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:

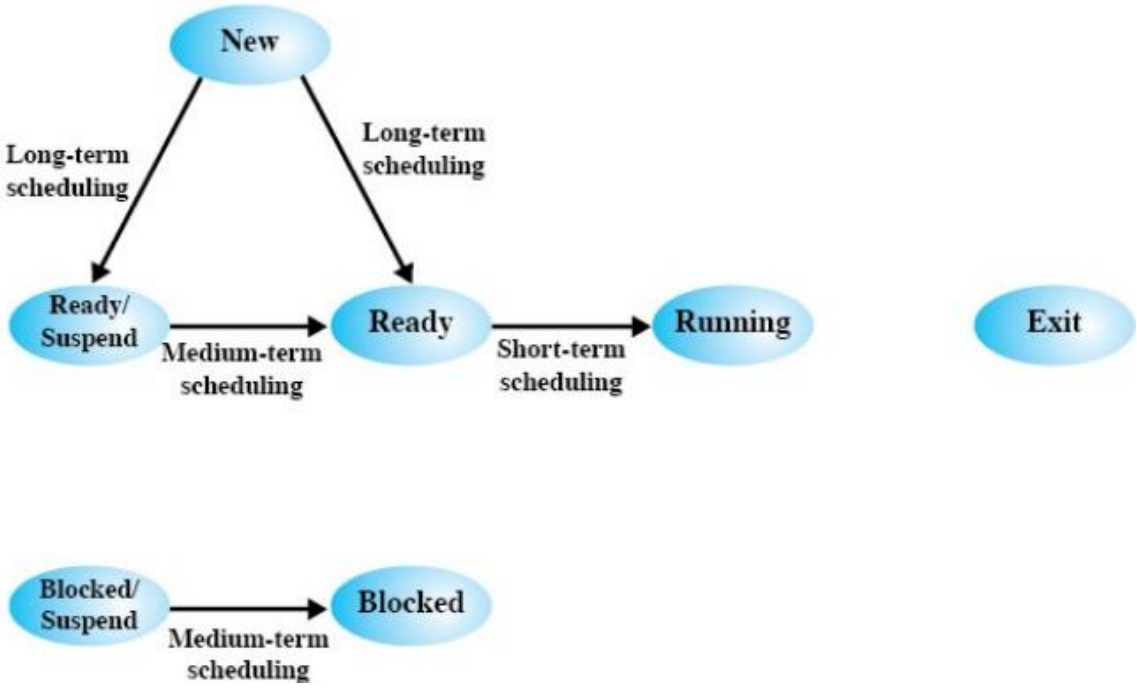
- El proceso necesita que se cumpla un evento esperado para seguir su ejecución.
- Dicho evento puede ser:
    - Terminación de una E/S requerida.
    - Llegada de una señal por parte de otro proceso.

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	9	3
2	1	5	2
3	2	3	1
4	3	7	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	16
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?

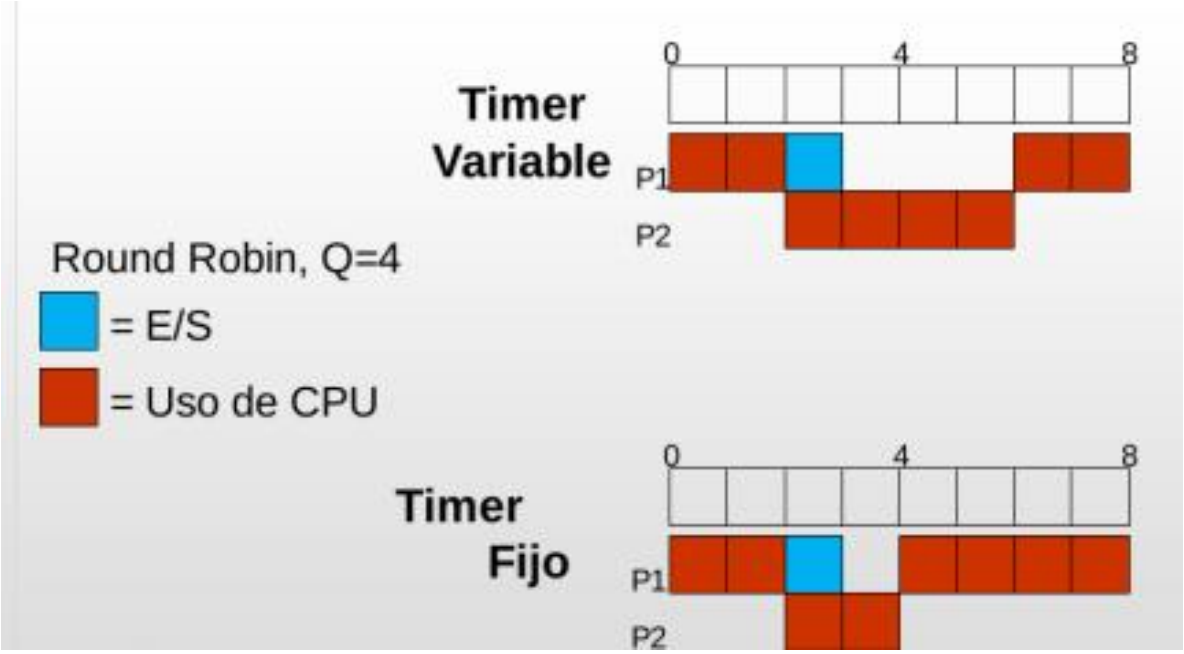
Timer Variable : El “contador” se inicializa en Q (contador := Q) cada vez que un proceso es asignado a la CPU

- 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