RESOLUCION P4 - CSO

1. Responda en forma sintética sobre los siguientes conceptos:

a. **Proceso**: es un programa en ejecución, tiene program counter, es dinámico, su ciclo de vida comprende desde que se lo ejecuta hasta que termina. Se los puede clasificar en:

CPU Bound – Ligados a la CPU

I/O Bound – Ligados a entrada/salida

Programa: conjunto de instrucciones, es estático, no tiene program counter, existe desde que se edita hasta que se borra.

b. **Tiempo de retorno (TR)**: tiempo que transcurre entre que el proceso llega al sistema y completa su ejecución.

Tiempo de espera (TE): tiempo que el proceso se encuentra en el sistema esperando a ejecutarse.

c. **Tiempos promedios** de espera y de retorno es el promedio de tiempo entre los bloques de procesos ejecutados. Sirve para determinar que algoritmo de planificación conviene utilizar.

d. **Quantum:** es una medida que determina cuanto tiempo podra usar el CPU para cada proceso. Si es muy pequeño genera overhead de context switch

(desperdicio cpu para actualizar contextos) y si es muy grande habrá menor overhead, pero pérdida de equidad y mayor tiempo de respuesta, especialmente en sistemas interactivos; cada proceso tendrá mucho tiempo para ejecutarse antes de que se interrumpa y pase al siguiente.

e. **Non-preemptive (no apropiativo):** una vez que un proceso esta en estado de ejecucion, continua hasta que termina o hasta que se bloquea por algun evento como I/O.

Preemptive (apropiativo): el proceso en ejecucion puede ser interrumpido y llevado a la cola de listos. Esto genera un mayor overhead pero mejor servicio y permite que un proceso no monopolice la CPU.

d. **LONG TERM SHEDULER:**

- Decide **cuándo admitir nuevos procesos** al sistema, es decir, cuáles pasan de estar en el disco (cola de nuevos) a la **memoria principal**.
- **Controla el grado de multiprogramación**, asegurándose de que no haya demasiados procesos en memoria, lo que podría sobrecargar el sistema.

MEDIUM TERM SHEDULER:

- Realiza el **swapping**, moviendo procesos **entre memoria y disco** según sea necesario.
- Puede **suspender procesos temporalmente** (swap-out) para liberar memoria y reactivarlos después (swap-in), ajustando el grado de multiprogramación.

SHORT TERM SHEDULER:

- Decide **qué proceso de los que están en memoria se ejecutará a continuación** en la CPU.
- Funciona muy rápido y selecciona procesos de la cola de **listos** para que pasen a ejecución.

e. **DISPATCHER:** es un componente clave del sistema operativo encargado de realizar el **cambio de contexto** entre procesos. Es decir, su tarea principal es **pasar el control de la CPU** al proceso que el **short-term scheduler** (planificador a corto plazo) seleccionó para ejecutarse.

2. Procesos

a.

i. **top**: Monitorea en tiempo real los procesos y recursos del sistema.

ii. **htop**: Versión interactiva y amigable de top con colores y navegación fácil.

iii. **ps**: Muestra una instantánea de los procesos en ejecución en ese momento.

iv. **pstree**: Muestra los procesos en forma de árbol jerárquico.

v. **kill**: Envía señales a procesos para terminarlos usando su PID.

vi. **pgrep**: Busca procesos por nombre y devuelve sus PIDs.

vii. **killall**: Termina todos los procesos que coincidan con un nombre.

viii. **renice**: Cambia la prioridad de un proceso en ejecución.

ix. **xkill**: Cierra ventanas gráficas haciendo clic sobre ellas (X Window).

x. **atop:** Monitorea el rendimiento del sistema con detalles avanzados.

d.

i. Los **pipes** son un mecanismo de comunicacion unidireccional utilizado para transferir datos entre procesos; permiten que un proceso escriba en el pipe y otro proceso lea de él.

ii. En C se usa la la funcion `pipe()` para crear un pipe.

iii.
```
int pipefd[2];
int result = pipe(pipefd);
```

pipefd[0]: Descriptor para leer del pipe.
pipefd[1]: Descriptor para escribir en el pipe.

