PRÁCTICA 4 - CSO

1. **. Responda en forma sintética sobre los siguientes conceptos:**
   a. **Programa y Proceso.**
   Programa → estático, no tiene program counter, 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.**
   TR: tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución
   TE: tiempo que pasa son ejecutarse

   c. **Defina Tiempo Promedio de Retorno (TPR) y Tiempo promedio de espera (TPE) para un lote de JOBS.**
   TPR: tiempos promedio que transcurre desde que se inicia la ejecución de un trabajo o tarea en un sistema hasta que se completa y se obtiene el resultado.
   TPR = (ΣTiempo de Espera de Todas las Tareas + ΣTiempo de Ejecución de Todas las Tareas) / Número de Tareas

   TPE: tiempo promedio que un trabajo o tarea pasa en espera antes de comenzar su ejecución
   TPE = ΣTiempo de Espera de Todas las Tareas / Número de Tareas

   d. **¿Qué es el Quantum?**
   Medida que determina cuánto tiempo podrá usar el procesador cada proceso

   e. **¿Qué significa que un algoritmo de scheduling sea apropiativo o no apropiativo (Preemptive o Non-Preemptive)?**
   Apropiativo: el proceso en ejecución PUEDE SER INTERRUMPIDO y llevado a la cola de listos
   No apropiativo: una vez que un proceso está en estado de ejecución, continúa hasta que termina o se bloquea por algún evento.

   f. **¿Qué tareas realizan?:**
   i. Short Term Scheduler: determina qué proceso pasará a ejecutarse
   ii. Long Term Scheduler: admite nuevos procesos a memoria (controla el grado de multiprogramación)
   iii. Medium Term Scheduler: realiza el intercambio entre el disco y la memoria cuando el SO lo determina (puede disminuir el grado de multiprogramación)

   g. **¿Qué tareas realiza el Dispatcher?**
   Se refiere a una parte fundamental del núcleo del SO encargada de la administración de la CPU y la asignación de recursos a los procesos o hilos en ejecución. Su función principal es la planificación y programación de la CPU para

garantizar una utilización eficiente de este recurso y un rendimiento adecuado del sistema.

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):

- **top** → proporciona vista en tiempo real de los procesos que se están ejecutando + info de la utilización de los recursos del sistema. Por defecto cada 3 segundos

- **htop** → ofrece información detallada sobre el uso de la CPU, la memoria y otros recursos, y permite al usuario interactuar con los procesos en ejecución.
- **ps** → muestra procesos en ejecución
- **pstree** → muestra gráficamente la estructura de procesos en ejecución en el sistema como un árbol jerárquico
- **kill** → se utiliza de la siguiente manera:
  kill [opciones] [ID_de_proceso]
  donde las ***opciones*** pueden ser:
  "TERM" o "-15" → solicita al proceso que se detenga. El proceso tiene la oportunidad de realizar una limpieza antes de finalizar
  "KILL" o "-9" → envía la señal SIGKILL, que fuerza la terminación inmediata del proceso sin oportunidad de limpieza. Puede causar la pérdida de datos no guardados.
  "-HUP" o "-1" → envía la señal SIGHUP, que se utiliza comúnmente para reiniciar un proceso
  Por otro lado, el ***ID del proceso*** es el identificador al que se le envía la señal. Este puede ser buscado con el comandos "ps" o "pgrep"

- **pgreppkillkillall** → ?
- **pgrep** → busca procesos basados en sus nombres o atributos y devuelve los identificadores de procesos.
- **pkill** → envía señales en función de sus nombres o atributos. Se puede especificar el nombre o criterios que deben coincidir.
- **killall** → envía señales a procesos en función de sus nombres. A diferencia de pkill y killall envía una señal a los procesos que coincidan con el nombre proporcionado
- **renice** → se utiliza de la siguiente manera:
  renice [opciones] prioridad -p PID
  donde:
  opciones → pueden especificarse opciones adicionales
  prioridad→ valor de prioridad que se desea asignar al proceso. ***Valor +***

disminuye prioridad del proceso, valor - aumenta prioridad
-p → especifica la proporción del PID para seleccionar el proceso al que se le quiere cambiar la prioridad

- **xkill** → termina de manera forzada una aplicación o proceso. Puede causar pérdida de datos no guardados.
- **atop** → proporciona información detallada del rendimiento del sistema, la utilización de recursos y los procesos en ejecución. Para utilizarlo primero hay que instalarlo
- **fork** → crea una copia exacta del proceso que la llama, y esta copia se convierte en el proceso hijo

b.	c.
#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; }	

b. Observe detenidamente el siguiente código. Intente entender lo que hace sin necesidad de ejecutarlo
El código crea tres procesos hijos (con el fork()) a partir de un proceso padre y muestra mensajes indicando si son procesos hijos o padre.

