# Práctica 4

## 1. Responda en forma sintética sobre los siguientes conceptos:

A. Programa y Proceso.

Programa:

- Es estático
- Existe desde que se edita hasta que se borra
- No tiene program counter
- Reside en disco

Proceso:

- Es dinámico (se puede cargar y descargar de memoria )
- Tiene program counter
- Reside en memoria RAM
- Existe desde que se ejecuta hasta que termina

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

TIempo de retorno: Es el tiempo que transcurre entre que el proceso llega al sistema hast que completa su ejecución ( todo el tiempo de vida del proceso)
Tiempo de espera: Tiempo que el proceso se encuentra en el sistema esperando, es decir, el tiempo que pasa sin ejecutarse ( tiempo que estuvo esperando en la cola de ready por ser seleccionado por el short term) // se puede obtener haciendo (TR-CPU)

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

Tiempo promedio de retorno : Es el promedio de TR de todos los procesos de un mismo lote. Se obtiene sumando todos los TR de los procesos del lote y dividiéndolo por la cantidad total de procesos.
Tiempo promedio de espera Es el promedio de TE de todos los procesos de un mismo lote. Se obtiene sumando todos los TE de los procesos del lote y dividiéndolo por la cantidad de procesos.

D. ¿Qué es el Quantum?

Es la medida que determina cuanto tiempo podrá usar el procesador un proceso

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

Preemptive: El proceso en ejecución puede ser interrumpido y llevado a la cola de ‘ready’. Este tipo de algoritmos producen mucha interacción entre procesos lo cual puede generar desperdicio de CPU

haciendo el cambio de contexto. Sin embargo, mejora el servicio porque no va a haber procesos que nunca se ejecuten por sus caracteristicas (o sea que tengan deadlock)

- Mayor overhead pero mejor servicio
- Se evita la monopolización del procesador.

Non-Preemptive: Una vez que el proceso está en estado de ejecución continua hasta que termina o se bloquea por algún evento (ej: I/O)

F. ¿Qué tareas realizan?: i. Short Term Scheduler ii. Long Term Scheduler iii. Medium Term Scheduler

Short Term Scheduler: Determina cuáles procesos de los que están en estado de 'Ready' van a pasar a ejecutarse (pasar a running).

Long Term Scheduler: Admite nuevos procesos a memoria, controla el grado de multiprogramación. Selecciona de los procesos en estado de 'New' cuales tienen prioridad de cargarse en memoria RAM

Medium Term Scheduler: Realiza el swapping (intercambio entre el disco y la memoria) cuando el SO lo necesite. Disminuye el grado de multiprogramación (quitar los procesos de la RAM y ponerlos en disco para tener la posibilidad de liberar memoria y admitir nuevos procesos)

G. ¿Qué tareas realiza el Dispatcher?

-Cambios de contexto/context switch despachando el procesos elegido por el short term
-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 el estado de los procesos en ejecución en el sistema, junto con información del uso de CPU, memoria y otros recursos.

ii. htop

versión mejorada de top que presenta una interfaz más amigable y opciones de navegación más intuitivas.

iii. ps

muestra información de los procesos activos.

Por defecto muestra:

• PID : Id del Proceso.

• TTY : Terminal.

• TIME : Tiempo de ejecución.

• CMD : Comando.

Se puede usar junto con opciones como ps aux para ver todos los procesos del sistema con detalles de usuario, consumo de CPU y memoria.

iv. pstree

Muestra los procesos en ejecución en una estructura de árbol.

v. kill

e usa para enviar señales a los procesos, siendo la señal más común SIGTERM (para solicitar la terminación de un proceso) y SIGKILL (para forzar la terminación de un proceso). La sintaxis básica es kill [PID], donde [PID] es el ID del proceso que se quiere terminar.

vi. pgreppkillkillall

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 n

vii. killall

Finaliza todos los procesos que coincidan con un nombre dado o de un usuario dado

viii. renice

cambia la prioridad de ejecución de un proceso en el sistema. Esto permite ajustar cuántos recursos del CPU se asignan a un proceso en comparación con otros. Un valor de nice más alto reduce la prioridad, mientras que un valor más bajo aumenta la prioridad. La sintaxis básica es renice [nuevo_nivel] -p [PID].

ix. xkill

permite terminar aplicaciones gráficas en entornos de escritorio. Al ejecutarlo, el cursor se convierte en una "X", y al hacer clic en una ventana, el proceso asociado se cierra. Es especialmente útil para cerrar aplicaciones que no responden en entornos gráficos.

x. atop

es una herramienta avanzada para monitorear el sistema en tiempo real, similar a top, pero con información más detallada y orientada al análisis de rendimiento. Muestra estadísticas detalladas de CPU, memoria, almacenamiento y redes, así como datos históricos si se ejecuta en modo de registro.

