Introducción a los Sistemas Operativos
Práctica 4

El objetivo de esta práctica es que el alumno comprenda los aspectos base acerca de la planificación de procesos en un Sistema Operativo (tipos de planificadores, algoritmos y sus variantes, etc.). Además, para la autocorrección de los ejercicios, es deseable la utilización del simulador que se encuentra en cátedras virtuales.

1. Responda en forma sintética sobre los siguientes conceptos:

(a) Programa y Proceso.

Un **programa** es una secuencia de instrucciones u órdenes basadas en un lenguaje de programación que una computadora interpreta para resolver un problema o una función específica.
Los programas son estáticos, es decir, están en la memoria secundaria, no cambian.

Un **proceso** puede entenderse informalmente como un programa en ejecución.
Formalmente un proceso es "Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados".
En la práctica de la materia, los conceptos de 'Proceso', 'Job' y 'Tarea' se usan indistintamente, hacen referencia a lo mismo.

Los procesos son dinámicos. Tienen su propio espacio de memoria, y recursos asignados, y están en ejecución activa, tienen PC (Program Counter), realizando las instrucciones del programa de manera secuencial o concurrente. Los procesos son dinámicos porque cambian constantemente a medida que se ejecutan las instrucciones, interactúan con otros procesos y pueden ser suspendidos, reanudados o finalizados en cualquier momento.

Según su historial de ejecución, los podemos clasificar en:

- **CPU Bound (ligados a la CPU):** En toda su ejecución, solo usan tiempo de la CPU

- **I/O Bound (ligados a entrada/salida):** Ejecutan tiempo en CPU y tiempo en E/S.

(b) Defina Tiempo de retorno (TR) y Tiempo de espera (TE) para un Job.

TR Tiempo de Retorno:

Es el tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución.
Es todo el tiempo de vida del proceso, desde que se crea (New) hasta que termina su ejecución (Terminated)

TE Tiempo de Espera:

Es el tiempo que el proceso se encuentra en el sistema esperando, es decir el tiempo que pasa sin ejecutarse (TR - Tcpu).
Es lo que espera en memoria hasta que el Short Term Scheduller lo selecciona para ser ejecutado.

(c) Defina Tiempo Promedio de Retorno (TPR) y Tiempo promedio de espera (TPE) para un lote de JOBS.

Son el tiempo promedio de los TR y TE para un conjunto de procesos.

(d) ¿Qué es el Quantum?

El quantum es una medida que determina cuánto tiempo podrá usar el procesador cada proceso:

Si el quantum es pequeño**:**

La CPU va a estar mucho tiempo haciendo cambio de contexto, lo que puede generar overhead (si el quantum es muy grande puede estar más tiempo cargando procesos, que tiempo efectivo de ejecución)

Si el quantum es grande:

Con uno demasiado grande se mejora la eficiencia en cuanto el tiempo efectivo de ejecución, pero se aumenta el tiempo de espera en la cola de listo.

(e) ¿Qué significa que un algoritmo de scheduling sea apropiativo o no apropiativo (Preemptive o Non-Preemptive)?

Los algoritmos de planificación se pueden dividir entre Apropiativos y No apropiativos.

No apropiativo: Una vez que un proceso esta en estado de ejecución, continua hasta que termina o se bloquea por algún evento (e.j. I/O).
Nadie lo puede suspender, ni siquiera si viene un proceso con más prioridad.
Siempre se ejecuta hasta el final

Apropiativo: El proceso en ejecución puede ser interrumpido y llevado a la cola de Ready.
Se pierde performance de la CPU, ya que se pierde tiempo haciendo el cambio de contexto, pero mejora el servicio, evita que haya procesos que tarden mucho en ejecutarse .

(f) ¿Qué tareas realizan?: i. Short Term Scheduler ii. Long Term Scheduler iii. Medium Term Scheduler

Los tres mencionados son tipos de planificadores (En inglés Scheduler).
Los planificadores son procesos que se encargan de administrar la multiprogramación. Es decir gestionan y deciden si se pueden cargar nuevos procesos a memoria RAM y cuáles de ellos se van a poder ejecutar, en qué orden y con qué prioridad.

Están diseñados para tener el menor tiempo de respuesta posible y maximizar el rendimiento (evitar que la CPU esté ociosa). Es decir, su objetivo es hacer un uso eficiente del procesador.

El **Long Term Scheduler** se encarga de admitir nuevos procesos a memoria. Selecciona de los nuevos procesos, cuál tiene prioridad o es urgente que se cargue a memoria RAM.

El **Medium Term Scheduler** se encarga de realizar el Swapping (intercambio) entre el disco y la memoria. Es decir, saca procesos de memoria Ram y los poné en disco, disminuyendo el grado de multiprogramación y liberando memoria.

El **Short Term Scheduler** se encarga de determinar que proceso entre los que están en memoria RAM en estado de Ready, pasarán al estado Running (ejecutarse).

El nombre de los Schedulers hace referencia a la frecuencia con la que se ejecutan. El Short Term Scheduler es el que se ejecuta con más frecuencia, y el Long Term Scheduler el que menos.

(g) ¿Qué tareas realiza el Dispatcher?

El Dispatcher es responsable de realizar el cambio de contexto, es decir, cambiar de un proceso a otro en la CPU. Se encarga de guardar el estado del proceso que se está desalojando y cargar el estado del proceso que se está ejecutando en la CPU. También gestiona la asignación y liberación de recursos necesarios para la ejecución del proceso.

2. Procesos:
(a) Investigue y detalle para que sirve cada uno de los siguientes comandos. (Puede que algún comando no venga por defecto en su distribución por lo que deberá instalarlo):

i. top: Este comando se usa para mostrar los procesos de Linux. Proporciona una vista dinámica en tiempo real del sistema en ejecución. Por lo general, este comando muestra la información resumida del sistema y la lista de procesos o subprocesos que actualmente administra el Kernel de Linux.

ii. htop: Es una alternativa a top, que ofrece una interfaz más sencilla que incluye colores, manejo con el mouse y la posibilidad de desplazarse por los procesos.

iii. ps: Este comando te ofrece una visión general de todos los procesos que se están ejecutando en un determinado momento.

iv. pstree: Muestra los procesos en ejecución como un árbol, organizados en una relación padre-hijo.

v. kill: Permite terminar un proceso utilizando su PID (Process ID).

vi. pgrep: Este comando se usa para buscar procesos en función del nombre o el PID

pkill: Este comando se usa para terminar o eliminar el proceso en función del nombre o el PID

vii. killall: Este comando mata todos los procesos que tienen determinado nombre.

viii. renice: Altera la prioridad de los procesos en ejecución.

ix. xkill: Este comando permite cerrar ventanas de X Server.

x. atop: Es una herramienta para monitorear los recursos del sistema en Linux. Muestra mucha información relacionada con la cantidad de carga en los recursos del sistema a nivel de proceso

