1. Responda en forma sintética sobre los siguientes conceptos:

(a) Programa y Proceso

Programa

- Es estático
- No tiene program cunter
- Existe desde que se edita hasta que se borra

Proceso

- Es dinámico
- Tiene program counter
- Su ciclo de vida comprende desde que se lo ejecuta hasta que termina

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

- Retorno: tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución
- Espera: tiempo que el proceso se encuentra en el sistema esperando, es decir el tiempo que pasa sin ejecutarse. Es básicamente la diferencia entre el tiempo de retorno y el tiempo de ocupación del procesador

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

TPR: es el promedio de tiempos de retorno entre todos los procesos pertenecientes al lote. Se calcula como la suma del TR de cada proceso, dividida entre la cantidad de procesos.

TPE: es el promedio de tiempos de espera calculado entre todos los procesos involucrados en el lote. Se calcula como la suma del TE de todos los procesos, dividida entre la cantidad de procesos.

(d) ¿Qué es el Quantum?

Quantum (Q): medida que determina cuanto tiempo podrá usar el procesador cada proceso

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

- Nonpreemptive: una vez que un proceso está en estado de ejecución, continua hasta que termina o se bloquea por algún evento (e.j. I/O)
- Preemptive: el proceso en ejecución puede ser interrumpido y llevado a la cola de listos:
    - Mayor overhead pero mejor servicio
    - Un proceso no monopoliza el procesador

(f) ¿Qué tareas realizan?:

i. Short Term Scheduler: determina que proceso pasar a ejecutarse. Es el que selecciona, de entre los procesos en estado ready, aquel que va a pasar a ser ejecutado.

ii. Long Term Scheduler: admite nuevos procesos a memoria (controla el grado de multiprogramación). Se encarga del paso de estado nuevo a listo

iii. Medium Term Scheduler: realiza el swapping (intercambio) entre el disco y la memoria cuando el SO lo determina (puede disminuir el grado de multiprogramación).

(g) ¿Qué tareas realiza el Dispatcher?

Hace cambios de contexto, cambio del modo de ejecución, despacha el proceso elegido en Short Term → salta a la instrucción a ejecutar.

Descarga los registros del procesador y otros datos relevantes a las estructuras de datos que se utilizan para controlar el proceso (PCB) saliente, y la carga en el procesador y otros puntos de los mismos datos del proceso entrante.

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: muestra en tiempo real información del sistema. (Por defecto cada 3 segundos)

$ top [-d SEGUNDOS] [-u USUARIO]

ii. htop: con htop podremos obtener un resultado más amigable, no esta instalado por defecto se ha de instalar mediante $ apt-get install htop ... una vez instalado ...

$ htop

iii. ps: muestra información de los procesos activos.

ps [OPCIONES]

Por defecto muestra:

- **PID** : Id del Proceso.
- **TTY** : Terminal.
- **TIME** : Tiempo de ejecución.
- **CMD** : Comando.

Opciones

- **a** : Muestra todos los procesos asociados a un TTY.
- **-e** / **-A** : Muestra todos los procesos.
- **x** : Muestra los no asociados.
- -**f** : Muestra el formato largo:
    - **UID** : Usuario que lo ejecutó.
    - **PPID** : Id del proceso padre.
    - **C** : Uso del procesador.
    - **STIME** : Inicio de ejecución.
- **u** : Orientado al usuario:
    - **USER**
    - **% CPU** : uso de procesador.

- **% MEM** : uso de memoria.
- **VSZ** : Memoria virtual.
- **RSS** : Memoria física.
- **STAT** : Estado.
- **START** : Iniciado.

Significados de los estados STAT :

- **S** : Esperando (Sleep).
- **R** : Ejecutando (Running)
- **D** : Esperando entrada/salida.
- **T** : Pausa.
- **Z** : No responde (Zombie)

Información adicional de los STAT:

- **s** : Proceso padre
- **l** : Proceso con hilos.
- **+** : En primer plano.

iv. pstree: muestra en forma de árbol los procesos en ejecución.

Con el parámetro "-a", nos muestra la línea de comandos utilizada. Por ejemplo, si el comando utiliza la ruta a un fichero de configuración.

Por defecto se inhabilita la visualización en el árbol de los nombres repetidos. Para evitar esto, podemos utilizar el parámetro "-c"

Si nos interesa podemos ver el árbol de un proceso específico, de la siguiente manera:

pstree -H [PID]

v. kill: es un comando unix/linux que permite enviar señales a uno o varios procesos del sistema a través de la shell (bash, ksh, etc). Las señales más utilizadas suelen ser la de matar un proceso (9 ó SIGKILL), pararlo (TERM) o reiniciarlo (1 ó HUP) pero hay muchas más que pueden ser útiles en ocasiones. El listado completo de señales disponibles puede visualizarse ejecutando kill -l

vi. pgrep: busca todos los procesos actualmente en el sistema cuyas características coincidan exactamente con las indicadas en la línea de comandos, y lista su PID en pantalla.

La sintaxis es pgrep [-flvx] [-d delimiter] [-n|-o] [-P ppid,...] [-g pgrp,...] [-ssid,...] [-u euid,...] [-U uid,...] [-G gid,...] [-t term,...] [pattern]

-d delimitador: establece la cadena que se utilizará para delimitar cada PID en la salida, por defecto es un salto de línea (exclusivo de pgrep).

-f: el patrón pattern normalmente sólo se compara con el nombre del proceso, si se establece -f se comparará con la línea de comandos completa.

-g pgrp,...: sólo tomará procesos que se correspondan con los PGIDs indicados. El PGID 0 es interpretado como el PGID bajo el que se ejecuta el propio comando.

-G gid,...: sólo tomará procesos cuyo ID real de grupo coincida con uno de los indicados. Se puedenutilizar valores numéricos o simbólicos.

-l: lista el nombre del proceso además de su PID. (exclusivo de pgrep).

-n: sólo selecciona el más nuevo de los procesos encontrados (el que haya iniciado más recientemente).

-o: el opuesto a -n, selecciona al más antiguo de los procesos encontrados.

-P ppid,...: sólo selecciona procesos cuyo PPID (parent process ID) coincida con uno de los indicados.

-s sid,...: sólo selecciona aquellos procesos cuyo ID de sesión de proceso coincide con uno de los indicados. El ID de sesión 0 es interpretado como aquel bajo el que se ejecuta el propio comando.

-t term,...: sólo selecciona procesos cuya terminal de control se encuentra entre las indicadas. El nombre de la terminal debe ser especificado con el prefijo "/dev/".

-u euid,...: sólo selecciona procesos cuyo ID de usuario efectivo coincide con el indicado. Pueden utilizarse tanto valores simbólicos como numéricos.