B. Observe detenidamente el siguiente código. Intente entender lo que hace sin necesidad de ejecutarlo

```
#i n cl u d e <s t d i o . h>
```

```
#i n cl u d e <s y s / t y p e s . h>
#i n cl u d e <u ni s t d . h>
i n t main ( v oid ) {
i n t c ;
pid_t pid ;
p r i n t f ( " Comienzo . : \ n " ) ;
f o r ( c = 0 ; c < 3 ; c++ )
{
pid = f o r k ( ) ;
}
p r i n t f ( " P r oce s o \n " ) ;
r e t u r n 0 ;
}
```

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

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.

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.

```
#i n cl u d e <s t d i o . h>
#i n cl u d e <s y s / t y p e s . h>
#i n cl u d e <u ni s t d . h>
i n t main ( v oid ) {
i n t c ;
i n t p=0;
pid_t pid ;
f o r ( c = 0 ; c < 3 ; c++ )
{
pid = f o r k ( ) ;
}
p++;
p r i n t f ( " P r oce s o %d\n " , p ) ;
r e t u r n 0 ;
}
```

i. ¿Qué valores se muestran por consola?.

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

## D. Comunicación entre procesos:

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

Los pipes son un mecanismo de comunicación entre procesos que permite transferir datos de un proceso a otro de manera unidireccional o bidireccional. Funcionan como un canal de comunicación en memoria, en el que un proceso escribe datos y otro los lee.

### ii. ¿Cómo se crea un pipe en C?.

En C, un pipe se crea con la función pipe(), definida en <unistd.h>. Su sintaxis básica es:
int pipe(int pipefd[2]);
pipefd es un array de dos enteros. Después de la llamada, pipefd[0] será el descriptor de archivo para leer, y pipefd[1] será el descriptor para escribir.

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

El parámetro requerido es un array de dos enteros (int pipefd[2]):

pipefd[0]: Descriptor del extremo de lectura del pipe. Los datos escritos en el pipe por otro proceso se leen desde aquí.

pipefd[1]: Descriptor del extremo de escritura del pipe. Los datos escritos aquí estarán disponibles para ser leídos desde el extremo de lectura.

Este array permite al sistema operativo vincular el pipe al espacio de memoria compartida entre procesos.

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

Los pipes solo permiten comunicar procesos emparentados, podemos decir que los procesos emparentados son aquellos que se crean utilizando un fork y que tienen una relación padre hijo.

## 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 SO debe tener sobre los procesos se almacena en la pcb y consta de

- **Identificación del proceso (PID)**: Un identificador único para distinguir el proceso.
- **Estado del proceso**: Indica si el proceso está en ejecución, listo, bloqueado, etc.
- **Contador de programa (PC)**: Dirección de la próxima instrucción a ejecutar.
- **Registros de la CPU**: Estado actual de los registros del proceso (acumuladores, punteros, etc.).
- **Información de memoria**: Tabla de páginas o direcciones utilizadas por el proceso.
- **Información de recursos**: Archivos abiertos, dispositivos asignados, etc.
- **Información de tiempo**: Uso del CPU, tiempo de espera, etc.
- **Información de permisos y credenciales**: ID de usuario/grupo (UID, GID).

## 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?

**Nuevo** (*New*):

- El proceso esta en el estado de inicializacion
- Un proceso es creado por otro proceso (el proceso de padre)
- Se crea la pcb y queda esperando a ser seleccionado por el long term .

**Listo** (*Ready*):

- Esta en memoria pero no se está ejecutando
- esperando a ser seleccionado por el short

**Ejecutando** (*Running*):

- Se realiza un context switch
- 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.