(b) Observe detenidamente el siguiente código. Intente entender lo que hace sin necesidad de ejecutarlo.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main ( void ) {
        int c;
        pid_t pid;
        printf ( " Comienzo . : \ n " ) ;
        for ( c = 0; c < 3;  c++ ) {
                pid = fork ( ) ;
        }
        printf ( " Proceso \n " );
        return 0 ;
}
```

i. ¿Cuántas líneas con la palabra “Proceso” aparecen al final de la ejecución de este programa?.

Aparece 8 veces.
fork() es una función del sistema que se utiliza para crear un nuevo proceso. Cuando se llama a fork(), se crea una copia idéntica del proceso que llama, y ambos procesos (el original y el nuevo) continúan ejecutándose a partir de **ese** punto.

ii. ¿El número de líneas es el número de procesos que han estado en ejecución?.

Si, efectivamente.
Todos al final imprimen “Proceso” para luego terminar.

Ejecute el programa y compruebe si su respuesta es correcta, Modifique el valor del bucle for y compruebe los nuevos resultados.

(c) Vamos a tomar una variante del programa anterior. Ahora, además de un mensaje, vamos a añadir una variable y, al final del programa vamos a mostrar su valor. El nuevo código del programa se muestra a continuación.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main ( void ) {
        int c;
        int p=0;
        pid_t pid;
        for ( c = 0; c < 3; c++ ) {
                pid = fork( ) ;
        }
        p++;
        printf ( "Proceso %d\n " , p );
        return 0;
}
```

i. ¿Qué valores se muestran por consola?.

“Proceso 1”

ii. ¿Todas las líneas tendrán el mismo valor o algunas líneas tendrán valores distintos?.

Todas las líneas tienen el mismo valor. Ya que el incremento de la variable P se hace fuera del for.
A la hora del fork(), que se realiza dentro del for, p vale 0 en todos los casos. Se incrementa luego y se imprime con el 1

iii. ¿Cuál es el valor (o valores) que aparece?.

Siempre “Proceso 1”, 8 veces.

Ejecute el programa y compruebe si su respuesta es correcta, Modifique el valor del bucle for y el lugar dónde se incrementa la variable p y compruebe los nuevos resultados.

* no lo probé

(d) Comunicación entre procesos:

i. Investigue la forma de comunicación entre procesos a través de pipes.

Los pipes son una forma de comunicación entre procesos en sistemas Unix y Linux. Se utilizan para establecer una canal de comunicación unidireccional entre dos procesos, permitiendo que uno escriba datos en el pipe y el otro los lea

ii. ¿Cómo se crea un pipe en C?.

En C, un pipe se crea utilizando la función pipe(). Esta función se encuentra en la biblioteca unistd.h.

iii. ¿Qué parámetro es necesario para la creación de un pipe?. Explique para qué se utiliza

El parámetro necesario para la creación de un pipe es una matriz de dos enteros. Esta matriz contendrá el descriptor de lectura (`filedes[0]`) y el descriptor de escritura (`filedes[1]`) de la pipe creada.
Estos descriptores son utilizados para las operaciones de lectura y escritura del pipe

iv. ¿Qué tipo de comunicación es posible con pipes?

Los pipes permiten la comunicación unidireccional de tipo "pipe anónimo" entre procesos. Es decir, se puede transmitir datos desde un proceso padre a un proceso hijo o viceversa, pero no en ambas direcciones a la vez. Los datos se escriben en un extremo del pipe y se leen desde el otro extremo, lo que permite la transferencia de información entre procesos

(e) ¿Cuál es la información mínima que el SO debe tener sobre un proceso?¿En que estructura de datos asociada almacena dicha información?

La información mínima que el sistema operativo debe tener sobre un proceso es su estado actual, identificador, prioridad, ubicación en memoria y recursos utilizados. Esta información se almacena en una estructura de datos llamada Bloque de Control del Proceso (PCB). El PCB es una estructura de datos que contiene información sobre cada proceso. Existe una por proceso. Es lo primero que se crea cuando se realiza un *fork*, y lo último que se desaloca cuando se termina.

(f) ¿Qué significa que un proceso sea “CPU Bound” y “I/O Bound”?

CPU Bound (ligados a la CPU):

En toda su ejecución, solo usan tiempo de la CPU. O al menos la gran mayor parte del tiempo.

I/O Bound (ligados a entrada/salida):

Ejecutan tiempo en CPU y tiempo en E/S. Son ralentizados por las operaciones de E/S.

(g) ¿Cuáles son los estados posibles por los que puede atravesar un proceso?

Listo (Ready):

En este estado, el proceso está listo para ejecutarse, pero aún no se le ha asignado tiempo de CPU. Espera en la cola de procesos listos hasta que el planificador de procesos (Short Term Scheduler) le asigne tiempo de CPU.

Ejecución (Running):

El proceso se encuentra en este estado cuando se está ejecutando en la CPU.

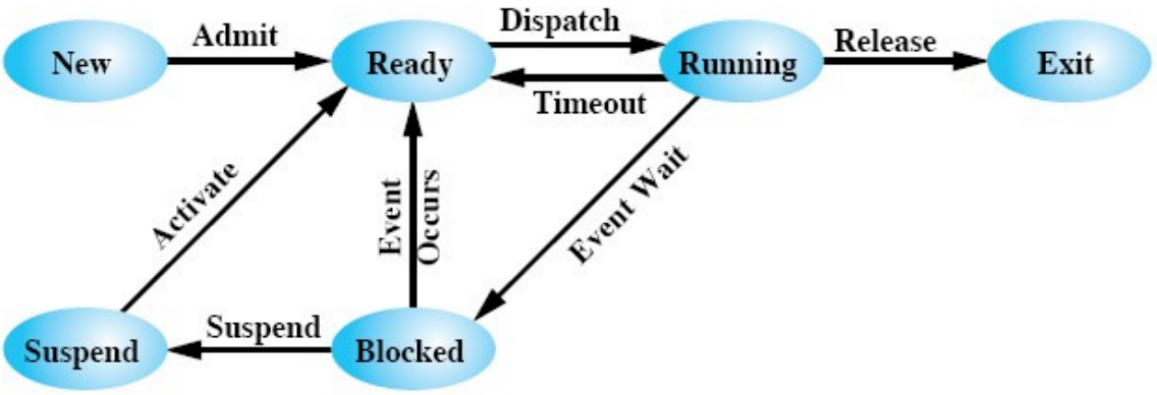
Bloqueado (Blocked):

También conocido como estado de espera, un proceso puede entrar en este estado cuando necesita esperar a que ocurra algún evento, como la finalización de una operación de E/S (entrada/salida) o la recepción de datos. El proceso se bloquea y no utiliza la CPU hasta que el evento se complete.

