UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Universidad Técnica Federico Santa María

Guía Certamen 1

"Practice Exercises"

Sistemas Operativos

Rodrigo Fernández - 2673002-3
rfernand@inf.utfsm.cl

Cristián Maureira - 2673030-9
cmaureir@inf.utfsm.cl

Gabriel Zamora - 2673070-8
gzamora@inf.utfsm.cl

Esteban Bombal - 2673004-k
ebombal@inf.utfsm.cl

Ignacio Villacura - 2673067-8
ivillacu@inf.utfsm.cl

Valparaíso, Septiembre 2008.

Capitulo 1

1.8 *¿Cuáles de las siguientes instrucciones debería ser priviligiada?*
R:

- Set value of timer
- Read the clock
- Clear memory
- Issue a trap instruction
- Turn off interrupts
- Modify entries in device-status table
- Switch from user to kernel mode
- Access I/O device

1.13 *En un entorno de multiprogramación y tiempo compartido, varios usuarios comparten el sistema simultáneamente.Esta situación puede resultar en varios problemas de seguridad*

- ¿Cuáles son los dos problemas?
  R:
  El robo o copiado de la información o programas de un usuario y el usar los recursos del sistema (CPU, memory,disk space, peripherals) sin un sistema de cuentas apropiados. Por ello, es necesario que exista mecanismos de seguridad, para evitar la espera permanente de un proceso, y se debe proveer de mecanismos para la sincronización de procesos a si como su debido sistema de permisos.
- ¿Podemos asegurar el mismo grado de seguridad en una máquina de tiempo compartido, como en una máquina dedicada? Explicar la respuesta
  R:
  Los sistemas dedicados, poseen un grado mayor de seguridad, ya que es limitado para los usuarios.

1.17 *Describa las diferencias entre el multiprocesamiento simétrico y asimétrico. ¿Cuáles son las 3 ventajas y 1 desventaja del sistemas multiprocesos?*
R:
En el multiprocesamiento simetrico no existe relación entre los procesadores, todos van a la par. En cambio, en el asimetrico, la relación entre los procesadores es del tipo maestro-esclavo.
Por otro lado, en el simetrico se deben manetener mecanismos de seguridad que administren el procesamiento de los jobs (que no se elija el mismo, que no se extravién, sincronizar los recursos entre ambos procesadores).
En el asimetrico, si falla el procesador maestro, falla todo el sistema.

- Ventajas Multiprocesamiento:
  1. Incrementa rendimiento, eficiencia, mayores procesos corriendo simultaneamente.
  2. Economia de Escala.
  3. Toleracia a fallas
- Desventaja:
  1. Mas complejo.

1.22 *¿Cuál es el propósito de las interrupciones?¿Cuáles son las diferencias entre una "trap" y una "interrupción"?¿Pueden las "traps" ser generadas intencionalmente por un programa de usuario?Si es asi, ¿Para qué proposito?*
R:
Las iterrupciones son los mensajes que los dispositivos envian al SO para adquirir el control de la CPU.
Las trap o excepción son interrupciones generadas a nivel de software cuando sucede algun tipo de error (ej: division por 0, loop infinito, procesos modificandose entre si o al SO), para terminar intencionalmente la ejecución de un proceso.

1.23 *DMA es usado por dispositivos I/O de alta velocidad para evitar el aumento de la carga de la ejecución de la CPU*

- ¿Cómo funciona la interfaz de CPU con el dispositivo para coordinar la transferencia?
  R:
  A traves de dos posibles metodos:
  1. Metodo simple: Rutina genérica invoca a la específica (muy lento).
  2. Tener una tabla en la cual se almacena una lista de punteros a las rutinas de interrución (se llama a la rutina de interrupción de forma indirecta a través de la tabla). Dicha matriz se indexa con un número de dispositivo unico. (Windows y UNIX)
- ¿Cómo sabe la CPU cuando las operaciones de memoria se completan?
  R: Porque los dispositivos envian una interrupción avisando que se completo la trasmición.
- La CPU esta permitida de ejecutar otros programas mientras que el controlador DMA esta tranferiendo datos.¿Puede éste proceso interferir con la execución de los programas de usuario? Si es así, describir que formas de interferencia son causados.
  R:
  No interfiere, ya que la transferencia de datos no utiliza cpu, ni ciclos de máquina.

