Examen Diciembre 2006

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del examen.

Formato

- **Indique su nombre completo y número de cédula en cada hoja (No se corregirán las hojas sin nombre, sin excepciones) Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.**
- **Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva. (No se corregirá la hoja que tenga el ejercicio compartido, sin excepciones)**
- **Si se entregan varias versiones de un problema solo se corregirá el primero de ellos.**

Dudas

- **Sólo se contestarán dudas de letra.**
- **No se aceptarán dudas en los últimos 30 minutos del examen.**

Material

- **El examen es SIN material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del examen, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.**

Aprobación

- **Para aprobar el examen se debe tener un ejercicio entero bien hecho y medio más.**

Finalización

- **El examen dura 4 horas.**

Problema 1

1. ¿Qué beneficio brinda la multiprogramación?
   Discuta según el sistema sea monoprocesador o multiprocesador.
2. Describa 3 métodos para efectuar una operación de Entrada/Salida. (Formas de interacción con la controladora del dispositivo).
3. El modo de ejecución provisto por hardware permiten brindar protección entre los procesos. Describa dos formas más de protección que se proveen a nivel de hardware.
4. Los *system calls* son una forma a los servicios del sistema operativo. ¿Existe alguna otra forma de acceder a estos servicios por parte de los procesos de usuario?
5. En el *Process Control Block* existe una sección para los registros. ¿Cuándo es utilizada esta área?
6. ¿A nivel de sistema operativo y *Process Control Block*, cuáles estructuras son necesarias para describir un hilo (*thread*)?
7. Calcule el tiempo de espera promedio para los siguientes procesos:

Proceso	Burst Time
P1	5
P2	6
P3	2
P4	3

   Utilizando:
   a) Un planificador *Shortest Job First* (SJF).
   b) Un planificador *Round-Robin* con tiempo de quantum 1.

   Nota: asuma que el sistema comienza con los 4 procesos en la cola de procesos listos con el orden P1, P2, P3 y P4.
8. Describa el método de asignación de direcciones (*address binding*) en tiempo de ejecución (*execution time*).
9. En un sistema que soporta memoria virtual. ¿Quién realiza las traducciones de memoria virtual a física? Describa como se hace la traducción de virtual a física en paginación.
10. Las estructuras RAID (*Redundant Arrays of Inexpensive Disks*) brindan servicios de mejora en la confiabilidad y en los tiempos de transferencias. Haga una tabla que clasifique los RAID tipo 0, 1 y 5, mencionando que servicio(s) mejoran.

Solución:
Nota: Las siguientes respuestas son una guía de la respuesta a las preguntas. No necesariamente está completa la respuesta.

1.
La maximización del recurso procesador tanto en sistemas de monoprocesador como en multiprocesadores.
2.
Polling (Espera activa), Interrupts (Interrupciones) y DMA.
Falta describir c/u.
3.
Pueden ser:

Instrucciones privilegiadas, protección de accesos a memoria a través de la MMU o protección de CPU a través de un timer.

Falta descripción.

4.
No. Los system calls son la única de acceder a los servicios.

5.
Cuando el proceso no tiene asignado el recurso procesador. En esa área se guarda el estado de los registros del procesador cuando el proceso fue despachado. Cuando el planificador le asigne el recurso procesador, el despachador cargará los registros del procesador con esos valores.

6.
A nivel del PCB, el sistema debe tener una estructura independiente para cada thread de un proceso que contenga por lo menos el stack, registros y el program counter.

7.
a) (0 + 2 + 5 + 10) / 4 = 4.25
b) (9 + 10 + 5 + 8) / 4 = 8 (Se tomo el orden P1, P2, P3 y P4).

8.
El método de asignación en tiempo de ejecución permite a la memoria de un proceso ser reubicada del lugar físico mientras el proceso está activo (ejecutando, listo, bloqueado) en el sistema.

9.
La traducción es realizada por el dispositivo MMU (Memory Management Unit). En el cambio de contexto, cuando un proceso es asignado a un procesador, se carga el registro PTBR (Pointer Table Base Register) de la MMU con la dirección base de la tabla de página. Cada acceso a memoria (virtual) generado por un proceso es traducido a memoria física por la MMU.
En paginación la dirección virtual se compone de una sección que indica la entrada de la tabla de páginas que se accede (para obtener el número de marco) y de otra sección que indica el desplazamiento dentro del marco.

10.

RAID	Confiabilidad	Tiempo acceso
0	No	Sí
1	Sí	Sí y No.*
5	Sí	Sí

*** Depende de la implementación. Se toma como válida cualquier respuesta.**

Problema 2

Tenemos un sistema operativo con soporte de threads en el kernel que gestiona su memoria con memoria virtual implementada como paginación bajo demanda, de dos niveles, y usa política FIFO de reemplazo con asignación local. El sistema dispone de 16MB de memoria RAM. Las direcciones virtuales que llegan a la MMU tienen 16 bits, de los cuales los 4 primeros indexan en la tabla de primer nivel. Las páginas ocupan 1KB.