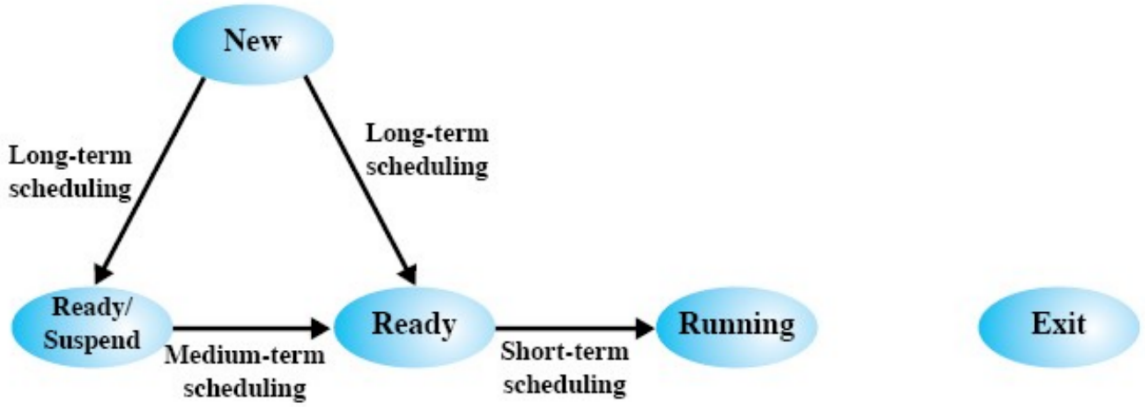
Terminado (Terminated):

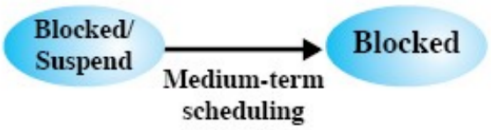
Un proceso llega a este estado cuando ha completado su ejecución o ha sido terminado por el sistema operativo. En este estado, los recursos utilizados por el proceso se liberan y se eliminan de la lista de procesos activos.

(h) Explique mediante un diagrama las posibles transiciones entre los estados.



(i) ¿Que scheduler de los mencionados en 1f se encarga de las transiciones?





3. Para los siguientes algoritmos de scheduling:
FCFS (Fisrt Coome First Served)
SJF (Shortest Job First)
Round Robin Prioridades

(a) Explique su funcionamiento mediante un ejemplo.

FCFS (First-Come-First-Served):

- **Funcionamiento:** Este algoritmo programa los procesos en el orden en el que llegan a la cola de procesos listos. El primer proceso que llega es el primero en ser servido. No se considera la duración del proceso en este método.
- **Ejemplo:** Supongamos que llegan tres procesos en el siguiente orden: A, B, C. El algoritmo FCFS los atenderá en ese mismo orden, sin tener nada más en cuenta
  .

SJF (Shortest Job First):

- **Funcionamiento:** El algoritmo SJF elige el proceso que tenga el menor tiempo de ejecución para ser ejecutado a continuación (cálculo basado en la ejecución previa). Esto minimiza el tiempo de espera total.
- **Ejemplo:** Si tenemos tres procesos con tiempos de ejecución restantes de 5 ms, 3 ms y 8 ms, el algoritmo SJF seleccionaría el proceso con 3 ms para ser ejecutado primero.

Round Robin:

- **Funcionamiento:** Round Robin es un algoritmo de asignación de tiempo en el que cada proceso se ejecuta durante un quantum de tiempo determinado y luego se mueve al final de la cola de procesos listos. Esto garantiza que todos los procesos obtengan un tiempo de CPU justo y evita que un proceso monopolice el CPU.
- **Ejemplo:** Si se establece un quantum de 10 ms y hay tres procesos A, B y C, se ejecutarán en el orden A, B, C, A, B, C, y así sucesivamente, asignando 10 ms a cada proceso en cada ciclo.

Prioridades:

- **Funcionamiento:** Los procesos se programan según su prioridad, donde la prioridad más alta se ejecuta primero. Puede haber varias políticas de asignación de prioridades, como prioridades fijas o dinámicas basadas en factores como el tiempo de llegada, la duración o la importancia.
- **Ejemplo:** Si tenemos tres procesos con prioridades 1, 2 y 3, el proceso con prioridad 3 se ejecutará antes que los otros dos.

(b) ¿Alguno de ellos requiere algún parámetro para su funcionamiento?

- **FCFS:** No requiere ningún parámetro adicional.
- **SJF:** Requiere conocer el tiempo de ejecución de cada proceso por adelantado o estimarlo de alguna manera.
- **Round Robin:** Requiere especificar el valor del quantum de tiempo.
- **Prioridades:** Requiere asignar una prioridad a cada proceso.

(c) Cuál es el más adecuado según los tipos de procesos y/o SO.

FCFS: No tiene en cuenta ningún factor más que el orden. No favorece a ningún tipo de procesos, pero en principio podríamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no.

SJF: Es más adecuado para procesos I/O Bound, ya que al ser No apropiativo, los procesos más largos podrían sufrir inanición si nunca se les asigna la CPU.

Round Robin: Considero que es bueno para ambos tipos de procesos. Es útil en sistemas mixtos con procesos CPU-bound e I/O-bound, ya que evita que un proceso monopolice la CPU y puede proporcionar tiempos de respuesta predecibles para procesos I/O-bound

Prioridades: No favorece a ningún tipo en particular. Es adecuado para sistemas donde se necesita asignar prioridades en función de la importancia relativa de los procesos. Puede adaptarse a las necesidades tanto de procesos CPU-bound como I/O-bound asignando prioridades en consecuencia

(d) Cite ventajas y desventajas de su uso.

FCFS:
Es simple y fácil de entender, pero no hace una selección lógica.

SJF:
Los procesos cortos se seleccionan antes que los procesos largos para minimizar el tiempo de espera y mejorar el tiempo de respuesta. Sin embargo, uno de los problemas es que los procesos largos pueden sufrir inanición, lo que significa que pueden quedar en la cola de procesos listos durante mucho tiempo sin ser atendidos si hay una corriente constante de procesos cortos que llegan.

Round Robin:
Proporciona equidad en el tiempo de CPU para todos los procesos, lo malo es que puede llevar a un alto tiempo de respuesta en procesos largos debido a la conmutación frecuente

Prioridades:
Permite asignar recursos en función de la importancia relativa de los procesos, pero puede llevar a la inanición de procesos con prioridades bajas si se asignan prioridades incorrectamente o si hay una corriente constante de procesos con mayor prioridad que llegan.

4. Para el algoritmo Round Robin, existen 2 variantes:

- **Timer Fijo**
- **Timer Variable**

(a) ¿Qué significan estas 2 variantes?

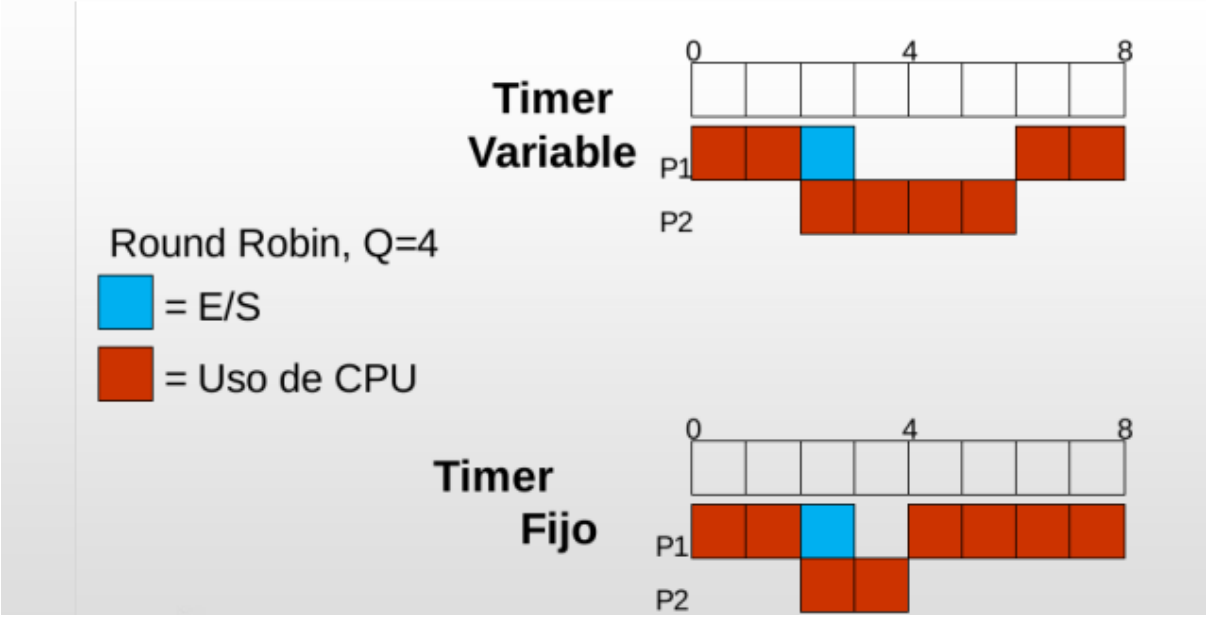
Con Timer Variable, el contador se inicializa en Q (el valor determinado de Quantum, por ejemplo en 4) cuando el proceso es asignado a la CPU, es decir cuando pasa al estado de Running.
Es el que más se usa.