1.24 *Algunos sistemas computacionales no proveen un modo de operación privilegiado en hardware.¿Es posible contruir un SO seguro para estos sistemas computacionales?* Dar

argumentos en caso que sea posible o no
R:
No es posible construir un SO seguro para este sistema informático, ya que se requiere protección para ciertos accesos a hardware por parte del usuario.

1.30 *Definir las propiedades escenciales de los siguientes tipos de SO*
R:

- Batch
  1. Ejecución de procesos sin interacción con humanos.
  2. Toda la informacion de entrada esta definida en scripts o en parametros de una linea de comandos.
  3. Rapido y eficiente gracias a que no esta constantemente interactuando con el usuario.
- Interactive
- Time sharing
  1. Tambien llamada multi-tarea, es una extensión logica en la cual la CPU cambia entre jobs frecuentemente de tal forma que los usuarios puedan interactuar con cada job mientras este corre.
  2. A lo anterior se le llama computación interactiva.
  3. Tiempo de respuesta: menor a 1 segundo.
  4. Procesos: Cada usuario tiene al menos un programa ejecutandose en memoria.
  5. CPU Scheduling.
  6. Swapping de procesos si estos no caben enteros en la memoria.
  7. Uso de memoria virtual para la ejecucion de procesos que no estan completamente en la memoria.
- Real time
  1. Sistema operativo multi-tarea dirigido a aplicaciones de tiempo real.
  2. Latencia mínimal de las interrupciones y del cambio de threads.
  3. Priority scheduling: Cambia de tarea solo cuando surge un evento de mayor prioridad.
  4. Time-sharing: Basicamente, usa round robin.
- Network
  1. Controla la red, sus traficos de mensajes y sus colas de acceso. Administra los recursos de la red y provee seguridad a la misma.
  2. Soporte basico de puertos de hardware
  3. Seguridad: Restricciones de autentificacion, autorizacion y login (controles de acceso).
  4. Servicios de nombre y de directorios.
  5. De acceso remoto, y administracion a traves de la red.

6. Tolerante a los fallos y posee una alta disponibilidad.

- Parallel
  1. De arquitectura mas compleja que los de programacion secuencial.
  2. Debe introducir controles para evitar problemas o bugs gracias al uso concurrente de los recursos..
  3. Mas velóz, y su velocidad es regida por la ley de Amdahl's
  4. La mayor barrera para tener un buen rendimiento es la obtencion de un buen sistema de comunicación y sincronización de las diferentes sub-procesos.
- Distributed
  1. Un tipo de computación paralela.
  2. Distribuye la memoria del sistema conectando los elementos de procesamientos a traves de una red.
  3. Gran escalabilidad.
  4. Parecido al Network SO, solo que la plataforma en la cual corre debe poseer una gran configuración y mas capacidad de RAM, altas velocidades del Procesador.
- Clustered
  1. Ejecución Remota Trasparente:
     Las aplicaciones deben ser ignorar el hecho de que un proceso puede estar corriendo en un nodo propio o remoto
  2. Balance de Carga:
     Debe implementar un inteligente mecanismo de colocación de procesos.
  3. Compatibilidad Binaria:
     Debe proveer una ejecución remota trasparente, balanceo de carga y otras características, sin requerir modificar aplicaciones ni volver a generar enlaces.
  4. Alta disponibilidad:
     Si un nodo falla, el sistema puede continuar operando.
  5. Portable:
     Tiene que poder serlo!
- Handheld
  1. Como Palm OS, Android y relativos.

Capitulo 2

2.4 *¿Cuáles son las 3 mayores actividades de un SO con respecto a la administración de almacenamiento secundario?*
R:

- Administración de espacio libre(Free-space management)
- Asignacion de almacenamiento(storage allocation)

- Planificacion de disco(disk scheduling)

2.7 *¿Cuál es el propósito de los programas de sistemas?*
R:
Los programas de sistema se pueden considerar como paquetes de utiles llamadas de sistema.Ellos proveen funcionalidades basicas a los usuarios entonces esos usuarios no tienen necesidad de escribir sus propios programas para resolver problemas comunes.

2.10 *¿Por qué algunos sistemas guardan el SO en un firmware, mientras otros lo hacen en el disco duro?*
R:
Para ciertos dispositivos, como PDAs y celulares, un disco con un sistema de archivos puede no estar disponible para el dispositivo.En estos casos el SO debe ser almacenado en el firmware.