-U uid,...: sólo selecciona procesos cuyo ID de usuario real coincide con el indicado. Pueden utilizarse tanto valores simbólicos como numéricos.

-v: realiza la negación de la comparación (selecciona los que no respetan el/los patrones)

-x: sólo selecciona aquellos procesos cuyo nombre coincide exactamente con el patrón indicado (o su línea de comando de origen si se selecciona la opción -f)

pkill: busca todos los procesos cuyas características coincidan exactamente con las indicadas en la línea de comandos, y les envía el mensaje indicado en el llamado al comando. Si no se indica una señal, se enviará la predeterminada SIGTERM.

La sintaxis es pgrep [-signal] [-fvx] [-n|-o] [-P ppid,...] [-g pgrp,...] [-s sid,...] [-u euid,...] [-U uid,...] [-G gid,...] [-t term,...] [pattern]

Comparte las opciones con el comando pgrep, con la excepción de:

-signal: define la señal a ser enviada a cada uno de los procesos que coincidan con los patrones indicados. Pueden utilizarse indistintamente valores simbólicos o numéricos de señal. (y aquellas indicadas en pgrep como exclusivas de dicho comando)

vii. killall: killall: mata todos los procesos con un nombre concreto o de un usuario o ambos.

$ killall [-u USUARIO] [-SENAL] [NOMBRE_PROCESO]

Ejemplos de uso:

Matar todos los procesos con el nombre de "sleep"

$ killall sleep

Matar todos los procesos del usuario "Pepito"

$ killall -u Pepito

Matar todos los procesos del usuario "Pepito" que se llamen "sleep"

$ killall -u Pepito sleep

viii. renice: cambia la prioridad de un proceso ejecutándose. No se puede aumentar la urgencia.

$ renice -n NUMERO_PRIORIDAD COMANDO

ix. xkill: fuerza la desconexión de un X server con uno de sus clientes (básicamente permite cerrar ventanas de programas). Forzar la desconexión no implica necesariamente que los procesos asociados al cliente terminarán de forma correcta, o siquiera que serán terminados.

La sintaxis es xkill [-display displayname] [-id resource] [-button number] [-frame] [-all]

Si no se indica ninguna de las opciones, se mostrará un cursor en forma de x (o una calavera, según la distribución y conjunto de símbolos del sistema) con el que se podrá seleccionar el cliente (ventana) a cerrar.

x. atop: es un monitor interactivo de la carga de trabajo en un sistema operativo Linux. Indica la ocupación de los recursos más críticos de hardware (desde una perspectiva de rendimiento) a nivel de sistema (i.e. CPU, memoria, disco y red). También muestra qué procesos son responsables de la carga indicada de cpu y memoria. La carga de disco y red dependen de la presencia de configuraciones y módulos del kernel.

Cada un determinado intervalo (predeterminado: 10 segundos, pero se puede modificar en la línea de comandos) se muestra información sobre la ocupación de recursos del sistema, seguida por una lista de procesos que han estado activos durante el último intervalo (si algún proceso no fue modificado durante este intervalo, no será mostrado, salvo que se haya presionado la tecla ‘a’). Si la lista de procesos activos no entra en la pantalla, se mostrará todos los que entren, ordenados en función de su actividad.

Al igual que top, los intervalos se repiten hasta alcanzar el número de muestras indicadas en el comando de llamada al programa, o hasta que se presione la tecla 'q' en caso de estar en modo interactivo.

Cuando se inicia, el programa evalúa si la salida estándar está conectada con una pantalla con un archivo o pipe. En el primer caso produce códigos de control de pantalla, y se comporta de forma interactiva; en el segundo caso produce una salida de texto plano en formato ASCII. En modo interactivo, la salida de atop es escalada de forma dinámica en función de las dimensiones actuales de la ventana o pantalla. Si se modifica el ancho de la pantalla o ventana, se agregarán o quitarán columnas de manera automática, mostrando aquellas más importantes que entren en la dimensión actual (para este fin, cada columna tiene asignado un valor de relevancia)

(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?.

4, una por el padre, 3 por los hijos.

ii. ¿El número de líneas es el número de procesos que han estado en ejecución?. Ejecute el programa y compruebe si su respuesta es correcta, Modifique el valor del bucle for y compruebe los nuevos resultados.

Si

(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 ;

printf ( " Comienzo . : \ n " ) ;

for ( c = 0 ; c < 3 ; c++ )

{

pid = fork ( ) ;