iv. Tipo de comunicacion con pipes:

   i. **Unidireccional**: un proceso escribe (usando pipefd[1]) y otro lee (usando pipefd[0])

   ii. **Bidireccional:** para lograr comunicacion bidireccional se necesitan dos pipes, uno para cada direccion.

   iii. **Comunicaciones padre-hijo:** ideal para procesos relacionados creados por `fork()`

v. **Información mínima**:

   i. Identificador del proceso (**PID**).

ii. Estado actual del proceso (listo, ejecutando, bloqueado, etc.).

iii. Contador de programa (ubicación de la instrucción actual).

iv. Contexto de la CPU (valores de registros, punteros, etc.).

v. Prioridad del proceso.

vi. Recursos asignados (memoria, archivos abiertos, etc.).

- **Estructura de datos**:
  - Esta información se almacena en la **tabla de procesos** o **PCB (Process Control Block)**.

vi. **Proceso "CPU Bound" y "I/O Bound"**

- **CPU Bound**: Un proceso que pasa la mayor parte del tiempo ejecutando operaciones en la CPU, con pocas operaciones de entrada/salida (ej., cálculos matemáticos intensivos).
- **I/O Bound**: Un proceso que pasa la mayor parte del tiempo esperando operaciones de entrada/salida, con poco uso de la CPU (ej., transferencia de datos desde un disco).

vii. **Estados posibles de un proceso:**

- **Nuevo (New)**: El proceso está siendo creado.
- **Listo (Ready)**: El proceso está esperando su turno para ser ejecutado por la CPU.
- **Ejecutando (Running)**: El proceso está siendo ejecutado en la CPU.
- **Bloqueado (Blocked/Waiting)**: El proceso está esperando un evento externo (ej., entrada/salida).
- **Terminado (Terminated)**: El proceso ha finalizado su ejecución.

3. Algoritmos de sheduling

FCFS → First Come First Served

- Se selecciona para ejecutar el programa mas viejo de la cola de listos, es decir, el primero en llegar. **Por orden de llegada.**
- No favorece a ningun tipo de proceso, pero en principio podriamos decir que los *CPU Bound* terminan al comenzar su primer rafaga, mientras que

los *I/O Bound* no.

SJF → Shortest Job First

- Politica nonpreemptive que selecciona el proceso con la rafaga mas corta. Si en la cola de listos hay un proceso que tiene requiere menos ciclos, se detiene la ejecucion del actual y se ejecuta ese. Tener en cuenta que los tiempos iran bajando a medida que se vayan ejecutando por partes.
- Los procesos largos pueden sufrir *starvation (**inanicion**)*

Round Robin

- Politica basada en un tiempo de reloj (**Quantum**).
- El Quantum determina cuanto tiempo podra un proceso utilizar la CPU. Si este es muy pequeño puede haber un ***overhead de context switch.*** Si es muy grande, el overhead disminuye pero puede perjudicar la respuesta de procesos interactivos.
- Cuando un proceso es expulsado de la CPU es colocado al final de la Ready Queue y se selecciona otro (FIFO **circular**).
- Existe un **contador** que indica las unidades de CPU que ejecuto cada proceso, que cuando llega a 0 se expulsa al proceso. Este puede ser **global** o **local** (***PCB***).
- Existen dos tipos de contadores: **Timer Fijo** y **Timer Variable.** Ejemplificando con un *Q=4*
  - **Timer Fijo**: *Se respeta el mismo quantum para todos los procesos*. Es decir, si un proceso termina en el quantum 3, el proximo proceso ejecutara su ciclo de CPU durante el quantum 4 y luego sera nuevamente expulsado y encolado (si es que no terminó).
  - **Timer Variable:** *Cada proceso tiene su quantum asignado*. Es decir, si el proceso ejecutandose solo necesito 3 de los 4 disponibles, el siguiente proceso que entre en ejecucion tendra tambien sus 4 quantums disponibles, a diferencia del Timer Fijo que solo podria utilizar uno.
- **Adecuado para:**
  - Sistemas interactivos, como sistemas multitarea, donde se necesita **respuesta rápida** para todos los procesos.
  - Balance entre procesos CPU-bound e I/O-bound.