2.15 *¿Cuáles son las 5 mayores actividades de un SO con respecto al manejo de archivos?*
R:

1. Creacion y Eliminacion de archivos.
2. Creacion y Eliminacion de direcciones.
3. El soporte de primitivas para manipular archivos y direcciones.
4. El mapeo de archivos en el almacenamiento secundario
5. El respaldo de archivos en es medios de almacenamiento estable

2.16 *¿Cuáles son las ventajas y desventajas de usar la misma interfaz de llamadas de sistema para manipular archivos y dispositivos?*
R:

2.18 *¿Cuáles son los 2 modelos de la comunicación "interprocesos"?¿Cuáles son las debilidades y fortalezas de los 2 enfoques?*
R:

- Shared-Memory Level:
  - Fortaleza:rapida, cuando los procesos estan en la misma maquina
  - Debilidad: Procesos diferentes necesitan asegurarse de que no estan esribiendo en el mismo lugar al mismo tiempo.Los procesos necesitan abordar problemas de proteccion de memoria y sincronizacion.
- Message-passing model:
  - Fortaleza:facil de implementar que el anterior
  - Debilidad:lento, porque se considera el tiempo de la configuracion de la conexion

2.20 *A veces es difícil lograr un enfoque por capas si dos componentes del SO son dependientes el uno del otro. Identificar un escenario en el cual no está claro como lograr el enfoque de capas de dos componentes del sistema que estan estrechamente unidos a sus funcionalidades.*
R: Cuando se quiere utilizar un servicio, sólo puede utilizar los servicios de niveles inferiores.
Entonces si tenemos un servicio de un nivel que necesite alguna llamada extraordinaria de un nivel superior y el resto de las llamadas sean a niveles inferiores, estaria mal hecho el enfoque por capas, porque NO se puede dejar que se necesiten los niveles superiores.un ejemplo concreto...nose..

2.21 *¿Cuál es la ventaja principal de un microkernel enfocado al diseño de sistemas?*
R:
Los beneficios normalmente incluyen los siguientes:

- añadir un nuevo servicio no requiere modificación del kernel
- es mas seguro hacer operaciones en modo usuario que en modo kernel
- un diseño simple de kernel y resultados funcionales tipicosen un so mas fiable.

¿Cómo interactuan los programas de usuarios y servicios de sistema en la arquitectura microkernel?
R:
Se comunican mediante *paso de mensajes*, el programa de usuario y el servicio nunca interactuan directamente, se comunican de manera indirecta intercambiando mensajes con el microkernel.

¿Cuáles son las desventajas de usar el enfoque microkernel?
R:
Sus principales dificultades son:

- la complejidad en la sincronización de todos los módulos que componen el microkernel y su acceso a la memoria
- mas sw de interfaz es necesario, existe la posibilidad de una perdida de rendimiento
- Mensajes de bugs pueden ser mas dificil de arreglar debido al largo viaje que tienen que tomar en comparacion a un kernel mononucleo.
- Proceso de gestion en general puede ser muy complicado.

2.22 *¿En qué formas es el enfoque de kernel modular similar al "layered approach"(enfoque de capas)?*
R:
El enfoque de kernel modular requiere que el subsistema interactue unos con otros a traves de interfaces cuidadosamente construidas que se suelen reducir (en terminos de la funcionalidad que esta expuesta a modulos externos). El Enfoque de kernel de capas es similar en este aspecto. *¿En cuántas formas es diferente?*
R:

El enfoque de kernel de capas impone un orden estricto de subsistemas de tal forma que en que en los subsistemas de capas inferiores no estan autorizados para invocar operaciones correspondientes a capas mas altas en otros subsistemas. No hay restricciones en el enfoque de kernel modular, aqui los modulos son libres de invocar operaciones unos con otros.

Capitulo 3

3.6 *Describa las diferencias entre short-term, medium-term y long-term scheduling*
R:

- **Long Term**: Determina que programas son admitidos para ejecucion y cuales deberan ser salidos. Es invocado pocas veces, segundos o minutos.
- **Medium Term**: Determina que procesos deben ser suspendidos o resumidos, es el que activa el context switch.
- **Short Term**: Determina que procesos estan listos para ocupar CPU y por cuanto tiempo. Invocado frecuentemente, milisegundos.