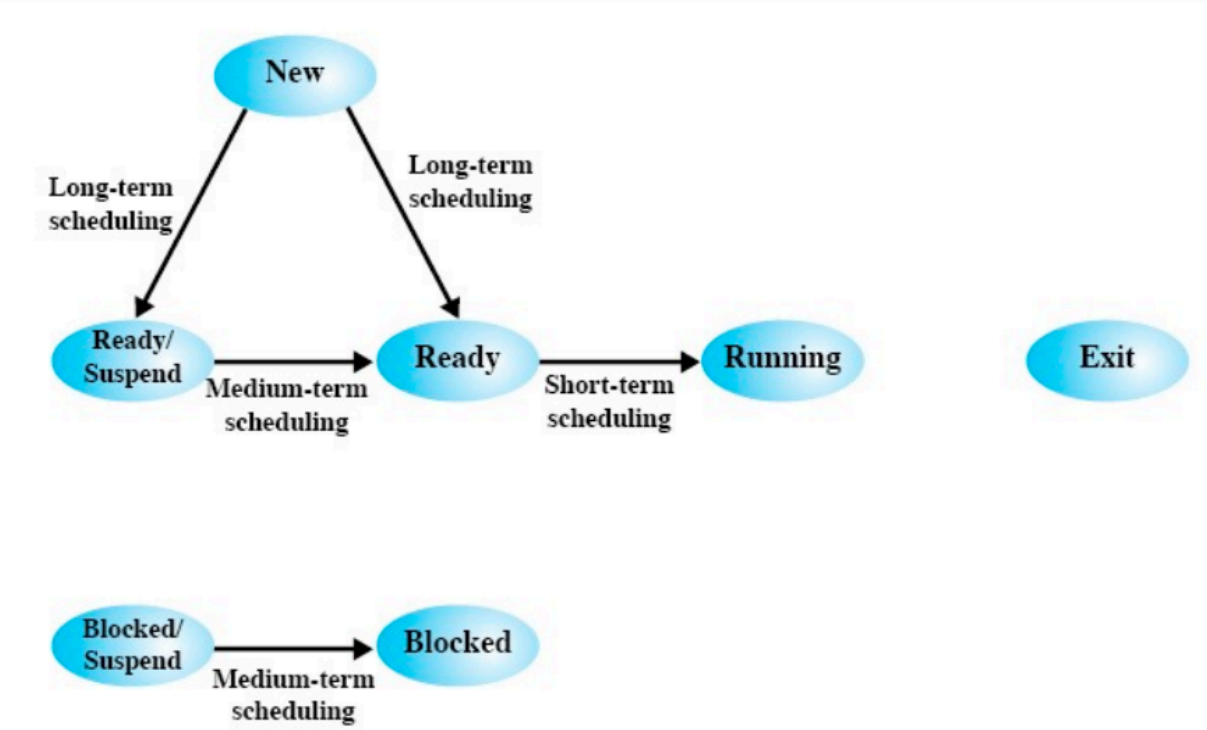
**Bloqueado** (*Blocked/Waiting*):

- El proceso está esperando un evento externo (I/O, semáforo, etc.) para poder continuar con su ejecución

**Terminado** (*Terminated*):

- El proceso ha completado su ejecución.
- Se eliminan de memoria todas las estructuras asociadas
- Finaliza con el borrado 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?

respondido en la anterior xd

## 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

Funcionamiento:

- Cuando hay que elegir un proceso para ejecutar, se selecciona el mas viejo

- 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

**EJEMPLO - FCFS**

| Proceso | Llegada | CPU | Prioridad | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | TR | TE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 9 | 3 | >1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9< | | | | | | | | | | | | | | | | 9 | 0 |
| P2 | 1 | 5 | 2 | | > | | | | | | | | 1 | 2 | 3 | 4 | 5< | | | | | | | | | | | 13 | 8 |
| P3 | 2 | 3 | 1 | | | > | | | | | | | | | | | | 1 | 2 | 3< | | | | | | | | 15 | 12 |
| P4 | 3 | 7 | 2 | | | | > | | | | | | | | | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7< | 21 | 14 |
| FCFS | | | R Queue | ~~1~~ | ~~2~~ | ~~3~~ | ~~4~~ | | | | | | | | | | | | | | | | | | | | | 14,5 | 8,5 |

SFJ

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

**EJEMPLO - SJF**

| Proceso | Llegada | CPU | Prioridad | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | TR | TE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 9 | 3 | >1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9< | | | | | | | | | | | | | | | | 9 | 0 |
| P2 | 1 | 5 | 2 | | > | | | | | | | | | | | 1 | 2 | 3 | 4 | 5< | | | | | | | | 16 | 11 |
| P3 | 2 | 3 | 1 | | | > | | | | | | | 1 | 2 | 3< | | | | | | | | | | | | | 10 | 7 |
| P4 | 3 | 7 | 2 | | | | > | | | | | | | | | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7< | 21 | 14 |
| SFJ | | | R Queue | ~~1~~ | ~~2~~ | ~~3~~ | ~~4~~ | | | | | | | | | | | | | | | | | | | | | 14 | 8 |

Round-Robin:

• Round Robin

• Política basada en un reloj

• Quantum (Q): medida que determina cuanto tiempo podr´a usar el procesador cada proceso: •

Pequeño: overhead de context switch

• Grande: ¿pensar?

