1. Responda en forma sintética sobre los siguientes conceptos:

A. Programa y Proceso.

Programa: Código estático almacenado en memoria secundaria, sin ejecutarse.
Proceso: Programa en ejecución, con estado, recursos asignados y contexto.

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

TR: Tiempo total desde que un Job llega al sistema hasta que finaliza.

TE: Tiempo que un Job pasa esperando (TE = TR - Tcpu)

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

…

C. ¿Qué es el Quantum?

Intervalo de tiempo fijo que un proceso puede ocupar en el procesador antes de que el sistema evalúe si debe continuar o ser reemplazado (usado en *Round Robin*).

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

Apropiativo (Preemptive): El sistema puede interrumpir un proceso en ejecución para dar paso a otro.
No apropiativo (Non-Preemptive): Un proceso mantiene el procesador hasta que finaliza o entra en espera.

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

Short Term Scheduler (Planificador a corto plazo): Selecciona qué proceso se ejecutará a continuación en el CPU. Actúa frecuentemente, basado en el estado de los procesos.
Long Term Scheduler (Planificador a largo plazo): Decide qué procesos entrarán al sistema desde la cola de entrada. Controla la carga del sistema.
Medium Term Scheduler (Planificador a mediano plazo): Suspende y reanuda procesos (swapping) para optimizar el uso de recursos.

F. ¿Qué tareas realiza el Dispatcher?

Cambia el control al proceso seleccionado por el *Short Term Scheduler*.
Realiza:

- Cambio de contexto.
- Carga de registros.
- Saltos a la dirección de inicio del proceso.

2.

a) Hecho en doc COMANDOS LINUX.

b) fork() detallado en el doc COMANDOS LINUX.

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

Un pipe crea un **canal unidireccional** de comunicación.

ii. ¿Cómo se crea un pipe en C?.

En C, la función que se utiliza para crear un pipe es `pipe()`. Esta función crea un canal unidireccional de comunicación entre dos procesos, y se especifica con un arreglo de dos enteros, que son los **descriptores de archivo**.

```
int pipe(int pipefd[2]);
```

`pipefd[0]`: Es el descriptor de archivo para el extremo de lectura del pipe.

`pipefd[1]`: Es el descriptor de archivo para el extremo de escritura del pipe.

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

El único parámetro necesario es `int pipefd[2]`.

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

En el caso de un **pipe anónimo**, este tipo de comunicación solo puede ocurrir entre procesos que tengan una relación de parentesco (como un proceso padre y su hijo).

Si quieres comunicación bidireccional entre procesos no relacionados, entonces puedes usar **pipes con nombre (FIFO)**, que se comportan como archivos y permiten la comunicación entre procesos independientes.

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

PID, PPID (proceso padre), estado del proceso (ejecutándose, esperando, etc), registros del procesador (registros de la CP, que incluyen el contador de programa, el puntero de pila y los registros de los procesos), prioridad (necesario para el scheduling), información sobre el uso de recursos como memoria, tiempo de CPU, y otros recursos del sistema, y por último información sobre las conexiones entre procesos, como las tuberías (pipes) o semáforos.

Toda esta información se almacena en una **estructura de datos llamada PCB (Process Control Block)**, es única para cada proceso.

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

Un proceso **CPU Bound** es aquel que **requiere más tiempo de CPU** que de operaciones de entrada/salida (I/O).

Un proceso **I/O Bound** es aquel que **requiere más tiempo de operaciones de entrada/salida** (como leer o escribir en disco, comunicarse con dispositivos externos) que de procesamiento en la CPU.

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

Nuevo (New):

- El proceso está siendo creado, pero aún no ha comenzado su ejecución.

Listo (Ready):

- El proceso está esperando para ser ejecutado. Está listo para ejecutarse tan pronto como el scheduler le asigne tiempo de CPU.

Ejecutando (Running):

- El proceso está actualmente siendo ejecutado por la CPU.