3.7 *Describa las acciones realizadas por el kernel para realizar el context-switch entre procesos*
R:

1. Proceso P_0 es interrumpido por llamada de sistema o interrupcion
2. Salva el estado de P_0 en PCB_0
3. Recupera informacion de proceso P_1 del PCB_1
4. Ejecuta P_1 desde el estado PCB_1

Despues hace lo mismo para los otros procesos.

3.9 *Incluyendo al proceso padre inicial, ¿cuantos procesos son creados por el programa?*

- Figura 3.28: ¿Cuantos procesos son creados?
  #include stdio.h¿
  #include unistd.h¿

  int main()
  {
  /* fork a child process */
  fork();
  /*fork another child process */
  fork();
  /* and fork another */
  fork();

```
return 0;
}
```

R:
En el primer fork() copia al proceso padre y crea un hijo(H_0) que sigue en la misma posicion donde fue creado y el padre queda en Wait. El proceso H_0 parte con 2 fork()'s adelante de el y luego hace un fork() creando otro hijo(H_1) y H_0 queda esperando. Luego H_1 hace el fork que queda y hace otro hijo, H_1 queda en wait, que no hace nada y termina y recursivamente se hace el resto. Por lo tanto se crean 8 hijos.

3.10 *Usando el siguiente programa, identifique los valores del PID en las lineas A,B,C y D. (Asuma que los PIDs actuales del padre y el hijo son 2600 y 2603, respectivamente)*

- Figura 3.29: ¿Cuales son los valores del PID?

```
#include sys/types.h¿
#include stdio.h¿
#include unistd.h¿

int main()
{
pid_t pid, pid1;
/* fork a child process */
pid = fork();

if (pidf ¡0) { /* error ocurred */
fprintf(stderr, "Fork Failed");
return 1;
}
else if (pid == 0) { /* child process */
pid1 = getpid();
printf("child: pid =%d",pid); /* A */
printf("child: pid1 =%d",pid1); /* B */
}
else { /* parent process */
pid1 = getpid();
printf("parent: pid =%d",pid); /* C */
printf("parent: pid1 =%d",pid1); /* D */
wait(NULL);
}
return 0;
}
```

R:
A:0, B:2603, C:2600, D:2600

Capitulo 4

4.1 *Provea dos ejemplos de programación en donde multi-hebras provee un mejor rendimiento que una solucion de una hebra*
R:

1. Un servidor Web que procese cada petición en threads separados.
2. Un programa GUI interactivo, por ejemplo, un debugger, donde un thread es usado para monitorear inputs, otro thread representa las aplicaciones corriendo y un tercer thread monitorea el rendimiento.

4.3 *Describa las acciones realizadas por el kernel en un context-switch entre hebras a nivel del kernel*
R:
El context-switching entre threads del kernel requiere generalmente guardar los valores de los registros de la CPU desde el thread que esta siendo 'switcheado' y recuperando los registros del CPU del nuevo thread que viene en el schedule.

4.7 *Provea dos ejemplos de programación en el cual multi-hebras no proveen un mejor rendimiento que una solucion de una hebra*
R:

1. Ejemplo 1:

```
string filenames[100];
/* Se crean threads del 1 al 100 */
create_threads();
/* Cada thread crea un archivo en particular, cuyo nombre se pasa al programa
ejecutado por el thread */

while (1){
not_created = FALSE;
for i = 0 .. 99 {
if (not(exists(filename[i])) { not_created = TRUE; }
}
if (not_created == FALSE){
printf("Done!");
return;
}
}
```

Este es el seudocódigo de un programa multihebra que no desempe na mejor rendimiento que un programa de una hebra para solucionar el mismo problema.

2. Ejemplo 2:

   Si el tiempo de computación de los procesos no difiere significativamente, un programa multihebra no es más útil en comparación a una solución de una hebra.

4.8 *Describa las acciones realizadas por una biblioteca de hebras en un context-switch entre hebras a nivel de usuario*
R:
Para realizar el context-switch entre hebras a nivel de usuario la biblioteca de threads guarda el contexto del anterior thread en su TCB (Thread Control Block) y carga el contexto del nuevo thread. El TCB contiene un stack de punteros, un Program Counter, valores de los registros y el estado actual del thread.

