación entre procesos. Entre éstas están los semáforos, los monitores, y la transferencia de mensajes.

Otra labor de la gestión de procesos es la planificación de los mismos. La labor de un algoritmo de planificación es determinar el proceso que debe ejecutarse a continuación, tomando en cuenta factores tales como el tiempo de respuesta, la eficacia y la equidad.

Entre los algoritmos de planificación más conocidos están el round robín, planificación por prioridades, colas de varios niveles, primero el trabajo más corto y la planificación garantizada.

5.1. Relación del tema con el currículo

Este tema es aplicado en el aula en los módulos profesionales siguientes, con las atribuciones docentes indicadas (PES/SAI):

Grado Medio

- Sistemas operativos monopuesto (SMR) (PES/SAI)

Grado Superior

- Sistemas informáticos (DAM / DAW) (PES/SAI)
- Implementación de sistemas operativos (ASIR) (PES/SAI)

6. Bibliografía

- De Anasagasti, Miguel. "Fundamentos de la Computadora" 9^aed 2004 Edt. Paraninfo
- Patterson D.A. y Hennessy JL. "Estructura y diseño de computadoras: la interfaz hardware/software" 4^a Ed. (2005) Edt McGraw-Hill
- Prieto A, Lloris A, Torres JC. "Introducción a la Informática" 4^aed. (2006) Edt. McGraw-Hill
- Stallings W. "Organización y Arquitectura de Computadoras" (2006) 5^a Ed. Edt. Prentice-Hall
- Ramos A, Ramos MJ y Viñas S "Montaje y Mantenimiento de Equipos" (2012). Edt McGraw-Hill
- Jiménez Cembreras, Isabel M^a "Sistemas Informáticos" 2^aEd (2018) Edt. Garceta