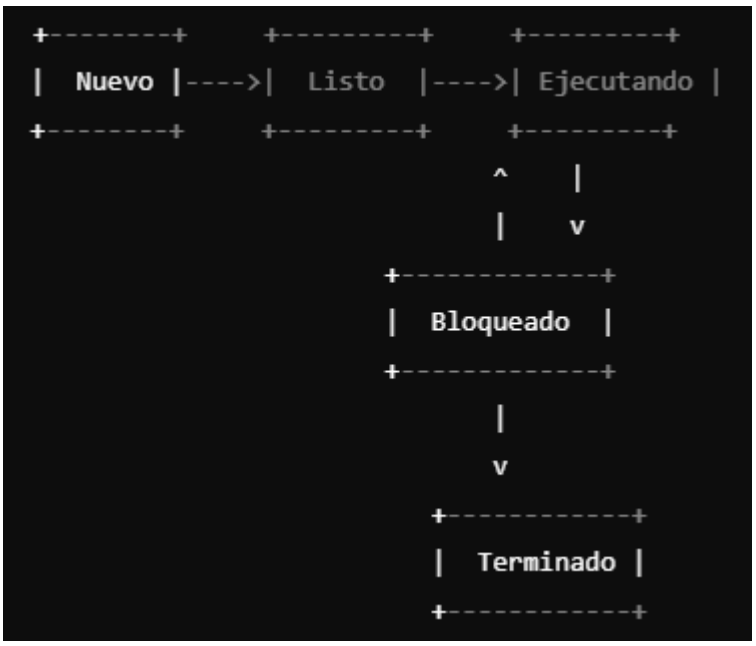
Bloqueado (Blocked o Waiting):

- El proceso no puede continuar porque está esperando que suceda un evento, como la finalización de una operación de I/O.

Terminado (Terminated):

- El proceso ha completado su ejecución o ha sido finalizado por alguna señal (como un `kill`).

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



3. Para los siguientes algoritmos de scheduling:

- **FCFS (First Coome First Served)**
- **SJF (Shortest Job First)**
- **Round Robin**
- **Prioridades**

(a) Explique su funcionamiento mediante un ejemplo.

FCFS: El primer proceso en llegar es el primero en ejecutarse.
SJF: El proceso con el menor tiempo de ejecución es el siguiente en ejecutarse. Si se cuenta con el instante de llegada de los procesos (en caso de que sean todos diferentes) entonces primero se ejecuta el que llegue en el menor instante que haya, después el factor que determina qué proceso sigue en ejecutarse (en caso de que se hayan acumulado más de 1) es el que tenga menor CPU. (al ser no-preemptive, el siguiente proceso con menor CPU deberá esperar a que termine el que se está ejecutando)
Round Robin: Cada proceso recibe una porción de tiempo (quantum) para ejecutarse. Si el proceso no termina en ese tiempo, se coloca al final de la cola de procesos listos y el siguiente proceso tiene su turno.
> P1 (4 ms), P2 (3 ms), P3 (2 ms) con quantum de 2ms.

El orden de ejecución será:
P1 → P2 → P3 → P1 → P2 → P1

El P1 final se indica porque termina siendo el último proceso, con un total de 9ms (ms total de los procesos) de demora.

Prioridades: Los procesos se ejecutan en orden de su prioridad. (MENOR **VALOR** DE PRIORIDAD = TENDRÁ MAYOR PRIORIDAD)

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

- **FCFS**: No requiere parámetros adicionales más allá del orden de llegada de los procesos.
- **SJF**: Requiere conocer el tiempo de ejecución de los procesos de antemano, lo cual no siempre es posible en la práctica.
- **Round Robin**: Requiere un parámetro adicional: el **quantum** o "tiempo de porción". Este parámetro define cuánto tiempo se le asigna a cada proceso antes de ser interrumpido.
- **Prioridades**: Requiere un parámetro adicional: la **prioridad** de cada proceso. Este valor puede ser asignado de manera estática o dinámica, dependiendo del sistema operativo.