En el caso del Timer Fijo, el contador se inicializa en Q cuando su valor llega a cero (y NO cuando un proceso es cargado para ejecutarse). Se puede ver como un valor de Q compartido entre los procesos. Es decir, si tenemos un Q de 10, y un proceso usó 6 de tiempo, el siguiente proceso tendría solo 4, y al llegar a 0 se le quita la CPU. Se puede ver como un valor Q de tiempo compartido entre los procesos.

(b) Explique mediante un ejemplo sus diferencias.

En este ejemplo, en Timer Variable, cuando el proceso 1 se va a E/S, el proceso 2 usa sus 4 quantums.

Con Timer Fijo, cuando cuando el proceso 1 se va a E/S, el proceso 2 se usa solo 2 quantums, cuando se acaban se le quita la CPU, se reinicia el Timer a 4 y se otorga la CPU a otro proceso (en este caso el 1 nuevamente, pero podría haber sido un proceso 3).



(c) En cada variante ¿Dónde debería residir la información del Quantum?

En el caso del Timer Fijo, el contador debería ser global, por lo que debería estar en algún que sea accesible por el Sistema Operativo.

En el caso del Timer Variable, el contador es local a cada proceso, y se guarda en la PCB de cada uno.

5. Se tiene el siguiente lote de procesos que arriban al sistema en el instante 0 (cero):

Job 1: 7 unidades de CPU
Job 2: 15 unidades de CPU
Job 3: 12 unidades de CPU
Job 4: 4 unidades de CPU
Job 5: 9 unidades de CPU

(a) Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:

i. FCFS (First Come, First Served)
ii. SJF (Shortest Job First)
iii. Round Robin con quantum = 4 y Timer Fijo
iv. Round Robin con quantum = 4 y Timer Variable

(b) Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.

(c) En base a los tiempos calculados compare los diferentes algoritmos.

Los algoritmos no apropiativos (FCFS y SJF) tienen un menor tiempo promedio. Sin embargo los algoritmos que estén últimos en la cola tienen muchísimo tiempo de espera. En este ejercicio son sólo 5 procesos por lo que no hay problema, pero con una cantidad grande de jobs puede ser un problema con posibilidad de generar inanición.

6. Se tiene el siguiente lote de procesos:

Job 1 | Llegada: 0 | 4 unidades de CPU
Job 2 | Llegada: 2 | 6 unidades de CPU

Job 3 | Llegada: 3 | 4 unidades de CPU
Job 4 | Llegada: 6 | 5 unidades de CPU
Job 5 | Llegada: 8 | 2 unidades de CPU

(a) Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:

i. FCFS (First Come, First Served)
ii. SJF (Shortest Job First)
iii. Round Robin con quantum = 1 y Timer Variable
iv. Round Robin con quantum = 6 y Timer Variable

(b) Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.

(c) En base a los tiempos calculados compare los diferentes algoritmos.

(d) En el algoritmo Round Robin, qué conclusión se puede sacar con respecto al valor del quantum.

Se llega a la misma conclusión que antes.
A mayor tiempo de CPU para cada proceso (como en el caso de los algoritmos no apropiativos, o un RR con Quantum alto) los en comparación se reducen. Sin embargo, si algún proceso tiene mucho tiempo de espera, podría generar inanición.
En el caso de este ejercicio, el Quantum = 6 fue suficiente para que el RR se comporte exactamente igual al FCFS.
Con un Quantum bajo, se “pierde” tiempo efectivo de CPU realizando el cambio de contexto, pero el tiempo de CPU se reparte de manera más equitativa entre los procesos y es más difícil que se genere inanición.

(e) ¿Para el algoritmo Round Robin, en que casos utilizaría un valor de quantum alto y que ventajas y desventajas obtendría?

Un round robin alto lo usaría en casos donde haya poca cantidad de procesos, y que los que hayan requieran mucho tiempo de CPU.
El quantum bajo es mejor para casos donde haya alto grado de multiprogramación, por ejemplo un pc hogareño que hace varias tareas a la vez. Un quantum alto, es mejor para máquinas que necesitan un alto procesamiento pero poca interacción del usuario.

Como ejemplo, Windows Home tiene un Quantum de 3, mientras que
Windows Server tiene un Quantum de 12.

7. Una variante al algoritmo SJF es el algoritmo SJF apropiativo o SRTF (Shortest Remaining Time First):

(a) Realice el diagrama del Gantt para este algoritmo según el lote de trabajos del ejercicio 6.

(b) ¿Nota alguna ventaja frente a otros algoritmos?

En este caso, con el mismo lote de jobs del ejercicio 6, me dio exactamente el mismo resultado que el SJF, por lo que no, con este lote específico de procesos como ejemplo, no pude notar alguna ventaja con respecto al SFJ.

Imagino que al dar prioridad a los procesos más cortos incluso si llegan después del inicio de otro proceso ajustando en tiempo real, minimiza la probabilidad de inanición de los procesos más cortos.
En contraparte, si vienen muchos procesos cortos, podría generar inanición de los procesos más largos, como ocurría en SJF.

8. Suponga que se agregan las siguientes prioridades al lote de procesos del ejercicio 6, donde un menor número indica mayor prioridad:

Job 1 | Prioridad: 3
Job 2 | Prioridad: 4
Job 3 | Prioridad: 2
Job 4 | Prioridad: 1
Job 5 | Prioridad: 2

(a) Realice el diagrama de Gantt correspondiente al algoritmo de planificación por prioridades según las variantes:

i. No Apropiativa
ii. Apropiativa

(b) Calcule el TR y TE para cada job así como el TPR y el TPE.

(c) ¿Nota alguna ventaja frente a otros algoritmos? Bajo que circunstancias lo utilizaría y ante que situaciones considera que la implementación de prioridades podría no ser de mayor relevancia?

La ventaja es que podemos asignar una prioridad relativa y subjetiva a los jobs, para procesos que consideremos importantes por algún motivo o para procesos que son realmente críticos para el sistema.
No lo considero relevante en sistemas en los que se necesite un acceso realmente equitativo a los recursos. Por ejemplo, en Cloud Computing, todos los usuarios deberían tener el mismo tiempo de CPU para que sea justo.