}
```

p++

printf (" Proceso %d\n " , p) ;

return 0 ;

}

i. ¿Qué valores se muestran por consola?.

Supongo que 0,1,2,3

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

Valores distintos

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

(d) Comunicación entre procesos:

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

El "|" nos permite comunicar dos procesos por medio de un pipe desde la shell

El pipe conecta stdout (salida estandar) del primer comando con la stdin (entrada estandar) del segundo.

- Por ejemplo:
  - $ ls | more
    - Se ejecuta el comando ls y la salida del mismo, es enviada como entrada del comanda more

Se pueden anidar tantos pipes como se deseen

ii. ¿Cómo se crea un pipe en C?.

A través de la instrucción pipe(&fd[N]) con N = 0 o N = 1 y fd un arreglo de dos enteros

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

Se debe establecer como parámetro un arreglo de dos enteros, indicando el primer dato ([0]) si el proceso recibirá datos a través del canal (es decir que se convierte en el descriptor de archivo de su entrada estándar), y el segundo dato si utilizará el pipe para escribir (convirtiéndolo en el descriptor de archivo de su salida estándar).

Para lograr una comunicación correcta, los procesos involucrados deberán cerrar el extremo del canal que no les incube (close(&fd[0]) si el pipe fue abierto con pipe(&fd[1]) y viceversa. De otra forma, el pipe nunca se dará por cerrado, puesto que no aparecerá una marca de EOF.

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

Los canales o tuberías sólo permiten comunicación de una sóla vía, es decir que un proceso sólo puede escribir o leer un pipe, pero no ambas a la vez.

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

- Es una estructura de datos asociada al proceso.
- Existe una PCB por proceso.
- Tiene referencias a memoria.
- Es lo primero que se crea cuando se crea un proceso y lo último que se borra cuando termina.
- El PCB lo podemos pensar como un gran registro en el que se guardan los atributos anteriormente mencionados, también guarda punteros y direcciones de memoría relacionadas al proceso.
    - Información asociada a cada proceso:
        - PID, PPID, etc.
        - Valores de los registros de la CPU (PC, AC, etc).
        - Planificación (estado, prioridad y tiempo consumido del proceso, etc).
        - Ubicación (donde está el proceso) en memoria.
        - Accounting (cantidades, cuanta memoria ocupó, cuanta entrada salida ocupó).

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

CPU Bound: significa que el proceso requiere principalmente de tiempo de procesador para su ejecución

I/O Bound: significa que el proceso depende en gran medida de operaciones de E/S para su ejecución

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

NEW:

- Estado inicialización.
- Un usuario dispara el proceso. Un proceso se crea por otro proceso: su proceso padre.
- En new se crean las estructuras asociadas al proceso, y el proceso queda en la cola de procesos, esperando ser cargado en memoría.

READY:

- Cuando el "Scheduler Long Term" elige al proceso para cargarlo en memoría, dicho proceso pasa al estado Ready.
- Esta en memoria pero no se esta ejecutando.
- Los procesos compiten para obtener la CPU.
- El proceso ahora solo necesita que se le asigne CPU.
- En cola de procesos listos (ready queue).
- Se selecciona y pasa a running.

RUNNING:

- El "Scheduler Short Term" eligió al proceso para asignarle CPU.
- Hay cambio de contexto.

- Dicho proceso tendrá la CPU a su disposición hasta que se termine el período de tiempo asignado (quantum o time slice), termine el proceso, o hasta que necesite una operación de E/S.

WAITING:

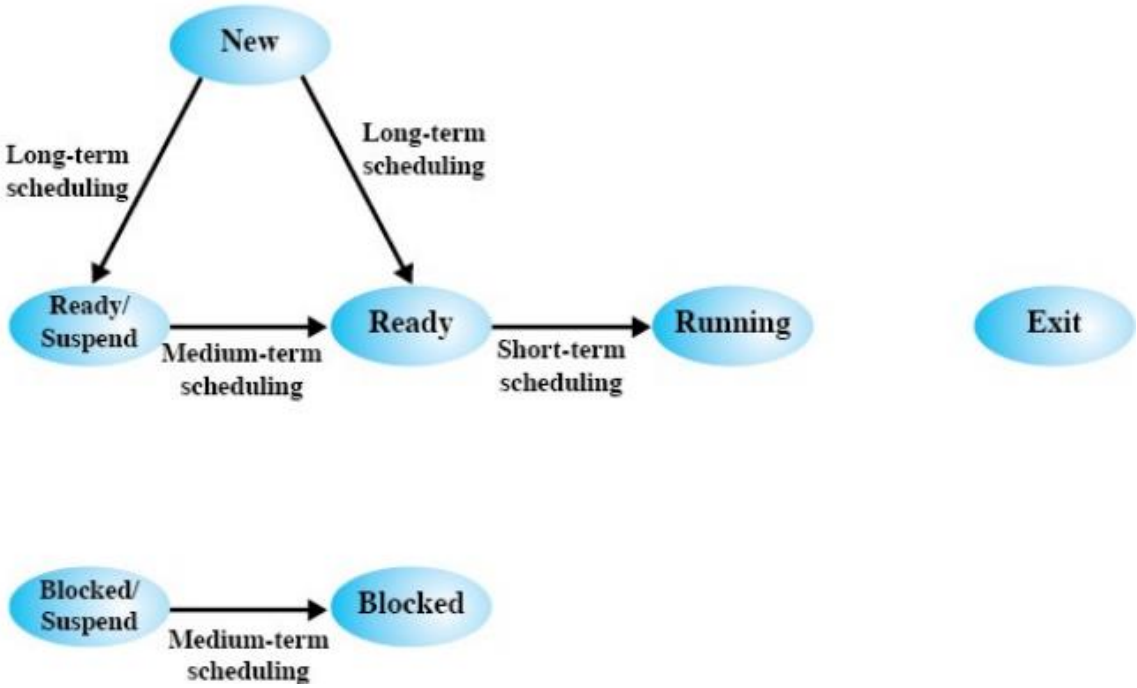
- El proceso necesita que se cumpla un evento esperado para seguir su ejecución.
- Dicho evento puede ser:
  - Terminación de una E/S requerida.
  - Llegada de una señal por parte de otro proceso.

TERMINATED:

- Eliminación de memoria principal de todas las estructuras asociadas.
- Finaliza con el borrador de la PCB.

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

(i) ¿Que scheduler de los mencionados en 1 f 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.

JOB	LLEGADA	CPU	PRIORIDAD
1	0	4	3
2	1	4	2
3	2	4	1
4	3	4	2

FCFS

- Cuando hay que elegir un proceso para ejecutar, se selecciona el más viejo
- No favorece a ningún tipo de procesos, pero en principio podíamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no

Orden → 1, 2, 3 , 4

SJF

- Política nonpreemptive que selecciona el proceso con la ráfaga más corto
- Calculo basado en la ejecución previa
- Procesos cortos se colocan delante de procesos largos
- Los procesos largos pueden sufrir starvation (inanición)

Orden → 1, 3, 2, 4

Round Robin

- Politica basada en un reloj
- Quantum (Q): medida que determina cuanto tiempo podrá usar el procesador cada proceso:
    - Pequeño: overhead de context switch
    - Grande: es igual que FCFS
- Cuando un proceso es expulsado de la CPU es colocado al final de la Ready Queue y se selecciona otro (FIFO circular)

Orden suponiendo Q=4 → 1, 2, 3, 4

PID	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	>			<												
2		>						<								
3			>									<				
4				>												<

Prioridades

- Cada proceso tiene un valor que representa su prioridad → menor valor, mayor prioridad
- Se selecciona el proceso de mayor prioridad de los que se encuentran en la Ready Queue

- Existe una Ready Queue por cada nivel de prioridad