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

d. Comunicación entre procesos:
i. Investigue la forma de comunicación entre procesos a través de pipes.
“|” permite comunicar dos procesos por medio de un pipe desde la shell

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 parámetro es necesario para la creación de un pipe? Explique para qué se utiliza.
El único parámetro necesario es una array de dos enteros. Este array se utiliza para almacenar los descriptores de archivos del pipe.
El primer elemento del array es el descriptor de archivo de lectura
El segundo elemento del array es el descriptor de archivo de escritura.

iv. ¿Qué tipo de comunicación es posible con pipes?
Solo permiten comunicación de una sola vía, es decir que un proceso 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?

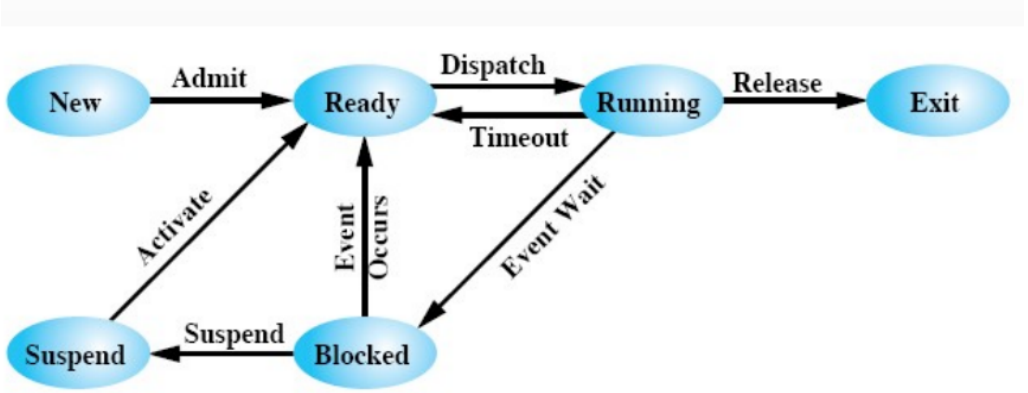
- Estructura de datos asociada al proceso

- 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 → el proceso requiere principalmente de tiempo de procesador para su ejecución

I/O Bound → 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?



3. Para los siguientes algoritmos de scheduling:
scheduling → administración de recursos del SO para ejecutar múltiples tareas o procesos de manera eficiente en una computadora.

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

(a) Explique su funcionamiento mediante un ejemplo.
(b) ¿Alguno de ellos requiere algún parámetro para su funcionamiento?
(c) Cuál es el más adecuado según los tipos de procesos y/o SO.
(d) Cite ventajas y desventajas de su uso.

- **FCFS (First Come First Served)**
  - Cuando hay que elegir un proceso para ejecutar, se selecciona el más 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
  - No es adecuado para procesos con diferentes necesidades de tiempo de CPU

- **SJF (Shortest Job First)**
  - Política non preemptive que selecciona el proceso con la ráfaga más corto
  - Cálculo basado en la ejecución previa
  - Procesos cortos se colocan delante de procesos largos

    - Los procesos largos pueden sufrir starvation (inanición)
    - Da prioridad a los procesos más cortos en términos de tiempo de CPU restante. Puede minimizar el tiempo de espera pero requiere conocimiento previo de la duración de procesos

- **Round Robin**
    - Existe un “contador” que indica las unidades de CPU en las que el proceso de ejecutó. Cuando el mismo llega a 0 el proceso es expulsado El “contador” puede ser Global o Local (PCB)
    - Existen dos variantes con respecto al valor inicial del “contador” cuando un proceso es asignado a la CPU → **Timer Variable o Timer Fijo**
    - Los procesos se ejecutan en turnos fijos y se asigna un quantum de tiempo a cada proceso. Si un proceso no termina durante su quantum, se mueve al final de la cola y se permite que otro proceso tome su lugar

- **Prioridades**
    - Cada proceso tiene un valor que representa su prioridad → menor valor, mayor prioridad
    - Se selecciona el proceso de mayot 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

(b) ¿Alguno de ellos requiere algún parámetro para su funcionamiento?
Round Robin requiere la definición del Quantum

(c) Cuál es el más adecuado según los tipos de procesos y/o SO?
- RR → más adecuado para I/O Bound
- Prioridades → No favorece a ningún tipo de procesos
- SJF → I/O Bound porque los CPU Bound al ser casi siempre lo más largos podrían sufrir inanición
- FCFS → No favorece a ningún tipo de procesos, pero en principio se podría decir que los CPU Bound terminan al comenzar su primera rafaga, mientras que los I/O Bound no.