9. Inanición (Starvation)

(a) ¿Qué significa?

(a) La inanición (starvation en inglés) es un problema que se genera cuando a un proceso se le niega durante un tiempo prolongado el acceso a la CPU. Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada.

(b) ¿Cuál/es de los algoritmos vistos puede provocarla?

Todos los algoritmos podrían provocar inanición:

FCFS (First Come, First Served):
Si un proceso de larga duración llega primero, los procesos más cortos tendrán que esperar mucho tiempo para su ejecución.

SJF (Shortest Job First) y SRTF (Shortest Remaining Time First):
Si hay procesos más cortos que llegan constantemente, los procesos más largos nunca se ejecutarán.

Round Robin:
Si el quantum es demasiado grande, los procesos más cortos tendrán que esperar mucho tiempo para su ejecución.

Planificación por prioridades:
Si hay procesos de alta prioridad que llegan constantemente, los procesos de baja prioridad nunca se ejecutarán.

(c) ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en b?

FCFS (First Come, First Served):
No hay solución ya que este algoritmo no selecciona bajo ningún criterio.

SJF (Shortest Job First) y SRTF (Shortest Remaining Time First):
No hay mucho que hacer.

Round Robin:
Una solución podría ser utilizar un quantum más pequeño, bajando el riesgo de inanición.

Planificación por prioridades:
Asignar de manera distinta las prioridades.

10. Los procesos, durante su ciclo de vida, pueden realizar operaciones de I/O como lecturas o escrituras a disco, cintas, uso de impresoras, etc.

El SO mantiene para cada dispositivo, que se tiene en el equipo, una cola de procesos que espera por la utilización del mismo (al igual que ocurre con la Cola de Listos y la CPU, ya que la CPU es un dispositivo más).

Cuando un proceso en ejecución realiza una operación de I/O el mismo es expulsado de la CPU y colocado en la cola correspondiente a el dispositivo involucrado en la operación.

El SO dispone también de un "I/O Scheduling" que administra cada cola de dispositivo a través de algún algoritmo (FCFS, Prioridades, etc.). Si al colocarse un proceso en la cola del dispositivo, la misma se encuentra vacía el mismo será atendido de manera inmediata, caso contrario, deberá esperar a que el SO lo seleccione según el algoritmo de scheduling establecido.

Los mecanismos de I/O utilizados hoy en día permiten que la CPU no sea utilizada durante la operación, por lo que el SO puede ejecutar otro proceso que se encuentre en espera una vez que el proceso bloqueado por la I/O se coloca en la cola correspondiente.

Cuando el proceso finaliza la operación de I/O el mismo retorna a la cola de listos para competir nuevamente por la utilización de la CPU.

Para los siguientes algoritmos de Scheduling:

- **FCFS**
- **Round Robin con quantum = 2 y timer variable.**

Y suponiendo que la cola de listos de todos los dispositivos se administra mediante FCFS, realice los diagramas de Gantt según las siguientes situaciones:

(a) Suponga que al lote de procesos del ejercicio 6 se agregan las siguientes operaciones de entrada salida:

Job 1 | I/O: (R1, 2, 1)
Job 2 | I/O: (R2, 3, 1) (R2, 5, 2)
Job 4 | I/O: (R3, 1, 2) (R3, 3, 1)

(b) Suponga que al lote de procesos del ejercicio 6 se agregan las siguientes operaciones de entrada salida:

Job 1 | I/O: (R1, 2, 3) (R1, 3, 2)
Job 2 | I/O: (R2, 3, 2)
Job 3 | I/O: (R2, 2, 3)
Job 4 | I/O: (R1, 1, 2)

11. Algunos algoritmos pueden presentar ciertas desventajas cuando en el sistema se cuenta con procesos ligados a CPU y procesos ligados a entrada salida. Analice las mismas para los siguientes algoritmos:

(a) Round Robin

En un entorno donde coexisten procesos ligados a la CPU y procesos ligados a entrada/salida, el algoritmo Round Robin asegura una distribución equitativa del tiempo de la CPU entre los procesos, lo que proporciona un trato justo a todos los procesos del sistema.

Lo malo de esto es que el RR con un quantum grande sería negativo para los procesos ligados a E/S, ya que no prioriza la necesidad de E/S (como lo podría hacer el algoritmo de prioridades) y puede resultar en retrasos para estos procesos I/O Bound.
También podría ocurrir que en el primer o segundo quantum de uso de CPU de un proceso, deba realizar una operación de I/O, perdiendo su quantum asignado y retrasando aún más su retorno. (Ejemplo: Proceso 2 del segundo Diagrama de Gantt del ejercicio 10, en su instrucción 3)

Si contamos con un quantum pequeño, la conmutación frecuente entre procesos se perdería mucho tiempo realizando el cambio de contexto del procesador, lo que puede ser negativo para procesos CPU Bound.

(b) SRTF (Shortest Remaining Time First)

El algoritmo SRTF (Shortest Remaining Time First) prioriza la ejecución de procesos más cortos en términos de tiempo de ejecución restante. Esto conduce a tiempos de espera menores y un rendimiento eficiente en términos de tiempo de retorno, especialmente para procesos más cortos. Sin embargo, la desventaja es que puede generar inanición para los procesos más largos, ya que se prioriza constantemente la ejecución de tareas más cortas, por lo que no sería adecuado para procesos que requieran mucho tiempo de CPU (CPU Bound)

12. Para equiparar la desventaja planteada en el ejercicio 11), se plantea la siguiente modificación al algoritmo: Algoritmo VRR (Virtual Round Robin):
Este algoritmo funciona igual que el Round Robin, con la diferencia que cuando un proceso regresa de una I/O se coloca en una cola auxiliar. Cuando se tiene que tomar el próximo proceso a ejecutar, los procesos que se encuentra en la cola auxiliar tienen prioridad sobre los otros. Cuando se elije un proceso de la cola auxiliar se le otorga el procesador por tantas unidades de tiempo como le falto ejecutar en su ráfaga de CPU anterior, esto es, se le otorga la CPU por un tiempo que surge entre la diferencia del quantum original y el tiempo usado en la última ráfaga de CPU.

(a) Analice el funcionamiento de este algoritmo mediante un ejemplo. Marque en cada instante en que cola se encuentran los procesos.

(b) Realice el ejercicio 10)a) nuevamente considerando este algoritmo, con un quantum de 2 unidades y Timer Variable.

13. Suponga que un SO utiliza un algoritmo de VRR con Timer Variable para el planificar sus procesos. Para ello, el quantum es representado por un contador, que es decrementado en 1 unidad cada vez que ocurre una interrupción de reloj. ¿Bajo este esquema, puede suceder que el quantum de un proceso nunca llegue a 0 (cero)? Justifique su respuesta.

Puede suceder que el proceso se complete antes de que se agote su quantum. Cuando un proceso termina su ejecución antes de que se agote su quantum, no se requerirá que se reponga el quantum para ese proceso específico, ya que el proceso ya habrá completado su ejecución. En este caso, el quantum para **ese proceso en particular** nunca llegará a cero, ya que finaliza su ejecución antes de que se agote su quantum.