• Cuando un proceso es expulsado de la CPU es colocado al final de la Ready Queue y se selecciona otro (FIFO circular)

**EJEMPLO - RR TV Q=4**

| Proceso | Llegada | CPU | Prioridad | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | TR | TE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 9 | 3 | >1 | 2 | 3 | 4 | | | | | | | | | | | | 5 | 6 | 7 | 8 | | | | | 9< | 24 | 15 |
| P2 | 1 | 5 | 2 | | >2 | | | 1 | 2 | 3 | 4 | | | | | | | | | | | | 5< | | | | | 19 | 14 |
| P3 | 2 | 3 | 1 | | | >3 | | | | | | 1 | 2 | 3< | | | | | | | | | | | | | | 9 | 6 |
| P4 | 3 | 7 | 2 | | | | >4 | | | | | | | | 1 | 2 | 3 | 4 | | | | | | 5 | 6 | 7< | | 20 | 13 |
| RR TV | | | R Queue | ~~1~~ | ~~2~~ | ~~3~~ | ~~4~~ | ~~1~~ | ~~2~~ | ~~4~~ | ~~1~~ | | | | | | | | | | | | | | | | | 18 | 12 |

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

**EJEMPLO - RR TF Q=4**

| Proceso | Llegada | CPU | Prioridad | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | TR | TE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 9 | 3 | >1 | 2 | 3 | 4 | | | | | | | | | 5 | 6 | 7 | 8 | | | | | 9< | | | | 21 | 12 |
| P2 | 1 | 5 | 2 | | > | | | 1 | 2 | 3 | 4 | | | | | | | | | 5< | | | | | | | | 16 | 11 |
| P3 | 2 | 3 | 1 | | | > | | | | | | 1 | 2 | 3< | | | | | | | | | | | | | | 9 | 6 |
| P4 | 3 | 7 | 2 | | | | > | | | | | | | | 1 | | | | | | 2 | 3 | 4 | | 5 | 6 | 7< | 21 | 14 |
| FCFS | | | R Queue | ~~1~~ | ~~2~~ | ~~3~~ | ~~4~~ | ~~1~~ | ~~2~~ | ~~4~~ | ~~1~~ | | | | | | | | | | | | | | | | | 16,75 | 10,75 |

• 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

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)

• Soluci´on: permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty

• Puede ser un algoritmo preemptive o no

**EJEMPLO - Algoritmo de prioridades**

| Proceso | Llegada | CPU | Prioridad | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | TR | TE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 0 | 9 | 3 | >1 | | | | | | | | | | | | | | | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9< | 24 | 15 |
| P2 | 1 | 5 | 2 | | >1 | | | | 2 | 3 | 4 | 5< | | | | | | | | | | | | | | | | 8 | 3 |
| P3 | 2 | 3 | 1 | | | >1 | | 2 | 3< | | | | | | | | | | | | | | | | | | | 3 | 0 |
| P4 | 3 | 7 | 2 | | | | > | | | | | | 1 | 2 | 3 | 4 | 5 | 6 | 7< | | | | | | | | | 13 | 5 |
| | | | Queue 1 | ~~3~~ | | | | | | | | | | | | | | | | | | | | | | | | 12 | 5,75 |
| | | | Queue 2 | ~~2~~ | ~~2~~ | ~~4~~ | | | | | | | | | | | | | | | | | | | | | | | |
| | | | Queue 3 | ~~1~~ | ~~1~~ | | | | | | | | | | | | | | | | | | | | | | | | |