- Procesos de baja prioridad pueden sufrir starvation (inanición)
- Solución: permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty
- Puede ser un algoritmo preemptive o no

Orden (no apropiativo) → 1, 3, 2, 4

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

Round Robin requiere de la definición de un quantum

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

FCFS → No favorece a ningún tipo de procesos, pero en principio podíamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no.

SJF → I/O Bound porque los CPU Bound al ser (casi siempre) los mas largos, podrian sufrir inanición.

Round Robin → Mas adecuado para I/O Bound.

Prioridades → No favorece a ningún tipo de procesos.

(d) Cite ventajas y desventajas de su uso.
Mas arriba esta

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

Timer Fijo

Timer Variable

(a) ¿Qué significan estas 2 variantes?

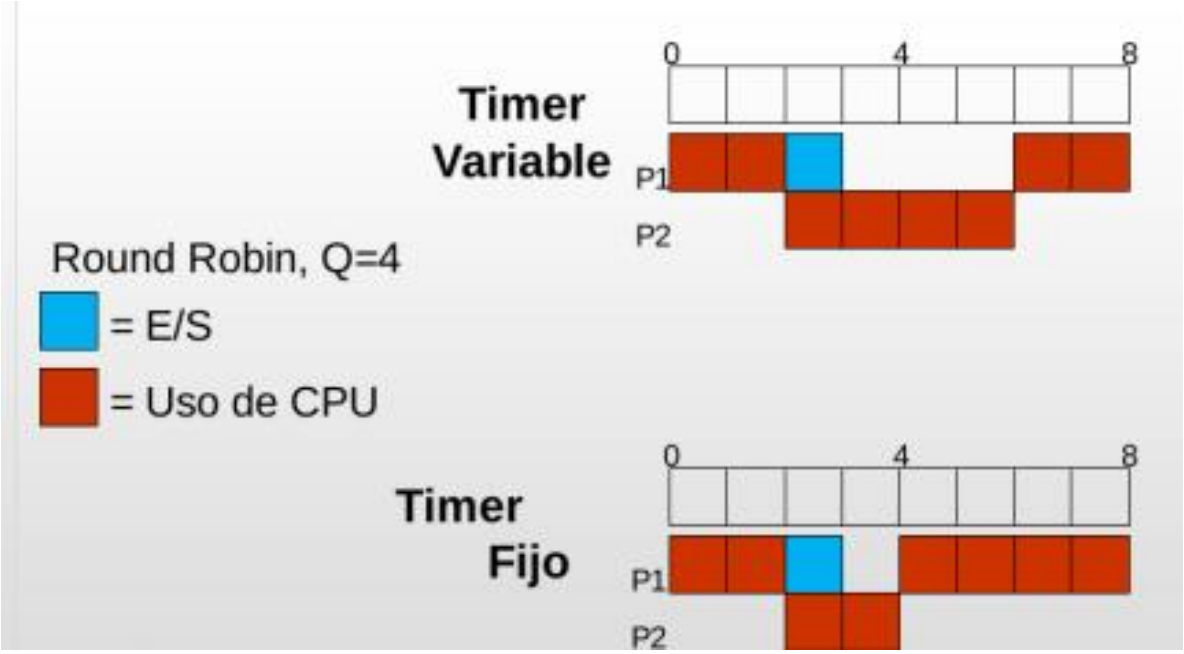
Timer Variable : El “contador” se inicializa en Q (contador := Q) cada vez que un proceso es asignado a la CPU

- Es el más utilizado
- Utilizado por el simulador

Timer Fijo : El “contador” se inicializa en Q cuando su valor es cero

- if (contador == 0) contador = Q;
- Se puede ver como un valor de Q compartido entre los procesos

(b) Explique mediante un ejemplo sus diferencias.



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

En el caso de Timer Variable, cada proceso debe almacenar la información de su Quantum en su PCB, mientras que en el caso de Timer Fijo, se debe almacenar en una estructura o espacio accesible directamente por el SO

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

(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

> marca llegada

(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 tienen promedios mas pequeños, pero hay demasiada diferencia en los tiempos de cada proceso.

6. Se tiene el siguiente lote de procesos: (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.

Los algoritmos no apropiativos tienen promedios más pequeños, pero hay demasiada diferencia en los tiempos de cada proceso. SJF mas eficiente en promedio. Menos eficiente el de Q=1

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

Poco Quantum → mucho cambio de contexto y promedios mas grandes.

Mucho Quantum → se hace mas similar a FCFS y por lo tanto, los promedios son menores.

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

Utilizaria Quantum para los procesos que requieren de bastante tiempo de CPU para que puedan terminar su tarea de manera rápida con un tiempo de retorno bajo.

La desventaja es que si hay procesos con pocas unidades de CPU, estarían esperando varios ciclos para usar la CPU por poco tiempo y darse por terminados.

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?
Tiene mucho mejor promedio, sobre todo con respecto al Tiempo de Espera.

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

(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?

El no apropiativo es exactamente igual al SJF. En cuanto a los promedios el apropiativo tiene promedios mas grandes. Utilizaría la prioridad cuando se trata de algún proceso relacionado a la administración realizada por el SO que se tiene que actuar de manera inmediata.

9. Inanición (Starvation)

(a) ¿Qué significa?

La inanición es un problema relacionado con los sistemas multitarea, donde a un proceso o un hilo de ejecución se le deniega siempre el acceso a un recurso compartido (en este caso la CPU). Sin este recurso, la tarea a ejecutar no puede ser nunca finalizada.

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

Cualquier algoritmo que involucre algún tipo de diferenciación entre procesos y asigne recursos con base en ese criterio es susceptible de generar inanición, traducido a tipos de algoritmos, implica que los tipos SJF, SRTF y por prioridades, todos pueden generar inanición; los primeros dos en aquellos procesos que sean más largos que, y el tercero en aquellos procesos de menor prioridad

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

Si, la solución es permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty

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 mas).

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 administrada 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:

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

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

Beneficia a los procesos ligados a CPU. Los procesos E/S podrían, en un sólo quantum utilizar sólo una parte del mismo antes de bloquearse para esperar un proceso de E/S, y no volverán a la cola de listos hasta que dicha operación termine. Los procesos CPU bound, por otro lado, sólo abandonan el CPU por haberse acabado su quantum (o el propio proceso), y ni bien esto sucede retornan a la cola de listos, requiriendo esperar menos tiempo por el uso del procesador.

Un proceso I/O bound usara poco tiempo de CPU antes de esperar a que se complete la E/S y round robin lo pondrá al final de la cola incluso si usó poco tiempo. Un proceso CPU Bound va a usar la CPU aprovechando todo el Quantum.

(b) SRTF (Shortest Remaining Time First)

Favorece a los procesos ligados a E/S, ya que estos por principio hacen menos uso de CPU, con lo que desde el momento en que llegan a la cola de listos tienen más posibilidades de acceder al procesador. Por otro lado, al requerir ráfagas cortas de CPU, cuentan con más chances de hacer uso de la totalidad de su ráfaga y esperar al nuevo proceso de E/S (o terminarse) antes de ser expulsados.