14. El algoritmo SJF (y SRTF) tiene como problema su implementación, dada la dificultad de conocer la duración de la próxima ráfaga de CPU.
Es posible realizar una estimación de la próxima, utilizando la media de las ráfagas de CPU para cada proceso.
Así, por ejemplo, podemos tener la siguiente fórmula:

$$S_n+1 = 1\ n\ T_n + n - 1\ n\ S_n$$

Donde:

T_i = duración de la ráfaga de CPU i-ésima del proceso.

S_i = valor estimado para el i-ésimo caso

S_i = valor estimado para la primer ráfaga de CPU. No es calculado.

(a) Suponga un proceso cuyas ráfagas de CPU reales tienen como duración: 6, 4, 6, 4, 13, 13, 13 Calcule que valores se obtendrían como estimación para las ráfagas de CPU del proceso si se utiliza la formula 1, con un valor inicial estimado de S1=10. La formula anterior 1 le da el mismo peso a todos los casos (siempre calcula la media). Es posible reescribir la formula permitiendo darle un peso mayor a los casos mas recientes y menor a casos viejos (o viceversa). Se plantea la siguiente formula:

$$S_n+1 = \alpha T_n + (1 - \alpha) S_n \quad (2)$$

Con $0 < \alpha < 1$

(b) Analice para que valores de α se tienen en cuenta los casos mas recientes.

(c) Para la situación planteada en a) calcule que valores se obtendrían si se utiliza la formula 2 con α = 0, 2; α = 0, 5 y α = 0, 8.

(d) Para todas las estimaciones realizadas en a y c ¿Cuál es la que mas se asemeja a las ráfagas de CPU reales del proceso?

15. Colas Multinivel Hoy en día los algoritmos de planificación vistos se han ido combinando para formar algoritmos más eficientes. Así surge el algoritmo de Colas Multinivel, donde la cola de procesos listos es divida en varias colas, teniendo cada una su propio algoritmo de planificación.

(a) Suponga que se tienen dos tipos de procesos: Interactivos y Batch. Cada uno de estos procesos se coloca en una cola según su tipo. ¿Qué algoritmo de los vistos utilizaría para administrar cada una de estas colas?. A su vez, se utiliza un algoritmo para administrar cada cola que se crea. Así, por ejemplo, el algoritmo podría determinar mediante prioridades sobre que cola elegir un proceso.

Procesos interactivos: Round Robin
Batch: FCFS

(b) Para el caso de las dos colas vistas en a: ¿Qué algoritmo utilizaría para planificarlas?

Utilizaría el algoritmo de prioridades

16. Suponga que en un SO se utiliza un algoritmo de planificación de colas multinivel. El mismo cuenta con 3 colas de procesos listos, en las que los procesos se encolan en una u otra según su prioridad. Hay 3 prioridades (1 , 2 , 3), donde un menor número indica mayor prioridad. Se utiliza el algoritmo de prioridades para la administración entre las colas. Se tiene el siguiente lote de procesos a ser procesados con sus respectivas operaciones de I/O: Suponiendo que las colas de cada dispositivo se administran a trabes de FCFS y que cada cola de procesos listos se administra por medio de un algoritmo RR con un quantum de 3 unidades y Timer Variable, realice un diagrama de Gantt:

(a) Asumiendo que NO hay apropiación entre los procesos.

(b) Asumiendo que hay apropiación entre los procesos.

17. En el esquema de Colas Multinivel, cuando se utiliza un algoritmo de prioridades para administrar las diferentes colas los procesos pueden sufrir starvation. La técnica de envejecimiento se puede aplicar a este esquema, haciendo que un proceso cambie de una cola de menor prioridad a una de mayor prioridad, después de cierto periodo de tiempo que el mismo se encuentra esperando en su cola. Luego de llegar a una cola en la que el proceso llega a ser atendido, el mismo retorna a su cola original. Por ejemplo: Un proceso con prioridad 3 esta en cola su cola correspondiente. Luego de X unidades de tiempo, el proceso se mueve a la cola de prioridad 2. Si en esta cola es atendido, retorna a su cola original, en caso contrario luego de sucederse otras X unidades de tiempo el proceso de mueve a la cola de prioridad 1. Esta última acción se repite hasta que el proceso obtiene la CPU, situación que hace que el mismo vuelva a su cola original.

(a) Para los casos a y b del ejercicio 16 realice el diagrama de Gantt considerando además que se tiene un envejecimiento de 4 unidades.

18. La situación planteada en el ejercicio 17, donde un proceso puede cambiar de una cola a otra, se la conoce como Colas Multinivel con Realimentación.

Suponga que se quiere implementar un algoritmo de planificación que tenga en cuenta el tiempo de ejecución consumido por el proceso, penalizando a los que más tiempo de ejecución tienen. (Similar a la tarea del algoritmo SJF que tiene en cuenta el tiempo de ejecución que resta). Utilizando los conceptos vistos de Colas Multinivel con Realimentación indique que colas implementaría, que algoritmo usaría para cada una de ellas así como para la administración de las colas entre sí.

Tenga en cuenta que los procesos no deben sufrir inanición.

El algoritmo de colas multinivel con realimentación se puede definir por los siguientes parámetros:

1- El número de colas.
2- El algoritmo de planificación de cada cola.
3- El algoritmo de planificación entre las distintas colas.
4- El método usado para determinar cuándo pasar un proceso a una cola de prioridad más alta.
5- El método usado para determinar cuándo pasar un proceso a una cola de prioridad más baja.
6- El método usado para determinar en qué cola se introducirá un proceso cuando haya que darle servicio

Para implementar un algoritmo de planificación que considere el tiempo de ejecución consumido por el proceso y penalice a los que más tiempo consumen, yo elegiría:

Número de colas:

Se pueden usar 3 colas, una para procesos de alta prioridad, otra para procesos de prioridad media y otra para procesos de baja prioridad.

Algoritmo de planificación de cada cola:

Para la cola de alta prioridad, se puede usar Round Robin con un quantum corto, para asegurarse de que todos los procesos tengan acceso justo a los recursos de la CPU.

Para la cola de prioridad media, se puede usar Round Robin con un quantum intermedio, por el mismo motivo que el anterior, pero dando un poco más de tiempo a cada proceso

Para la cola de baja prioridad, se puede usar FCFS para que los procesos se ejecuten en el orden en que llegan.

Algoritmo de planificación entre las distintas colas:

Los procesos que han consumido poco tiempo de CPU podrían ser promovidos a una cola de prioridad más alta, mientras que los procesos que han consumido mucha CPU podrían ser degradados a una cola de prioridad más baja.

Método usado para determinar cuándo pasar un proceso a una cola de prioridad más alta:

Puede considerarse un umbral de tiempo en el que, si un proceso ha excedido ese tiempo de ejecución en la cola de alta prioridad, se promueva degrade a la cola de menor prioridad.

Método usado para determinar cuándo pasar un proceso a una cola de prioridad más baja:

Puede considerarse un umbral de tiempo de inactividad en la cola de baja prioridad, y si un proceso ha estado inactivo durante ese tiempo, puede ser promovido a la cola de mayor prioridad.