B. ¿Alguno de ellos requiere algún parámetro para su funcionamiento?

El RR requiere definición de quantum

C. Cual es el más 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.

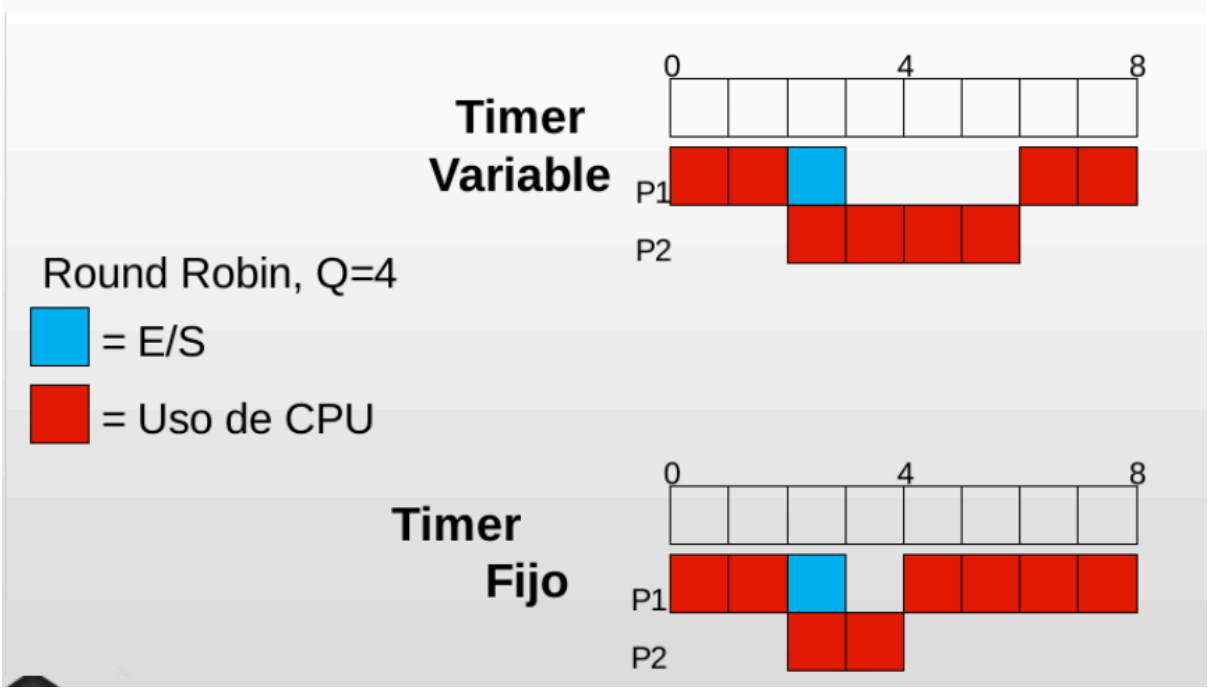
D. Cite ventajas y desventajas de su uso.

## 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
Timer fijo: El “contador” se inicializa en Q cuando su valor es cero • if (contador == 0) contador = Q;

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):

| Job | Unidades de CPU |
|---|---|
| 1 | 7 |
| 2 | 15 |
| 3 | 12 |
| 4 | 4 |
| 5 | 9 |

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

respondido en otro doc :)

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 mejores tiempos los tiene el FCFS, seguido del SFJ, el RR TV y por ultimo el RR TF. Se puede concluir que los algoritmos no apropiativos (FCFS y SJF) tienen tiempos de retorno más cortos ¿??

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

A. Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:

| Job | Llegada | Unidades de CPU |
|---|---|---|
| 1 | 0 | 4 |
| 2 | 2 | 6 |
| 3 | 3 | 4 |
| 4 | 6 | 5 |
| 5 | 8 | 2 |

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.

Quantum mas corto= menos inanición de procesos
Quantum mas largo= se comporta igual que el FCFS
Ademas cuando el Quantum es muy chico hay muchos context switch y se genera overhead

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

respondido arriba

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

Un valor de quantum alto beneficia a procesos CPU bound(ya que no dependen de informacion de E/S) y los bajos a los I/O bound.

## 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 mejor promedio pero porque los procesos son de rafagas cortas,si hubiera procesos de rafaas largas estos podrian sufrir inanicion.
El SRTF favorece a los algoritmos I/O bound.

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

| Job | Prioridad |
|---|---|
| 1 | 3 |
| 2 | 4 |
| 3 | 2 |
| 4 | 1 |
| 5 | 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?

El no apropiativo es igual al SFJ.

La ventaja que tiene es que mejora la E/S pero desfavorece a los procesos largos y de baja prioridad que pueden sufrir de inanicion

# 9. Inanición (Starvation)

A. ¿Qué significa?

La inanición es una situación en la cual un proceso en el sistema nunca obtiene acceso a un recurso compartido (en este caso la cpu) necesario para completar su ejecución debido a que otros procesos de mayor prioridad o más favorecidos continuamente acaparan dichos recursos.

B. ¿Cuál/es de los algoritmos vistos puede provocarla?

a) SJF (Shortest Job First)/ SRTF:
Estos algoritmos priorizan los trabajos más cortos o los que tienen al cual le resta menos tiempo de ejecuciòn en su siguiente r´afaga
Un proceso con un tiempo de CPU largo puede quedar postergado indefinidamente si siempre hay procesos más cortos que llegan al sistema.
b) Prioridades (Scheduler por Prioridades):
Los procesos de menor prioridad pueden no ejecutarse si continuamente llegan procesos de mayor prioridad.
La inanición ocurre especialmente cuando la prioridad de los procesos no se ajusta dinámicamente.
c) Round Robin (Timer Variable con Quantum Muy Bajo):
Aunque raro, si los procesos con alta prioridad tienen preferencia en el uso de CPU y los procesos de baja prioridad tienen un quantum pequeño, esto puede causar inanición en entornos con un número elevado de procesos de alta prioridad.