https://lass.cs.umass.edu/~shenoy/courses/fall12/lectures/notes/Lec05_notes.pdf

https://people.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html#:~:text=A%20big%20advantage%20of%20round,one%20a%20chance%20to%20run.

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 v 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 en caso de que el proceso se complete de manera total en una ráfaga antes de que se le termine el Quantum dado.

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 formula:

$$S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n \qquad (1)$$

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 fórmula 1, con un valor inicial estimado de S1=10.

Ráfaga (n)	1	2	3	4	5	6	7	8
Estimación (Sn)	10	6	5	5.3	5	6.6	7.6	8.4
Tiempo real (Tn)	6	4	6	4	13	13	13	

$\frac{1}{1}6 + \frac{1-1}{1}10 = 6$

$\frac{1}{2}4 + \frac{2-1}{2}6 = 5$

$\frac{1}{3}6 + \frac{3-1}{3}5 = 5.3$

$\frac{1}{4}4 + \frac{4-1}{4}5.3 = 5$

$\frac{1}{5}13 + \frac{5-1}{5}5 = 6.6$

$\frac{1}{6}13 + \frac{6-1}{6}6.6 = 7.6$

$\frac{1}{7}13 + \frac{7-1}{7}7.6 = 8.4$

La fórmula 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 más recientes y menor a casos viejos (o viceversa). Se plantea la siguiente formula:

$$S_{n+1} = \alpha T_n + (1 - \alpha) S_n \qquad (2)$$

Con $0 \langle \alpha \langle 1$.

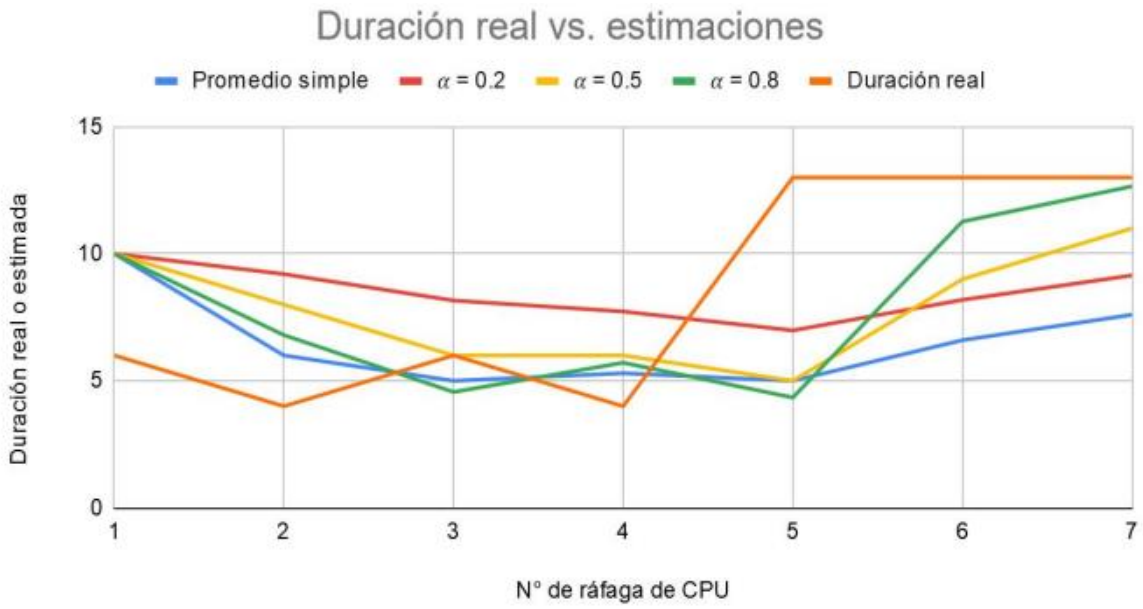
(b) Analice para que valores de α se tienen en cuenta los casos más recientes.

Para valores de $\alpha \rightarrow 1$. En ese caso tiene más peso el primer término de la ecuación, que representa el tiempo más reciente

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

Ráfaga (n)	**α**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Estimación (Sn)	**0.2**	10	9.2	8.16	7.72	6.976	8.19	9.15	9.92
	0.5	10	8	6	6	5	9	11	12
	0.8	10	6.8	4.56	5.71	4.34	11.27	12.65	12.93
Tiempo real (Tn)		6	4	6	4	13	13	13	

(d) Para todas las estimaciones realizadas en a y c ¿Cuál es la que más 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?.

PROCESOS INTERACTIVOS → ROUND ROBIN

BATCH → FCFS

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.

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

PRIORIDADES Con AGING

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 está 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.

Mediante la planificación con colas multinivel realimentadas, un proceso se puede mover de una cola a otra dependiendo de su comportamiento en tiempo de ejecución.

En las colas multinivel realimentadas se separan los procesos en grupos pero dependiendo de las características de su ráfaga de CPU. Los procesos con ráfagas cortas irán a una cola más prioritaria de procesos preparados que los procesos con ráfagas largas.

El funcionamiento de este algoritmo consiste en ejecutar los procesos de la cola de prioridad más alta, a continuación se pasan a ejecutar los procesos de la siguiente cola y así sucesivamente. Con esta distribución, los procesos con ráfagas cortas se ejecutarán de forma rápida sin necesidad de llegar muy lejos en la jerarquía de colas de listos. Mientras que los procesos con ráfagas largas irán degradándose gradualmente.

Para gestionar a los procesos de la forma más justa, es necesario conocer su longitud, si están limitados por entrada/salida o por el procesador, la memoria que van a necesitar, etc.

La forma óptima de atenderlos es:

- Establecer una preferencia para los trabajos más cortos y penalizar a los que se han estado ejecutando durante más tiempo.
- Favorecer a los trabajos limitados por entrada/salida, para mantener los recursos ocupados y dejar el procesador libre el mayor tiempo posible.

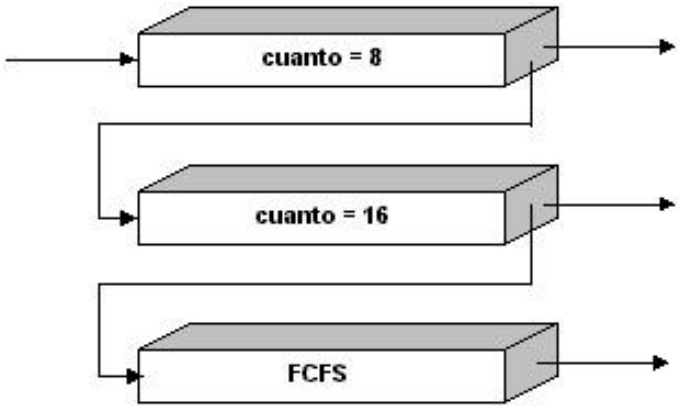
En general, a un proceso se le concede un tiempo T de permanencia en una cola, cuando lo supera, pasará a la cola inmediatamente inferior con menor prioridad, es decir, se disminuirá su prioridad en una unidad. La elección del valor que se le dará al tiempo T varía mucho de un sistema a otro, depende del número de procesos existentes, del tipo de procesos y del número de colas.

Se pueden usar mecanismos de envejecimiento para evitar el bloqueo indefinido de un proceso, estos mecanismos consisten en incrementar la prioridad de los procesos que estén demasiado tiempo esperando en una cola de prioridad baja, para pasarlos a una cola de prioridad más alta y que se puedan ejecutar antes.

En resumen, este algoritmo se puede definir por los siguientes parámetros:

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

Mayor prioridad → menos Quantum. Beneficia a E/S ya que los CPU Bound pueden llegar a sufrir inanición si hay muchos de E/S → Tratarlo con aging



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.

$Nice_i$ = 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?

Las actividades que conllevan poco uso de la CPU, las que son inherentes al procesador y requieren una interacción. Priorizar estas tareas permite que aprovechen al máximo el poco uso de CPU que puedan necesitar y poder minimizar el tiempo de espera.

Si quiero escribir este trabajo en Word, no quiero que la tecla que presiono me aparezca 10 segundos después. Los que son CPU Bound, son más invisibles al 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

Se favorecerían los de E/S, ya que el usuario utilizaría mas a esos debido a la interacción.

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

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).

(a) cortos acotados por CPU → Beneficiado

(b) cortos acotados por E/S → Beneficiado

(c) largos acotados por CPU → Penalizado si aparecen constantemente más cortos