Método usado para determinar en qué cola se introducirá un proceso cuando haya que darle servicio:

Se puede utilizar un enfoque híbrido que tenga en cuenta tanto la prioridad del proceso como su tiempo de ejecución actual. Por ejemplo, si un proceso tiene una prioridad alta pero ha consumido mucho tiempo de CPU, podría ser colocado en la cola de prioridad media para dar oportunidades a otros procesos de alta prioridad

19. Un caso real: “Unix Clasico “ (SVR3 y BSD 4.3)

Estos sistemas estaban dirigidos principalmente a entornos interactivos de tiempo compartido. El algoritmo de planificación estaba diseñado para ofrecer buen tiempo de respuesta a usuarios interactivos y asegurar que los trabajos de menor prioridad (en segundo plano) no sufrieran inanición.

La planificación tradicional usaba el concepto de colas multinivel con realimentación, utilizando RR para cada uno de las colas y realizando el cambio de proceso cada un segundo (quantum). La prioridad de cada proceso se calcula en función de la clase de proceso y de su historial de ejecución. Para ello se aplican las siguientes funciones:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2} \tag{3}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j \tag{4}$$

donde:

CPU_j (i) = Media de la utilización de la CPU del proceso j en el intervalo i.

P_j (i) = Prioridad del proceso j al principio del intervalo i (los valores inferiores indican prioridad más alta).

$Base_j$ = Prioridad base del proceso j.

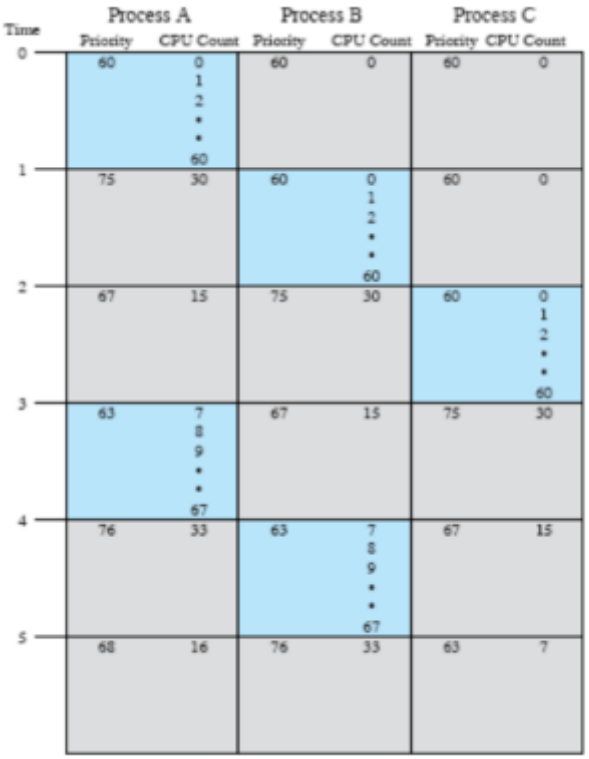
$Nice_j$ = Factor de ajuste.

La prioridad del proceso se calcula cada segundo y se toma una nueva decisión de planificación. El propósito de la prioridad base es dividir los procesos en bandas fijas de prioridad. Los valores de CPU y nice están restringidos para

impedir que un proceso salga de la banda que tiene asignada. Las bandas definidas, en orden decreciente de prioridad, son:

- **Intercambio**
- **Control de Dispositivos de I/O por bloques**
- **Gestión de archivos**
- **Control de Dispositivos de I/O de caracteres**
- **Procesos de usuarios**

Veamos un ejemplo: Supongamos 3 procesos creados en el mismo instante y con prioridad base 60 y un valor nice de 0. El reloj interrumpe al sistema 60 veces por segundo e incrementa un contador para el proceso en ejecución.
Los sectores en celeste representan el proceso en ejecución.



(a) Analizando la jerarquía descripta para las bandas de prioridades: ¿Que tipo de actividad considera que tendrá más prioridad? ¿Por qué piensa que el scheduler prioriza estas actividades?

Se prioriza el intercambio de datos en memoria ram, así como las operaciones de E/S en bloques (Generalmente un disco duro) o la gestión de archivos… Es decir, evidentemente se priorizan los procesos del kernel de SO.

Si bien puede parecer contradictorio que los procesos del usuario se encuentren últimos en la banda de prioridades, el scheduler prioriza las actividades antes mencionadas ya que son críticas para el funcionamiento fluido del sistema, además de que requieren poco tiempo de CPU y de forma poco frecuente, por tanto se realizan rápidamente, evitando que sufran inanición y pasando a dar CPU a los procesos del usuario.

(b) Para el caso de los procesos de usuarios, y analizando las funciones antes descriptas: ¿Qué tipo de procesos se encarga de penalizar? (o equivalentemente se favorecen). Justifique

La prioridad en modo usuario depende de dos factores: el factor de amabilidad ($Nice_j$) y el tiempo de uso de la CPU (CPU_j).

El factor de amabilidad es un número entero entre 0 y 39. Su valor por defecto es 20. Se denomina factor de amabilidad, porque un usuario incrementando este valor está disminuyendo la prioridad de sus procesos y en consecuencia le está cediendo el turno de uso de CPU a los procesos de otros usuarios. A los procesos en segundo plano el Kernel les asigna de forma automática un factor de amabilidad elevado.

El campo CPU_j almacena una medida del uso de la CPU por parte del proceso. Este campo se inicializa a 0 cuando el proceso es creado. En cada tic, la rutina de tratamiento de la interrupción del reloj incrementa CPU_j para el proceso actualmente en ejecución, hasta un máximo de 127.
Además, cada segundo, el núcleo invoca a una rutina denominada *schedcpu* que reduce el valor de CPU_j de un proceso mediante un factor de disminución (decay)

Este esquema evita que los procesos de baja prioridad nunca lleguen a ser ejecutados. También favorece a los procesos limitados por E/S (procesos que requieren muchas operaciones de E/S, por ejemplo, las consolas de comandos y los editores de texto) en contraposición a los procesos limitados por la CPU (procesos que requieren mucho uso de la CPU, por ejemplo, los compiladores). Un proceso limitado por E/S, mantiene una alta prioridad ya que su tiempo de uso de la CPU es pequeño, y recibe tiempo de CPU rápidamente cuando la necesita. Por contra, los procesos limitados por la CPU tienen valores de tiempo de uso de la CPU altos y por tanto una baja prioridad.

Fuente: Díaz Martínez, J. M., & Muñoz Mansilla, R. (2008). En Nombre del editor (Ed.), FUNDAMENTOS DEL SISTEMA OPERATIVO UNIX (Capítulo 6: Planificación de los procesos en UNIX, pag. 266). Editorial UNED

(c) La utilización de RR dentro de cada cola: ¿Verdaderamente favorece al sistema de Tiempo Compartido? Justifique

Si se implementa de manera equilibrada y se ajusta en función de la carga del sistema y la naturaleza de los procesos en ejecución, el uso de Round-Robin dentro de cada cola en un sistema de tiempo compartido favorece una asignación equitativa de recursos y una respuesta rápida para los procesos más cortos. Sin embargo, es importante considerar el impacto en el rendimiento y la eficiencia del sistema al manejar procesos de larga duración y minimizar los posibles efectos negativos de los cambios frecuentes de contexto

20. A cuáles de los siguientes tipos de trabajos:
(a) cortos acotados por CPU
(b) cortos acotados por E/S
(c) largos acotados por CPU
(d) largos acotados por E/S

benefician las siguientes estrategias de administración:
(a) prioridad determinada estáticamente con el método del más corto primero (SJF):

Se benefician cortos acotados por CPU y cortos acotados por E/S. Ya que se da prioridad a los procesos cortos.