4.10 *Cuales de los siguientes componenes del estado del programa son compartidos por las hebras en un procesamiento multi-hebra*

1. Valores de Registro
2. Memoria Heap
3. Variables Globales
4. Memoria de Stack

R:

- Heap memory
- Global variables

4.11 *¿Puede una solución multi-hebra usando multiples hebras de niveles de usuario obtener un mejor rendimiento en un sistema con multiprocesador que dentro de uno de 1 solo procesador? Explique.*
R:
No hay mejor rendimiento, pues el sistema operativo vé solo un proceso individual y no va a gestionar los diferentes threads del proceso en procesadores diferentes.

4.13 *El programa muestro a continuación usa la API Pthreads. ¿Que es lo seria el output del programa en la linea C y linea P?*

- Figura 4.14: Programa en C para el Ejercicio 4.13
  #include ¡stdio.h¿
  #include ¡pthread.h¿

  int value = 0;
  void *runner(void *param); /* the thread */

```
int main(int argc, char *argv[])
{
int pid;
pthread_t tid;
pthread_attr_t attr;

pid = fork();

if (pid == 0) { /* child process */
pthread_ttr_init(&attr);
pthread_create(&tid,&attr,runner,NULL);
pthread_join(tid,NULL);
fprintf("CHILD: valuea = %d",value); /* LINE C */
}
else if (pid ¿0) { /* parent process */
wait(NULL);
fprintf("PARENT: valuea = %d",value); /* LINE P */
}

void *runner(void *param) {
value = 5;
pthread_exit(0);
}
}
```

R:
Linea C, 5.
Linea P, 0.

Capitulo 5

5.2 *Explicar la diferencia entre scheduling interrumpible y no interrumpible*

El scheduling no interrumpible ocurre cuando un proceso esta sienco ejecutado por la CPU por cierto intervalo de tiempo, la CPU no puede tomar otro proceso y dejar a medias la ejecución. En cambio el scheduling interrumpible puede si puede tomar otro.

5.3 *Suponer que los siguientes procesos arriban para ejecucion en los tiempos indicados. Cada proceso correra el tiempo indicado. Responder usando scheduler no interrumpible.*

Proceso	Tiempo de llegada	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a Cual es el tiempo de ejecucion promedio con el algoritmo FCFS

P1 - P2 - P3

$(8 + (7{,}6 + 4) + (11 + 1))/3 = 10{,}53$

b Cual es el tiempo de ejecucion promedio con el algoritmo SJF

P3 - P2 - P1

$((1 + 1) + (1{,}6 + 4) + (6 + 8))/3 = 7{,}2$

c El algoritmo SJF se supone para mejorar el rendimiento, pero aviso de que hemos elegido elegido P1 a en tiempo 0 porque no sabiamos que el proceso mas corto llegaria pronto. Calcular el tiempo promedio si la CPU se deja inactiva durante 1 tiempo, para luego usar SJF. Recuerde que los procesos P1 y P2 estan a la espera durante el tiempo de inactividad, por lo que su tiempo de espera puede aumentar.

Si se deja inactiva la CPU, todos los procesos estan listos al comenzar a procesar, por lo que el orden es el mismo:

P3 - P2 - P1

$((1 + 1) + (0{,}6 + 1 + 4) + (6 + 8))/3 = 7{,}2$

5.4 *Que ventaja se tiene con diferentes tama nos de quantum de tiempo en diferentes niveles de un sistema de multinivel de colas*

Si se tuviera quantum de tiempos muy largos, sería parecido a una cola FCFS, y no se repartiría bien los tiempos entre los procesos. Por el contrario con quantum de tiempos cortos, se produce muchos cambios de contexto, con la perdida de eficiencia, ya que el CPU se encarga de repartir los tiempos entre los procesos en vez de procesarlos. Con diferentes quantum de tiempos se garantiza la ejecución uniforme segun el tiempo de ejecución de cada proceso.