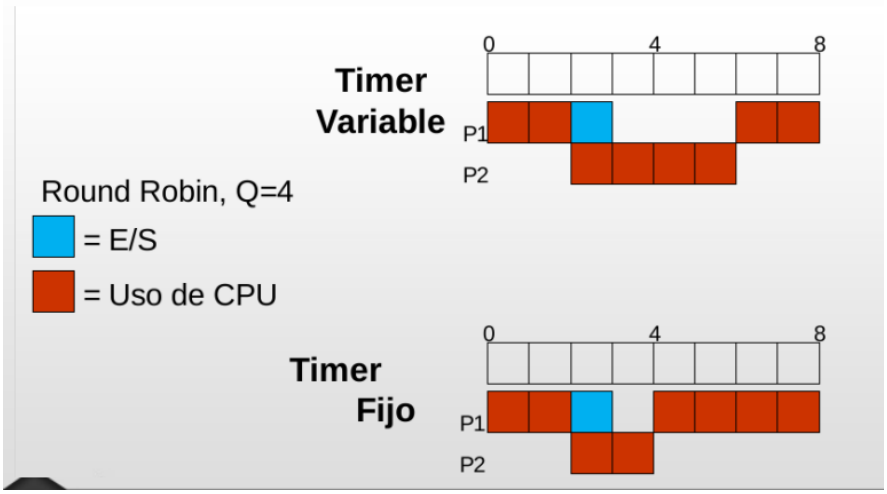
4. Para el algoritmo Round Robin, existen 2 variantes:

- **Timer Fijo**
- **Timer Variable**

(a) ¿Qué significan estas 2 variantes?
(b) Explique mediante un ejemplo sus diferencias.
(c) En cada variante ¿Dónde debería residir la información del Quantum?

TIMER VARIABLE	TIMER FIJO
• 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	• 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)



c) En caso de TV → cada proceso debe almacenar información de su Quantum en su PCB
En caso de TF → almacena en una estructura o espacio accesible directamente por el SO

9) (Starvation)
a) ¿Qué significa?
Se produce cuando un proceso se queda “atrapado” en un estado en el que no puede avanzar debido a las formas en que se gestionan las prioridades o sincronización del sistema.
(b) ¿Cuál/es de los algoritmos vistos puede provocarla?
SJF, Prioridades, RR (creo)
(c) ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en b?

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

- Ineficiencia con procesos CPU Bound: los procesos al tener un tiempo fijo asignado pueden consumir todo este tiempo de CPU sin dejar a otros procesos ejecutarse, lo que genera una baja utilización de la CPU.
- Ineficiencia con procesos I/O Bound: al tener asignado un quantum de tiempo fijo no tienen suficiente tiempo de CPU para realizar por completo las operaciones de E/S

b) SRTF (Shortest Remaining Time First)

- Inestabilidad y cambio frecuente de contexto: al tener procesos CPU Bound con tiempos largos de CPU y procesos I/O Bound con tiempos cortos de CPU puede llevar a una sobrecarga del sistema debido a los cambios frecuentes de contexto.
- Starvation (procesos largos): los procesos CPU Bound pueden experimentar inanición si se presentan procesos I/O Bound cortos con frecuencia. Esto ocurre porque los procesos largos no pueden obtener suficiente tiempo de CPU para avanzar.
- Complejidad de implementación: SRTF requiere estimación precisa de los tiempos restantes de CPU (complicado para la práctica).

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.
Puede suceder en caso de

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 fórmula:

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

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 qué 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. La fórmula anterior 1 le da el mismo peso a todos los casos (siempre calcula la media). Es posible reescribir la fórmula permitiendo darle un peso mayor a los casos más recientes y menor a casos viejos (o viceversa).**
Se plantea la siguiente fórmula: Sn+1 = αTn + (1 − α)Sn con 0<α<1

n → ráfaga, Sn → estimación, Tiempo Real → Tn

n	1	2	3	4	5	6	7	8
Sn	**10**	6	5	5.3	5	6.6	7.6	8.3
Tn	**6**	**4**	**6**	**4**	**13**	**13**	**13**	

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

S2 = 1 * 6 + (1-1)/1 * 10 = 6
S3 = ½*4 + (2-1)/2 * 6 = 5
S4 = ⅓ * 6 + (3-1)/3 * 5 = 5,3
S5 = ¼ * 4 + (4-1)/4 * 5,3 = ~5
S6 = ⅕ * 13 + (5-1)/5 * 5 = 6.6
S7 = ⅙ * 13 + (6-1)/6 * 6.6 = 7.6
S8 = 1/7 * 13 + (7-1)/7 * 7.6 = 8.3

b) **Analice para que valores de α se tienen en cuenta los casos más recientes.**
Para α = 1 tiene más peso el primer término de la ecuación (tiempo más reciente)

c) **Para la situación planteada en a) calcule qué valores se obtendrían si se utiliza la fórmula 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 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?. A su vez, se utiliza un algoritmo para administrar cada cola que se crea. Así, por ejemplo, el algoritmo podría determinar mediante prioridades sobre qué cola elegir un proceso.**
   Interactivos → RR, debido a que garantiza que cada proceso se ejecute en intervalos iguales manteniendo un buen tiempo respuesta.
   Batch → FCFS

b. **Para el caso de las dos colas vistas en a: ¿Qué algoritmo utilizaría para planificarlas?**
   Prioridades con AGING