En resumen, todos los procesos que diferencien a los procesos y les asignen CPU en base a este son propensos a generar inanición

C. ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en b?

Si, se pdoría usar

- Aging: que aumenta dinámicamente la prioridad de un proceso cuanto más tiempo pasa en la cola de espera. Esto asegura que incluso los procesos con baja prioridad eventualmente sean ejecutados.
- Penalty: técnica que introduce un castigo o ajuste negativo para ciertos procesos que ocupan recursos por largos períodos o de manera intensiva (evitar que un proceso monopolice la CPU)

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:

| Job | I/O (rec,ins,dur) |
|---|---|
| 1 | (R1, 2, 1) |
| 2 | (R2, 3, 1) (R2, 5, 2) |
| 4 | (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 | I/O (rec,ins,dur) |
|---|---|
| 1 | (R1, 2, 3) (R1, 3, 2) |
| 2 | (R2, 3, 2) |
| 3 | (R2, 2, 3) |
| 4 | (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 TF

**Desventajas en sistemas con CPU-bound e I/O-bound:**

B. **Ineficiencia con procesos I/O-bound:**
   a. Los procesos I/O-bound suelen requerir muy poco tiempo de CPU antes de entrar en operaciones de entrada/salida. Como Round Robin da a todos los procesos un quantum fijo, puede desperdiciar tiempo en estos procesos si no agotan su quantum.
   b. Esto provoca que los procesos I/O-bound esperen más tiempo antes de poder volver a ejecutarse, lo que puede aumentar su tiempo de espera y retorno.

C. **Desbalance de ejecución:**
   a. Los procesos CPU-bound, que suelen requerir mucho tiempo de CPU, monopolizan más ciclos, ya que siempre consumen su quantum completo. Esto puede retrasar a

los procesos I/O-bound, que dependen de alternar rápidamente entre CPU y operaciones de I/O.

D. **Overhead por cambios de contexto:**
    a. Si hay muchos procesos I/O-bound en el sistema, el cambio frecuente entre procesos puede generar un alto overhead, especialmente si la mayoría de los procesos quedan bloqueados frecuentemente.

E. SRTF (Shortest Remaining Time First)

**Desventajas en sistemas con CPU-bound e I/O-bound:**

1. **Favorece procesos I/O-bound:**
    - Los procesos I/O-bound tienden a tener tiempos de CPU más cortos entre sus operaciones de entrada/salida. Por lo tanto, SRTF los prioriza constantemente sobre los procesos CPU-bound.
    - Esto puede provocar inanición (**starvation**) en procesos CPU-bound, que son continuamente postergados mientras haya procesos I/O-bound en la cola.
2. **Problemas con los tiempos estimados:**
    - Si los tiempos de CPU restantes de los procesos no son estimados correctamente, los procesos CPU-bound pueden ser innecesariamente penalizados, ya que el algoritmo no optimiza su ejecución.
3. **Frecuentes cambios de contexto:**
    - Con procesos I/O-bound entrando y saliendo constantemente de la cola de listos, el sistema debe interrumpir frecuentemente a los procesos en ejecución para reprogramar al proceso con el menor tiempo restante, lo que aumenta el overhead.

**Ejemplo:** Si un proceso CPU-bound está ejecutándose y llega un proceso I/O-bound con menor tiempo restante, el proceso CPU-bound será interrumpido incluso si el I/O-bound solo necesita ejecutar por una fracción mínima antes de entrar en I/O.

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 encuentran 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 faltó 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.

No entiendo como sería que nunca llegue a cero, si siempre se decrementa en 1 en cada instante de cpu asignada

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:

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

Donde:

Ti = duración de la ráfaga de CPU i-ésima del proceso.
Si = valor estimado para el i-ésimo caso
Si = 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

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

$\text{Con } 0 \langle \alpha \langle 1.$

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?

//ni en pedo hago tantas cuentas

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.

   Cola de procesos interactivos (prioridad alta):

   Los procesos interactivos requieren tiempos de respuesta rápidos porque están ligados a la interacción con el usuario.
   Algoritmo sugerido: Round Robin con un quantum pequeño.
   Ventajas:
   - Garantiza tiempos de respuesta cortos, lo cual es ideal para procesos interactivos.
   - Distribuye equitativamente el tiempo de CPU entre los procesos interactivos.

   Desventajas:
   - Puede aumentar el overhead del sistema si el quantum es muy pequeño.