5.5 *Muchos algoritmos de CPU-scheduling son parametrizados. Por ejemplo Round Robin requiere un parametro para indicar el time slice (tiempo de corte). Multinivel Feedback Queues requieren parametros para definir el numero de colas, el algoritmo de scheduling para cada cola, el criterio usado para mover procesos entre colas, etc.*
Estos algoritmos son, por lo tanto realmente conjuntos de algoritmos (por ejemplo, el conjunto de algoritmos de RR para todas los tiempos de corte, y asi sucesivamente). Un conjunto de algoritmos pueden incluir otro (por ejemplo, el FCFS es un Round Robin con tiempo de cuantum infinito). Cual relacion hay entre los siguientes pares de conjuntos de algoritmos

a Prioridad y SJF

Tienen relacion debido a que SJF asigna prioridades prediciendo el tiempo de ejecicion.

b Colas feedback multinivel y FCFS

En las primera colas del multinivel, los quantum de tiempo son peque nos, luego van aumentando hasta que en la ultima cola, hasta que la ultima cola se convierte en un FCFS, debido a que el quantum de tiempo es igual al tiempo de proceso.

c Prioridad y FCFS

Son algoritmos diferentes, debido a que el algoritmo de prioridad asigna un entero para la prioridad, dependiendo si ocupa CPU o E/S, pero FCFS asigna prioridades por orden de llegada.

d RR y SJF

Tienen relacion ya que en el Round Robin, en las primeras colas, completa los procesos con menores tiempos de ejecucion, lo que es similar al SJF donde se da prioridad a los procesos mas cortos.

5.10 *Discutir como los siguientes pares de criterios de scheduling entran en conflicto en ciertas configuraciones.*

a Utilizacion de CPU y tiempo de respuesta

Puede suceder que si se tiene un Round Robin, en donde los quantum de tiempos son peque nos, se logre tener siempre la CPU al tope trabajando, pero teniendo entre sus operandos latencias que sumadas, generen un tiempo considerable a la hora de calcular el tiempo de respuesta promedio de los procesos.

b Tiempo promedio de ejecucion y maximo tiempo de espera promedio.

Aca no supe, por que si se tiene un tiempo de ejecucion alto, tambien su espera es alta, van de la mano, no se me ocurrio un ejemplo que se contradiga esto, habria que verlo experimentalmente o llevar la duda a otro lado.

c Utilizacion de dispositivos de E/S y utilizacion de CPU

Se supone que los CPU I/O Burst son el tiempo en donde la cpu ejecuta procesos pero los dispositivos de E/S esperan, podria suceder que se requiera tanto sacarle el maximo provecho a la CPU, como al dispositivo de E/S, como en un juego que exija harto recurso, si se desea algo eficiente, tendria que darsele harta prioridad tanto a los dispositivos E/S como al procesamiento de los datos y no en forma extrema para cada lado.

5.12 *Considerar el siguiente gurpo de procesos. con el largo de tiempo de CPU burst en milisegundos*

Proceso	Burst Time	Prioridad
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Se asume que los procesos arriban en el orden $P_1 \cdots P_5$ en el tiempo 0

a Dibujar 4 cartas Gantt que ilustren la ejecucion de esos procesos usando los algoritmos: FCFS, SJF, prioridad no interrumpible (un peque no numero implica alta prioridad), y RR (quantum = 1).

FCFS	P1	P2	P3	P4	P5										
	0	10	11	13	14										
SJF	P2	P4	P3	P5	P1										
	0	1	2	4	9										
Prioridad NI	P2	P5	P3	P1	P4										
	0	1	6	8	18										
RR	P1	P2	P3	P4	P5	P1	P3	P5	P1	P5	P1	P5	P1	P5	P1 ... P1
	0	1	2	3	4	5	6	7	9	10	11	12	13	14	15

b Cual es el tiempo de ejecucion promedio para cada proceso en cada algoritmo del punto a

EL procedimiento es el mismo al 5.2

c Cual es el tiempo de espera de cada proceso en cada algoritmo del punto a.

FCFS: $(0 + 10 + 11 + 13 + 14)/5 = 9{,}6$
SJF: $(0 + 1 + 2 + 4 + 9)/5 = 3{,}2$
Prioridad NI: $(0 + 1 + 6 + 8 + 18)/5 = 6{,}6$
RR: $(t_{p1} + ...t_{p5})/5 = (15 + 1 + 6 + 3 + 14)/5 = 7{,}8$

d Cual es el agoritmo que da como resultado el minimo tiempo de espera de promedio.
SJF

5.13 *Cual de los siguientes algoritmos de sheduling podria producir starvation*

a FCFS

b SJF

c RR

d Prioridad (correcta, ya que los procesos con prioridad menor podrian nunca ejecutarse, anecdota de los 30 a nos)

5.16 *Considere a un sistema con scheduler de multinivel de colas. Cual estrategia puede usar un usuario para maximizar la cantidad de tiempo del CPU para procesos de usuario*