(d) largos acotados por E/S → Penalizados si aparecen constantemente más cortos + un plus por esperar la E/S

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

(no lo entendí mucho la verdad, supongo que los cpu al no tener e/s no se benefician?)

(a) cortos acotados por CPU → penalizados

(b) cortos acotados por E/S → beneficiados

(c) largos acotados por CPU → penalizados

(d) largos acotados por E/S → beneficiados

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

Porque al ser el Quantum tan largo, va a llegar un punto donde los procesos saldrán por su propia cuenta (finalizaron o por E/S), antes de ser expulsado. De esta forma se eliminaría por completo la rotación de procesos, ya que cada proceso encolado acabaría por sí mismo sin regresar a la cola de ejecución a esperar su siguiente turno (porque no lo necesitaría, salvo que se abandone el procesador por E/S).

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?

Homogéneos, fuertemente acoplados

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

En un multiprocesador simétrico, el núcleo puede ejecutar en cualquier procesador, y normalmente cada procesador realiza su propia planificación del conjunto disponible de procesos e hilos. El núcleo puede construirse como múltiples procesos o múltiples hilos, permitiéndose la ejecución de partes del núcleo en paralelo. Éste complica al sistema operativo, ya que debe asegurar que dos procesadores no seleccionan un mismo proceso y que no se pierde ningún proceso de la cola. Se deben emplear técnicas para resolver y sincronizar el uso de los recursos.

En el multiprocesamiento simétrico, los procesadores comparten la misma memoria.

	MULTIPROCESAMIENTO SIMÉTRICO	MULTIPROCESAMIENTO ASIMÉTRICO
Trabajo	Cada procesador ejecuta las tareas en el sistema operativo.	Solo el procesador maestro ejecuta las tareas del sistema operativo.
Proceso	El procesador toma los procesos de una cola preparada común, o puede haber una cola preparada privada para cada procesador.	Procesador maestro asigna procesos a los procesadores esclavos, o tienen algunos procesos predefinidos.
Arquitectura	Todo el procesador en multiprocesamiento simétrico tiene la misma arquitectura.	Todos los procesadores en multiprocesamiento asimétrico pueden tener la misma o diferente arquitectura.
Comunicación	Todos los procesadores se comunican con otro procesador mediante una memoria compartida.	Los procesadores no necesitan comunicarse ya que están controlados por el procesador maestro.
Al fallar	Si un procesador falla, la capacidad de computación del sistema se reduce.	Si falla un procesador maestro, se envía un esclavo al procesador maestro para continuar la ejecución. Si un procesador esclavo falla, su tarea cambia a otros procesadores.
Dificultad	El multiprocesador simétrico es complejo ya que todos los procesadores deben sincronizarse para mantener el equilibrio de carga.	El multiprocesador asimétrico es simple ya que el procesador maestro accede a la estructura de datos.

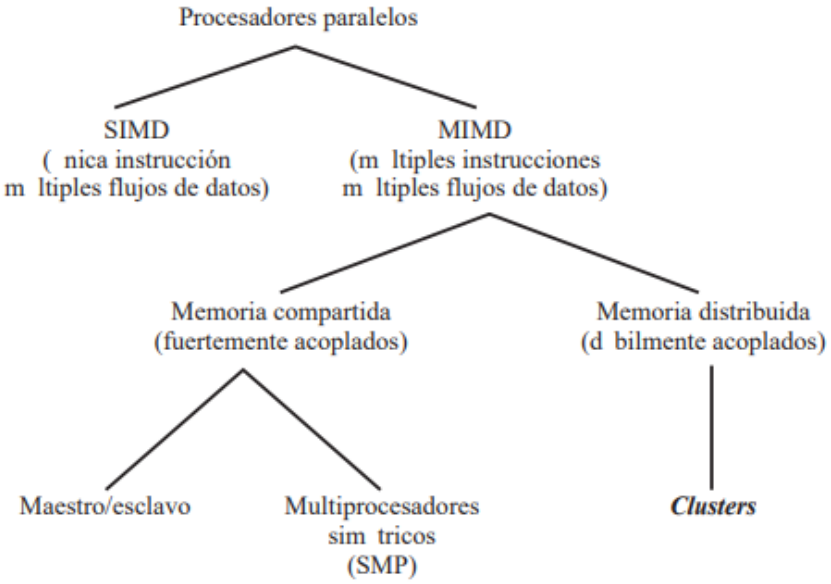


Figura 4.8. Arquitectura de procesadores paralelos.

(malditas tildes)

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

Maestro-esclavo (Master-slave en inglés) es un modelo de protocolo de comunicación en el que un dispositivo o proceso (conocido como maestro o master) controla uno o más de otros dispositivos o procesos (conocidos como esclavos o slaves). En este tipo de comunicación, el dispositivo maestro es siempre quién tiene toda iniciativa de comunicación en detrimento de los dispositivos considerados esclavos que lo único que pueden hacer es esperar una petición del maestro para poder responder a éste.

Para ser más precisos, el dispositivo maestro administra la red y actúa de mediador, de forma que pregunta periódicamente a cada dispositivo esclavo para ver si necesita realizar alguna acción determinada. El dispositivo esclavo podrá responder con una petición al maestro si necesita algo.

Ningún modelo es perfecto y éste tampoco lo es, una desventaja es que es imprescindible tener un dispositivo en la red que actúe como maestro y si por alguna razón su funcionamiento fuera erróneo, toda la red quedaría paralizada. Además es lento pues tiene que ir preguntando a todos los dispositivos si tienen algo que hacer y luego esperar una respuesta de cada uno de ellos durante un tiempo predeterminado.

Y aún en el caso de que uno de los dispositivos esclavos quisiera realizar alguna acción, puede que necesitara algo de algún otro dispositivo, lo que aumenta el tiempo de respuesta, como en este ejemplo:

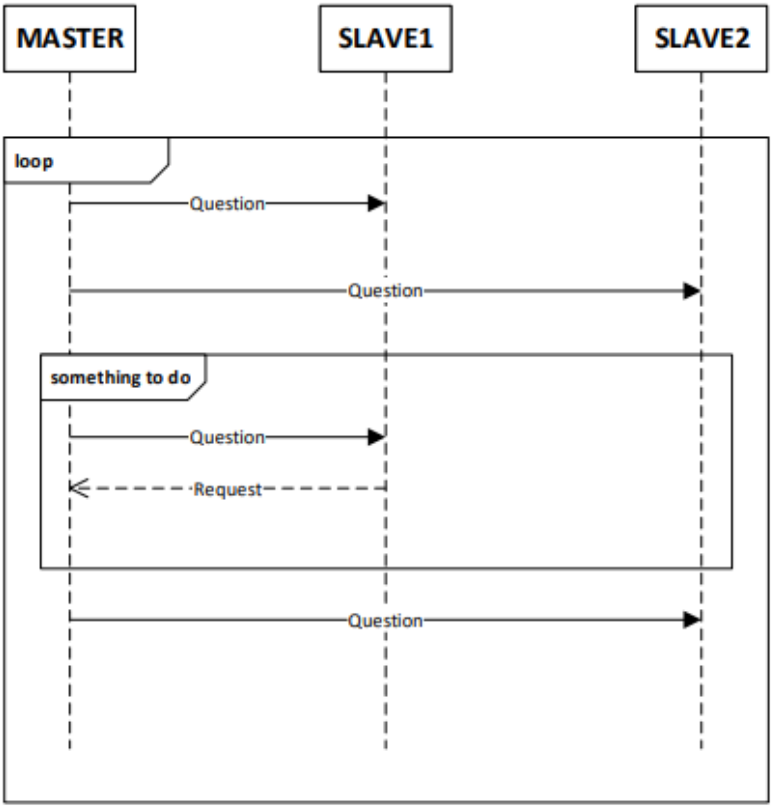
1º - El dispositivo maestro pregunta al dispositivo esclavo (1) si necesita algo.

2º - El dispositivo esclavo pide un dato que sólo conoce otro dispositivo esclavo (2).

3º - El dispositivo maestro pide al otro dispositivo esclavo (2) el dato.

4º - El otro dispositivo esclavo (2) responde al dispositivo maestro con el dato.

5º - El dispositivo maestro responde al dispositivo esclavo (1) con el dato requerido.



Como ventaja podríamos decir que es el modelo ideal para las redes de tipo halfduplex, porque se asegura que sólo habrá un dispositivo transmitiendo al mismo tiempo.