   Cola de procesos batch (prioridad baja):

   Los procesos batch son aquellos que no dependen de la interacción directa con el usuario y pueden ser ejecutados en segundo plano.
   Algoritmo sugerido: SJF (Shortest Job First) o SRTF (Shortest Remaining Time First).
   Ventajas:
   - Minimiza el tiempo promedio de espera y el tiempo promedio de retorno para procesos batch.
   - Aprovecha al máximo la CPU al priorizar los procesos más cortos.

   Desventajas:
   - Puede provocar inanición en procesos más largos si llegan constantemente procesos más cortos (se puede mitigar con técnicas como aging).

B. Para el caso de las dos colas vistas en a: ¿Qué algoritmo utilizaría para planificarlas?

   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:

| Job | Llegada | CPU | I/O (rec,ins,dur) | Prioridad |
|---|---|---|---|---|
| 1 | 0 | 9 | (R1, 4, 2) (R2, 6, 3) (R1, 8, 3) | 1 |
| 2 | 1 | 5 | (R3, 3, 2) (R3, 4, 2) | 2 |
| 3 | 2 | 5 | (R1, 4, 1) | 3 |
| 4 | 3 | 7 | (R2, 1, 2) (R2, 5, 3) | 2 |
| 5 | 5 | 5 | (R1, 2, 3) (R3, 4, 3) | 1 |

Suponiendo que las colas de cada dispositivo se administran a través 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

Manejaría por ejemplo 3 colas cada una con un nivel de prioridad, los procesos más largos tendrían menor prioridad y los más cortos tendrían menor prioridad, cada cola individualmente la manejaría usando un RR con un Quantum chico que favorece a los procesos de menor duración y para solucionar el problema de inanición de los procesos más largos, para administrar las colas entre ellas usaría un algoritmo por prioridades con aging para que los procesos más largos vayan escalando en prioridad a medida que los procesos más chicos van terminando su ejecución. ???

## 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.

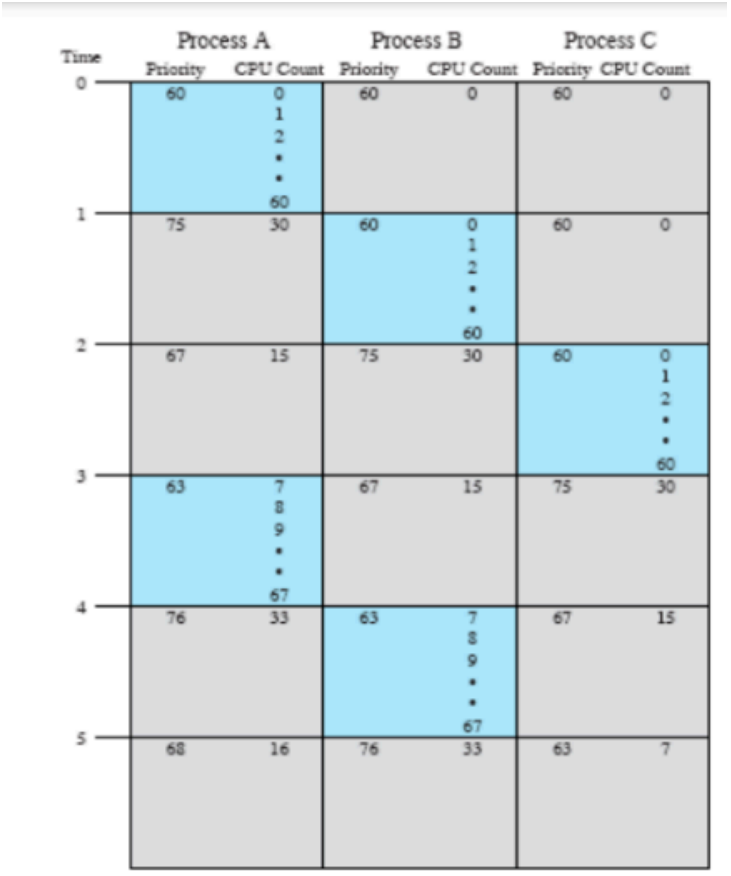
La prioridad del proceso se calcula cada segundo y se toma una nueva decisión de planificación. El propósito de la prioridad base es dividir los procesos en bandas fijas de prioridad.
Los valores de CPU y nice están restringidos para impedir que un proceso salga de la banda
que tiene asignada. Las bandas 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?
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
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



| Time | Process A | | Process B | | Process C | |
|---|---|---|---|---|---|---|
| | Priority | CPU Count | Priority | CPU Count | Priority | CPU Count |
| 0 | 60 | 0 1 2 • • 60 | 60 | 0 | 60 | 0 |
| 1 | 75 | 30 | 60 | 0 1 2 • • 60 | 60 | 0 |
| 2 | 67 | 15 | 75 | 30 | 60 | 0 1 2 • • 60 |
| 3 | 63 | 7 8 9 • • 67 | 67 | 15 | 75 | 30 |
| 4 | 76 | 33 | 63 | 7 8 9 • • 67 | 67 | 15 |
| 5 | 68 | 16 | 76 | 33 | 63 | 7 |

    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).