(b) prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S

Se benefician los largos acotados por E/S y cortos acotados por E/S. Ya que los trabajos que han pasado más tiempo desde la última E/S reciben una mayor prioridad.

21. Explicar porqué si el quantum "Q” en Round-Robin se incrementa sin límite, el método se aproxima a FIFO.

Si el quantum "Q" en el algoritmo Round-Robin se incrementa sin límite, es decir, si se le permite crecer indefinidamente, el comportamiento del algoritmo se aproxima al de FIFO (First In, First Out).
Esto se debe a que un quantum muy grande implicaría que cada proceso se ejecutara en su totalidad antes de pasar al siguiente, lo que es esencialmente la definición de FIFO.
En FIFO, el primer proceso que llega es el primero en ser atendido, y los procesos posteriores tienen que esperar a que los procesos anteriores se completen. Con un quantum infinitamente grande en Round-Robin, cada proceso se ejecutaría hasta su finalización antes de que el siguiente proceso en la cola pudiera recibir atención.

22. Los sistemas multiprocesador pueden clasificarse en:

- **Homogéneos: Los procesadores son iguales. Ningún procesador tiene ventaja física sobre el resto.**

- **Heterogéneos: Cada procesador tiene su propia cola y algoritmo de planificación.**

Otra clasificación posible puede ser:

- **Multiprocesador débilmente acoplados: Cada procesador tiene su propia memoria principal y canales.**

- **Procesadores especializados: Existe uno o más procesadores principales de propósito general y varios especializados controlados por el primero (ejemplo procesadores de E/S, procesadores Java, procesadores Criptográficos, etc.).**

- **Multiprocesador fuertemente acoplado: Consta de un conjunto de procesadores que comparten una memoria principal y se encuentran bajo el control de un Sistema Operativo**

(a) ¿Con cuál/es de estas clasificaciones asocia a las PCs de escritorio habituales?

Las PCs de escritorio habituales son sistemas multiprocesador homogeneos y fuertemente acoplados.

(b) ¿Qué significa que la asignación de procesos se realice de manera simétrica?

En un sistema multiprocesdor, que la asignación se realice de forma simetrica significa que cada procesador se planifica a sí mismo. Donde cada procesador puede tener su propia cola, o bien, pueden compartir las colas de listo entre sí. Si comparten, puede ser un problema que todos accedan a la misma cola y hay que controlar que un proceso no entre más de un procesador.

(c) ¿Qué significa que se trabaje bajo un esquema Maestro/esclavo?

En un sistema multiprocesdor, que se trabaje bajo un esquema Maestro/esclavo hace referencia a la asignación de procesos de manera asimetrica. En este esquema un procesador toma la decisiones y planifica a los demás, el resto se limita a ejecutar los procesos.

23. Asumiendo el caso de procesadores homogéneos:

(a) ¿Cuál sería el método de planificación más sencillo para asignar CPUs a los procesos?

La compartición de carga es un enfoque de asignación de procesos en sistemas multiprocesador en el cual los procesos no se asignan a un procesador específico. En cambio, se mantiene una cola global de hilos listos, y cada procesador, cuando está ocioso, selecciona un hilo de esa cola para ejecutarlo

(b) Cite ventajas y desventajas del método escogido

Ventajas:

- Distribución uniforme de la carga: La carga se distribuye de manera uniforme entre los procesadores, evitando que un procesador quede ocioso mientras haya trabajo pendiente.
- No requiere un planificador centralizado: No se necesita un planificador centralizado para asignar trabajos a procesadores. Cuando un procesador queda disponible, la rutina de planificación del sistema operativo se ejecuta en ese procesador para seleccionar el siguiente hilo de la cola global.
- Flexibilidad en la organización de la cola: La cola global puede organizarse y ser accesible utilizando diferentes esquemas de planificación, como FCFS, prioridades, o esquemas basados en la historia de ejecución.

Desventajas:

- Posible cuello de botella: La cola central puede convertirse en un cuello de botella si muchos procesadores buscan trabajo al mismo tiempo, especialmente en sistemas con numerosos procesadores.
- Efecto en la eficacia de la caché: Si los hilos expulsados no retoman su ejecución en el mismo procesador, las caches locales pueden volverse menos efectivas, afectando el rendimiento.
- Problemas de coordinación: Si todos los hilos se tratan como un conjunto común y no hay coordinación adecuada, los cambios de procesos necesarios pueden comprometer el rendimiento, especialmente en programas que requieren un alto grado de coordinación entre los hilos

24. Indique brevemente a que hacen referencia los siguientes conceptos:

(a) Huella de un proceso en un procesador

La huella de un proceso en un procesador se refiere a la cantidad de recursos que un proceso utiliza en el procesador. Estos recursos pueden incluir tiempo de CPU, memoria, ancho de banda de E/S, etc

(b) Afinidad con un procesador

La afinidad con un procesador se refiere a la capacidad de asignar prioridad o preferencia a un proceso o subproceso para ejecutar en una CPU.

Esto permite que el proceso o subproceso se ejecute solo en la CPU o núcleos designados, lo que puede mejorar el rendimiento o la eficiencia.

(c) ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador?

En algunos casos, puede ser mejor que un proceso se ejecute en el mismo procesador para minimizar la latencia y maximizar la velocidad de procesamiento.

Por ejemplo, si un proceso necesita acceder a datos almacenados en la memoria caché de un procesador, es más rápido si se ejecuta en el mismo procesador que si se ejecuta en otro procesador.

(d) ¿Puede el usuario en Windows cambiar la afinidad de un proceso? ¿y en GNU/Linux?

Sí, el usuario puede cambiar la afinidad de un proceso en Windows y GNU/Linux. En Windows, el usuario puede usar el Administrador de tareas o la línea de comandos para establecer la afinidad del procesador. En GNU/Linux, el usuario puede usar el comando taskset para consultar y/o fijar la afinidad de determinados procesos y/o hilos.

(e) Investigue el concepto de balanceo de carga (load balancing).

El balanceo de carga es una técnica que se utiliza para distribuir la carga de trabajo entre varios servidores, dispositivos, componentes de una red, y en el caso de los sistemas operativos, entre CPUs. Su objetivo es evitar la sobrecarga y el colapso del sistema. El balanceador de carga (SO) se encarga de distribuir los procesos de los usuarios en todos los procesadores para satisfacer esas solicitudes de manera que maximice la velocidad y la capacidad para poder garantizar que ningún procesador esté sobrecargado mientras otro está ocioso.

(f) Compare los conceptos de afinidad y balanceo de carga y como uno afecta al otro.

El balanceo de carga puede afectar a la afinidad por ejemplo en un sistema con balanceo de carga dinámico, donde los procesos pueden ser movidos entre procesadores para equilibrar la carga. Este movimiento puede influir en la afinidad, ya que un proceso que inicialmente tenía afinidad con un procesador específico podría ser movido a otro.

Y la afinidad puede afectar al balance de carga, por ejemplo en el caso que varios procesos tienen afinidad para ejecutarse en el mismo procesador, puede haber una concentración de carga en ese procesador específico. Esto podría resultar en un desequilibrio en la carga de trabajo entre los procesadores, ya que uno puede estar más ocupado que los demás.