NO SE ME OCURREEE

5.18 *Explicar las diferencias en cuanto los siguientes algoritmos de scheduling discriminan en favor de procesos cortos*

a FCFS: De ninguna forma ayuda a los procesos cortos, puede producirse el fenomeno del convoy y dejar a los mas cortillos al final y se demorarian demasiado

b RR: Ayuda ya que le da cierto intervalo de tiempo a cada proceso, independiente de su prioridad o tiempo de ejecucion, solo afecta la cantidad de procesos, solo si son muchos procesos y si el quantum de tiempo es peque no, el proceso peque no podria esperar harto.

c Multinivel de colas feedback: Ayuda mucho ya que limita las ejecuciones de procesos largos en intervalos de tiempo peque nos, de esta manera no se produce efectos convoy, ademas de derivar los proceso a otras colas con quantum mas alrgos.

Capitulo 6

6.8 *Condiciones de carrera son posibles (Race conditions) en muchos sistemas de computadoras. Considere la posibilidad de un sistema bancario con dos funciones: deposit(amount) y draw(amount) (depositar y retirar cantidad). Estas dos funciones se pasan la cantidad que se depositó o retiró de una cuenta bancaria. Supongamos una cuenta bancaria compartida existe entre un marido y la mujer y simultaneamente el marido llama a draw() y la esposa llama a deposit(). Describre como una condicion de carrera es posible y lo que se podria hacer para evitar que se produzca la condicion de carrera.*
R:

6.11 *¿Cuál es el significado del termino busy waiting?* R:
Ocurre cuando los loops continuos de la CPU, no hacen nada util, pero esperan que cualquier evento ocurra. Puede ser usado en la entrada de una seccion critica.
¿Qué otros tipos de espera hay en un sistema operativo. R:
Un proceso/hebra puede tambien ser bloqueado-esperando en una cola asociada con dispositivos I/O, semaforos, lock, o una condicion variable.. R:

6.13 *Explique por que la implementacion de las primitivas de sincronizacion por deshabilitacion de interrupciones no es apropiado en un sistema con unico procesador si las primitivas de sincronizacionvan a ser utilizados en programas a nivel de usuario.*
R:
Si un programa de nivel de usuario esta dando la habilidad para desabilitar interrupciones, entonces puede desabilitar el timer de interrupciones y prevenir el cambio de contexto, de tomar un lugar, esto dejando usar el proceso, sin dejar que otros procesos se ejecuten.

6.15 *Describir 2 estructuras de datos del kernel, en la cual las condiciones de carrera (race conditions) son posibles. Incluir una descripcion de como una condicion de carrera*

puede ocurrir.
R:

6.20 *Mostrar como implementar las operaciones wait() y signal() de un semaforo en un entorno multiprocesador usando TestAndSet(). La solucuon deberia reflejar un minimo busy waiting*
R:

int guard = 0; int semaphore value = 0; wait() while (TestAndSet(&guard) == 1); if (semaphore value == 0) atomically add process to a queue of processes waiting for the semaphore and set guard to 0; else semaphore value–; guard = 0; signal() while (TestAndSet(&guard) == 1); if (semaphore value == 0 && there is a process on the wait queue) wake up the first process in the queue of waiting processes else semaphore value++; guard = 0;

6.23 *Escribir un monitor de buffer delimitado (bounded-buffer) en el cual los buffers (porciones) son embebidos dentro del mismo monitor.*
R:
monitor bb char buffer[BUFFER_SIZE];
int in, out, count;
condition not_full, not_empty;
void produce(char x)
if(count==BUFFER_SIZE)
not_full.wait();

buffer[in]=x;
in=(in+1)count++;
not_empty.signal();
void consume(char x)
if(count==0)
not_empty.wait();

x=buffer[out];
out=(out+1)count–;
not_full.signal();
void init()

6.32 *Escribir un monitor que implemente un reloj alarma, que active una llamada a un programa para retrasarse el mismo para un numero de unidades de tiempo especidico (ticks). Puedes asumir la existencia de un reloj real por hardware en donde se invoque un procedimiento tick en tu monitor en intervalos regulares.*
R:
Answer: type alarm = monitor var X: condition;
procedure delay (T: integer); begin
X.wait (T);

```
X.signal;
end
procedure tick; begin
X.signal
end
```