Con la arquitectura maestro/esclavo, el núcleo del sistema operativo siempre ejecuta en un determinado procesador. El resto de los procesadores sólo podrían ejecutar programas de usuario y, a lo mejor, utilidades del sistema operativo. El maestro es responsable de la planificación de procesos e hilos. Una vez que un proceso/hilo esta activado, si el esclavo necesita servicios (por ejemplo, una llamada de E/S), debe enviar una petición al maestro y esperar a que se realice el servicio. Este enfoque es bastante sencillo y requiere pocas mejoras respecto a un sistema operativo multiprogramado uniprocesador. La resolución de conflictos se simplifica porque un procesador tiene el control de toda la memoria y recursos de E/S. Las desventajas de este enfoque son las siguientes:

- Un fallo en el maestro echa abajo todo el sistema.
- El maestro puede convertirse en un cuello de botella desde el punto de vista del rendimiento, ya que es el único responsable de hacer toda la planificación y gestión de procesos.

23. Asumiendo el caso de procesadores homogéneos:

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

(b) Cite ventajas y desventajas del método escogido

Compartición de carga

Los procesos no se asignan a un procesador particular. Se mantiene una cola global de hilos listos, y cada procesador, cuando está ocioso, selecciona un hilo de la cola. El término compartición de carga se utiliza para distinguir esta estrategia de los esquemas de balanceo de carga en los que los trabajos se asignan de una manera más permanente

La compartición de carga es posiblemente el enfoque más simple y que se surge más directamente de un entorno monoprocesador. Tiene algunas ventajas:

- La carga se distribuye uniformemente entre los procesadores, asegurando que un procesador no queda ocioso mientras haya trabajo pendiente.
- No se precisa un planificador centralizado; cuando un procesador queda disponible, la rutina de planificación del sistema operativo se ejecuta en dicho procesador para seleccionar el siguiente hilo.
- La cola global puede organizarse y ser accesible usando cualquiera de los esquemas de planificación, incluyendo esquemas basados en prioridad y esquemas que consideran la historia de ejecución o anticipan demandas de procesamiento.

Se analizan tres versiones diferentes de compartición de carga:

    - Primero en llegar, primero en ser servido (FCFS). Cuando llega un trabajo, cada uno de sus hilos se disponen consecutivamente al final de la cola compartida. Cuando un procesador pasa a estar ocioso, coge el siguiente hilo listo, que ejecuta hasta que se completa o se bloquea.
    - Menor número de hilos primero. La cola compartida de listos se organiza como una cola de prioridad, con la mayor prioridad para los hilos de los trabajos con el menor número de hilos no planificados. Los trabajos con igual prioridad se ordenan de acuerdo con qué trabajo llega primero. Al igual que con FCFS, el hilo planificado ejecuta hasta que se completa o se bloquea.
    - Menor número de hilos primero con expulsión. Se le da mayor prioridad a los trabajos con el menor número de hilos no planificados. Si llega un trabajo con menor número de hilos que un trabajo en ejecución se expulsarán los hilos pertenecientes al trabajo planificado.

Usando modelos de simulación, los autores informan que, sobre un amplio rango de características de trabajo, FCFS es superior a las otras dos políticas en la lista precedente. Es más, los autores muestran que cierta forma de planificación en pandilla, expuesta en la siguiente subsección, es generalmente superior a la compartición de carga.

La compartición de carga tiene varias desventajas:

- La cola central ocupa una región de memoria a la que debe accederse de manera que se cumpla la exclusión mutua. De manera que puede convertirse en un cuello de botella si muchos procesadores buscan trabajo al mismo tiempo. Cuando hay sólo un pequeño número de procesadores, es difícil que esto se convierta en un problema apreciable. Sin embargo, cuando el multiprocesador consiste en docenas o quizás cientos de procesadores, la posibilidad de que esto sea un cuello de botella es real.
- Es poco probable que los hilos expulsados retomen su ejecución en el mismo procesador. Si cada procesador está equipado con una cache local, ésta se volverá menos eficaz.

- Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa ganen acceso a procesadores a la vez. Si se necesita un alto grado de coordinación entre los hilos de un programa, los cambios de proceso necesarios pueden comprometer seriamente el rendimiento.

A pesar de las potenciales desventajas, este es uno de los esquemas más utilizados en los multiprocesadores actuales

Sistemas Operativos 5ta Edicion, William Stallings

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

(a) Huella de un proceso en un procesador

Estado que el proceso va dejando en la cache de un procesador

(b) Afinidad con un procesador

Preferencia de un proceso para ejecutar en un procesador

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

Dependiendo del caso puede haber varias ventajas, pero en general se trata de mejorar la performance aprovechando precisamente el concepto de huella del proceso. La ejecución consecutiva de ciclos de un proceso en un mismo procesador, aumenta las posibilidades de contar con información requerida (datos, texto, stack, punteros a memoria) ya cargada en el o los niveles de caché con los que este cuenta, reduciendo en gran medida el costo en tiempo que implican la resolución de direcciones y búsqueda en memoria principal.

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