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

FCFS: Es simple y adecuado para sistemas donde los procesos tienen tiempos de ejecución similares y no hay mucha necesidad de optimizar el uso del procesador. Sin embargo, no es adecuado para sistemas con procesos de tiempos de ejecución muy variados, ya que puede causar un "convoy effect" (procesos largos retrasando a los cortos).
SJF: Es muy eficiente si los tiempos de ejecución de los procesos son conocidos de antemano, ya que minimiza el tiempo promedio de espera. Sin embargo, en la práctica es difícil predecir el tiempo de ejecución de los procesos.
Round Robin: Es adecuado para sistemas interactivos y de tiempo compartido, donde todos los procesos deben tener una oportunidad de ejecutar durante una cantidad de tiempo limitada, asegurando que no haya un proceso monopolizando la CPU.
Prioridades: Es adecuado para sistemas donde algunos procesos deben tener una mayor prioridad que otros, como en sistemas de tiempo real o cuando se requiere que ciertos procesos se ejecuten antes que otros.

(d) Cite ventajas y desventajas de su uso.

1. FCFS (First Come, First Served):

- **Ventajas**:
    - Fácil de implementar.

    - Justo en términos de que los procesos se ejecutan en el orden en que llegan.
- **Desventajas**:
    - Puede causar largos tiempos de espera, especialmente si un proceso largo llega primero (convoy effect).
    - No optimiza el tiempo de espera promedio.

2. SJF (Shortest Job First):

- **Ventajas**:
    - Minimiza el tiempo de espera promedio si los tiempos de ejecución son conocidos.
    - Muy eficiente en términos de rendimiento de la CPU.
- **Desventajas**:
    - Requiere conocer los tiempos de ejecución de los procesos de antemano, lo que no siempre es posible.
    - Puede causar **starvation** (procesos con largos tiempos de ejecución siempre se retrasan).

3. Round Robin (RR):

- **Ventajas**:
    - Bueno para sistemas de tiempo compartido, asegurando que todos los procesos reciban tiempo de CPU.
    - Reduce el tiempo de espera para procesos interactivos.
- **Desventajas**:
    - El rendimiento depende del tamaño del quantum. Un quantum demasiado grande puede comportarse como FCFS; uno demasiado pequeño puede generar demasiadas interrupciones.
    - No es ideal para procesos de larga duración, ya que el contexto de cambio puede ser costoso.

4. Prioridades:

- **Ventajas**:
    - Ideal para sistemas donde ciertos procesos deben ejecutarse antes que otros (como en sistemas de tiempo real).
    - Permite un control fino sobre la ejecución de procesos importantes.
- **Desventajas**:
    - Puede causar **starvation** de procesos de baja prioridad.
    - La asignación de prioridades puede ser difícil de gestionar (¿cómo se asignan las prioridades?).

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

(a) ¿Qué significan estas 2 variantes?
Timer fijo: todos los procesos reciben el mismo quantum.
Timer variable: en esta variante, el **quantum asignado puede variar** según factores como la naturaleza del proceso, la prioridad, o el estado del sistema.

(b) Explique mediante un ejemplo sus diferencias.

…

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

Timer Fijo:

- El valor del quantum es constante y **puede almacenarse en el kernel del sistema operativo** como una variable global o una configuración general del planificador.
- No es necesario asociar el quantum a cada proceso individualmente.

Timer Variable:

- El quantum puede depender del proceso y, por lo tanto, debería **residir en la estructura de datos asociada a cada proceso** (por ejemplo, en la tabla de procesos o el PCB, *Process Control Block*).

9. Inanición (Starvation)

(a) ¿Qué significa?
Significa que un proceso pasa mucho tiempo esperando un recurso compartido debido al algoritmo utilizado por Scheduler

(b) ¿Cuál/es de los algoritmos vistos puede provocarla?
RR-TV
Prioridad
SJF
SRTF

(c) ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en b?
Aging (envejecimiento): Incrementa gradualmente la prioridad de los procesos que han estado esperando en la cola durante un largo tiempo.