Uso de prioridades

- Cada proceso tiene una **prioridad:** Mas chico el numero → Mayor prioridad
- Se selecciona el proceso que tenga mayor prioridad de la Ready Queue. Existe una por cada nivel de prioridad.
- Un proceso con **baja prioridad** puede sufrir starvation. **Aging/Penalty** permite cambiar la prioridad de un proceso durante su ciclo de vida como solucion.
- Pueden ser preemptive o no.

STRF → Shortest Time Remaining First

- Version *preemptive* del SJF
- Selecciona de la *"cola de listos"* el proceso al cual le queda menos tiempo de ejecucion.
- Favorece a los procesos ***I/O Bound***

11. Desventajas cuando en el sistema se cuenta con procesos ligados a CPU y procesos ligados a entrada salida

a. Round Robin (RR)

- **Procesos I/O-bound**:
  - Los procesos que dependen de operaciones de entrada/salida suelen usar poco tiempo de CPU antes de bloquearse por una operación de E/S.
  - Debido a los cambios frecuentes de contexto entre procesos, los procesos I/O-bound **podrían esperar innecesariamente** en la cola de listos después de completar su E/S, mientras la CPU ejecuta procesos CPU-bound.
- **Procesos CPU-bound**:
  - Los procesos CPU-bound pueden ser **interrumpidos constantemente** al terminar su quantum, lo que provoca que su ejecución total tarde más.

- El **overhead de cambio de contexto** aumenta si el quantum es pequeño, reduciendo la eficiencia general del sistema para procesos largos.

b. SRTF (Shortest Remaining Time First)

- **Procesos I/O-bound**:
  - Los procesos I/O-bound suelen tener tiempos de ejecución cortos y serán favorecidos porque siempre tienen menor tiempo restante.
  - Esto puede llevar a que los procesos CPU-bound **sufran starvation** (esperas indefinidas) si constantemente llegan nuevos procesos I/O-bound con tiempos más cortos.
- **Procesos CPU-bound**:
  - Al ser más largos, los procesos CPU-bound tienden a ser **interrumpidos con frecuencia** por procesos más cortos (I/O-bound), lo que genera tiempos de espera elevados y afecta el tiempo promedio de respuesta.
  - Si la carga de I/O-bound es alta, los CPU-bound **pueden nunca completar su ejecución a tiempo**.

Aspecto	Round Robin	SRTF
Procesos I/O-bound	Beneficiados moderadamente pero pueden esperar en la cola.	Son priorizados, pero puede generar starvation en otros.
Procesos CPU-bound	Fragmentación de ejecución y mayor tiempo de finalización.	Pueden sufrir starvation y largos tiempos de espera.
Overhead	Alto debido a frecuentes cambios de contexto.	Menor que RR, pero requiere cálculos frecuentes de tiempos.
Equidad	Balance razonable entre procesos.	Inequidad: favorece a los procesos cortos.

Planificacion con multiples procesadores

La planificacion de CPU es mas compleja cuando hay multiples CPUs, este enfoque divide las cargas entre distinas CPUs logrando mayores capacidades de procesamiento; si uno falla, el resto toma el control.

Criterios:

- **Planificacion temporal** → que proceso y durante cuanto

- **Planificacion espacial** → en que procesador ejecutar.
  - **Huella**: estado que el proceso va dejando en la cache de un procesador.
  - **Afinidad**: preferencia de un proceso para ejecutarse en un procesador.
- **Tipo de asignacion:**
  - **Estatica**: exist una afinidad de un proceso a una CPU
  - **Dinamica**: la carga se comparte → balanceo de carga
- **TIpo de politica:**
  - **Tiempo compartido:** se puede considerar una cola global o local a cada procesador.
  - **Espacio compartido:** grupos (threads) o particiones
- **Clasificaciones:**
  - **Homogeneos:** todas las CPUs son iguales, no existen ventajas sobre el resto.
  - **Heterogeneos:** cada procesador tiene su propia cola, clock y algoritmo de planificacion.
- **Otras:**
  - **Debilmente acoplados:** cada CPU tiene su propia memoria principal y canales.
  - **Fuertemente acoplados:** comparten memoria y canales.
  - **Especializados:** uno o mas procesadores princiales de uso general y uno o mas procesadores de uso especifico.