Vale aclarar antes que nada que ambos sistemas operativos trabajan con el concepto de afinidad débil por defecto, es decir que a todo nuevo proceso se asigna afinidad con el procesador que está libre para atenderlo, y se intentará que sea el mismo el que lo reciba en sucesivos llamados. La asignación concreta de una afinidad determinada por parte del usuario, tanto en Windows como en GNU/Linux implica el uso de afinidad fuerte.

Windows: Se puede modificar la afinidad de un proceso con uno o más procesadores desde la pestaña “Detalles” del administrador de tareas.

GNU/Linux: La afinidad de un proceso se puede asignar con el comando **taskset**:

El comando permite asignar afinidad a un proceso en ejecución, o definir la misma al momento de hacer la llamada a otro comando. El concepto clave respecto a los argumentos que acepta este comando es el de máscara. La máscara es un indicador del conjunto de procesadores con el que se asignará afinidad al proceso, y consiste en una secuencia de bits en los que la posición de los valores en 1 se corresponde con la asignación del procesador con número igual a la posición del bit, siendo el bit menos significativo el correspondiente al CPU 0. De esta forma, por ejemplo, la máscara 001011, asignará afinidad al proceso con los CPU 0,1 y 3. En general, de todas formas, la máscara se expresa en hexadecimal, por lo que el ejemplo anterior sería en realidad 0x0000000A. Cabe señalar que pueden indicarse procesadores que no existan en el sistema, siempre y cuando al menos uno de los asignados realmente esté

presente, y se asignarán sólo aquellos procesadores que se encuentren en el sistema. Si ninguno de los procesadores indicados existe, entonces el comando dará un error.

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

El balanceo de carga es una estrategia cuyo foco principal es una distribución aproximadamente equitativa de la carga de trabajo sobre todos los procesadores (aún si son de diferentes capacidades, que la carga sea proporcional a su capacidad en la misma medida para todos). En efecto es la estrategia (de entre varias posibles) que se escoja para asignar un proceso a un procesador determinado. Según cuál sea el criterio respecto a lo que se considere "carga equitativa", las características de los procesadores y procesos, así como la información que se posea por anticipado sobre estos últimos (dependencia de determinados recursos, tiempo requerido de CPU, memoria necesaria, etc.), pueden escogerse diversas metodologías para realizar la asignación de procesador a un proceso o conjunto de ellos.

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

Ambos conceptos están fuertemente relacionados a la historia de ejecución de los procesos entre un conjunto de procesadores, y cómo se ve afectada la performance del sistema con base en las decisiones que se tomen al respecto. Según la implementación, sin embargo, de cada uno, pueden funcionar en detrimento de una u otra forma de concebir la performance. La mejor forma de expresarlo es a través de ejemplos:

- El ejemplo más claro de la contraposición entre ambos conceptos es en el caso de la afinidad fuerte. Ésta asegura que mientras esté en el procesador, cada proceso haga un uso eficiente de su tiempo de cómputo. Sin embargo, dependiendo de la estrategia de balanceo de carga, y la certeza con la que la misma se haya aproximado a la realidad operacional de los procesos al momento de distribuirlos, puede suceder que un o más procesadores agoten sus tareas asignadas mucho antes (los procesos concluyeron mucho más rápido de lo esperado) que otros. Ante una situación de este tipo, los procesadores ahora ociosos no pueden tomar la carga de aquellos que aún se encuentran ocupados, puesto que los procesos, una vez asignada una afinidad, no pueden ser atendidos por procesadores que no pertenezcan a su grupo de afinidad. De esta forma, la carga de trabajo termina siendo no equitativa, aunque garantiza la afinidad a costa de uno o más procesadores ociosos.

- Otro ejemplo posible es cuando se maximiza el balanceo de carga a costa de la afinidad. Esto es, cuando un proceso se encuentra listo, pero no hay procesadores de su grupo de afinidad disponibles, éste es asignado a cualquier otro procesador disponible (el concepto de afinidad débil). De esta forma, la carga sobre cada procesador se mantiene más equitativa, pero una mayor parte del tiempo de atención de los procesos se pierde en la búsqueda de datos que no se encuentran ya tan cerca del CPU, sin contar además que en la medida en que el proceso cambia de procesadores con más frecuencia la huella que pueda dejar sobre cada uno es más pequeña y es más probable que la información sea borrada en el futuro próximo.

25. Si a la tabla del ejercicio 6 la modificamos de la siguiente manera: Y considerando que el scheduler de los Sistemas Operativos de la familia Windows utiliza un mecanismo denominado preferred processor (procesador preferido). El scheduler usa el procesador preferido a modo de afinidad cuando el proceso esta en estado ready. De esta manera el sheduler asigna este procesador a la tarea si este está libre.

(a) Ejecute el esquema anterior utilizando el algoritmo anterior.

Hice solo FCFS

(b) Ejecute el esquema anterior. Pero ahora si el procesador preferido no está libre es asignado a otro procesador. Luego el procesador preferido de cada job es el último en el cual ejecuto.

Hice solo FCFS

Igualmente, de que sirve afinidad si vas a ir al libre?

También interprete que ambos procesadores comparten cola.

(c) Para cada uno de los casos calcule el tiempo promedio de retorno y el tiempo promedio de espera.

(d) ¿Cuál de las dos alternativas planteadas es más performante?

La alternativa planteada en b)