   Cortos acotados por CPU se ven beneficiados.
   Cortos acotados por E/S se ven beneficiados.
   Largos acotados por CPU se ve penalizados siempre que haya procesos más cortos antes que estos.
   Largos acotados por E/S se ven penalizados por lo mismo que los anteriores más el tiempo de espera para usar la E/S.

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

   no entendi xd

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

Cuando el quantum q en Round Robin es **suficientemente grande** (mayor o igual al tiempo de ejecución del proceso más largo), el algoritmo **ya no realiza interrupciones**, y los procesos son ejecutados de principio a fin en el orden en que llegaron, como en FIFO.

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?

   Multiprocesador homogeneo o multiprocesador fuertemente acoplado

B. ¿Qué significa que la asignación de procesos se realice de manera simétrica?

- Todos los procesadores tienen acceso equitativo a los recursos del sistema, incluida la memoria, dispositivos de E/S y procesos en ejecución.
- El sistema operativo distribuye las tareas (procesos o subprocesos) de manera equilibrada entre los procesadores, sin preferencia por ninguno.
- **Ventaja:** Esto maximiza la eficiencia y el rendimiento, ya que todos los procesadores trabajan de manera cooperativa.

C. ¿Qué significa que se trabaje bajo un esquema Maestro/esclavo?

- Uno de los procesadores (el **maestro**) asume la responsabilidad de controlar el sistema y tomar decisiones clave, como la planificación de tareas y la gestión de recursos.
- Los demás procesadores (**esclavos**) solo ejecutan las tareas asignadas por el procesador maestro.
- os esclavos no toman decisiones, simplemente obedecen las instrucciones del maestro.

**Ventaja:** Simplifica la implementación de un sistema multiprocesador al centralizar el control.
**Desventaja:** Puede generar un cuello de botella, ya que el procesador maestro puede volverse un punto de saturación si debe manejar demasiadas tareas

## 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

ni idea

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

A. Huella de un proceso en un procesador

La **huella de un proceso** se refiere a los datos y estructuras relacionados con un proceso que permanecen en la **caché del procesador** después de su ejecución, como instrucciones o datos frecuentemente utilizados.

B. Afinidad con un procesador

Es la preferencia de un proceso para ejecutarse en un procesador específico.

C. ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador?

**Eficiencia de la caché:**

- Si un proceso se ejecuta en el mismo procesador, puede aprovechar los datos que permanecen en la caché del procesador (aprovechar la huella), reduciendo los accesos a la memoria principal y mejorando el rendimiento.

**Reducción de latencias:**

- Cambiar un proceso entre procesadores introduce latencias debido a la migración del contexto.

**Menor overhead:**

- Mantener un proceso en el mismo procesador minimiza el tiempo invertido en sincronización de datos entre cachés (coherencia de caché).

D. ¿Puede el usuario en Windows cambiar la afinidad de un proceso? ¿y en GNU/Linux?

Si, es posible tanto en windows como en linux.
En el caso de linux el usuario puede cambiar la afinidad usando el comando taskset
Ejemplo: taskset -c 0,1 <pid>
Esto restringe el proceso al núcleo 0 y 1.

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

El **balanceo de carga** es una técnica utilizada para distribuir equitativamente las tareas (procesos o subprocesos) entre los procesadores disponibles en un sistema multiprocesador.El objetivo es mejorar la eficiencia, disponibilidad y rendimiento del sistema al evitar la sobrecarga de un recurso particular y aprovechar al máximo os recursos disponibles

F. Compare los conceptos de afinidad y balanceo de carga y como uno afecta al otro.

Ambos conceptos  son importantes en la administracion de procesos y recursos en un sistema y por eso deben gestionarse cuidadosamente, ya que un balanceo agresivo puede afectar los beneficios de la afinidad y viceversa. Un sistema bien diseñado equilibrará estos aspectos para maximizar el rendimiento y la eficiencia.

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.

| Job | Llegada | CPU | Afinidad |
|---|---|---|---|
| 1 | 0 | 4 | CPU0 |
| 2 | 2 | 6 | CPU0 |
| 3 | 3 | 4 | CPU1 |
| 4 | 6 | 5 | CPU1 |
| 5 | 8 | 2 | CPU0 |

A. Ejecute el esquema anterior utilizando el algoritmo anterior.
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.
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 mas performante?