En un determinado instante van a ejecutarse dos procesos independientes A y B. El sistema operativo ha asignado 2 marcos al proceso A y 3 al proceso B. Secuencialmente el proceso A leerá su página 1, escribirá en sus páginas 2, 5 y 6, y finalmente leerá las paginas 2 y 3. El proceso B presenta un patrón idéntico.

Se pide responder lo siguiente:

a) ¿Cuantos bits ocupa una dirección que sale de la MMU? ¿Que tamaño tiene la memoria virtual? ¿Cuantas entradas tienen las tablas de segundo nivel?

b) Describe el estado de las tablas de páginas de los procesos y su evolución temporal, indicando los fallos de página y los accesos a disco realizados, así como los bits de presente/ausente, referencia y modificación. Si el reemplazo fuera con el algoritmo de segunda oportunidad, ¿sería diferente la evolución? Si procede, indicar en que.

c) ¿Influye la planificación de CPU en la evolución de las tablas de páginas?

d) ¿Cambiaría algo si en vez de ser dos procesos fueran dos threads del kernel del mismo proceso?

Solución:

a) Las direcciones que salen de la MMU son direcciones físicas. Si hay 16MB de memoria física, se necesitan 24 bits para direccionala. Las direcciones lógicas que llegan a la MMU ocupan 16 bits, por lo que la memoria virtual ocupa 2¡6 = 64KB. De esos 16 bits, 10 se emplean como desplazamiento dentro de la página (puesto que el marco es de 1KB), quedan 6 para el número de página. Como 4 se usan para indexar en la tabla de primer nivel, quedan 2 para indexar en la de segundo nivel, que por lo tanto tiene 4 entradas.

b) Como el sistema utiliza reemplazo local, cada proceso emplea solo los marcos que le han sido asignados y cuando ocurre un fallo de páginas, carga la nueva pagina en un marco que ocupaba otra pagina suya. Se elige la víctima a reemplazar entre las páginas del propio proceso. Por lo tanto se puede describir independientemente la evolución de las tablas de páginas de los procesos A y B. Para el proceso A, inicialmente la tabla de primer nivel y de segundo nivel están vacías. Los dos primeros accesos provocan sendos fallos de página, ambos caen en la primera entrada de la tabla de primer nivel. Supongamos que el sistema operativo le asigna los marcos 33 y 29. El primer acceso es una lectura y por ello no prende el bit de modificación. El segundo es una escritura y si lo pone a 1. El orden FIFO queda 1-2. Las tablas de

segundo nivel quedan como muestra la figura 1(a). La tabla de primer nivel no se muestra, pero solo tendría la primera entrada ocupada, apuntando precisamente a la primera tabla de segundo nivel. La escritura de la pagina 5 provoca un fallo de pagina y como no hay marcos físicos libres de este proceso, se invoca al algoritmo de reemplazo. El algoritmo FIFO elige como víctima la 1. Como la pagina 1 solo había sido leída, este fallo de pagina solo implica traer de disco la pagina 5. Las tablas quedan en el estado 1(b) y el orden FIFO 2-5. como las tablas de paginas de segundo nivel tienen 4 entradas, esta pagina 5 cae en la segunda tabla de segundo nivel, la apuntada por la segunda entrada de la tabla de primer nivel. La escritura de la pagina 6 provoca un nuevo fallo de pagina, y la víctima en este caso es la pagina 2, que ha de ser llevada a disco pues tenía el bit de modificación activo. El orden FIFO queda 5-6. La pagina 6 también cae en la segunda tabla de segundo nivel (figura 1(c)). La lectura de la pagina 2 provoca un nuevo fallo de pagina, el reemplazo elige como víctima a la 5, que manda a disco, y en el marco que ella ocupada se mete la página 2 (figura 1(d)). El orden FIFO queda 6-2. Obsérvese como esta página lógica primero se cargo en el marco 29 y ahora se carga en el 33. Finalmente la lectura de la pagina 3 provoca otro fallo de pagina y el reemplazo de la pagina 6 de la memoria física (figura 1(e)). El excesivo número de fallos de pagina ralentiza en exceso la ejecución de este proceso y puede hacer pensar que el numero de marcos asignados al proceso es insuficiente, provoca vapuleo y deba aumentarse. Si la política de reemplazo fuera la de segunda oportunidad, la evolución de las tablas para esta carga particular no hubiera variado nada, se habrían elegido las mismas paginas víctima.

	#marco	P	R	M
0	X	0	0	0
1	33	1	1	0
2	29	1	1	1
3	X	0	0	0

	#marco	P	R	M
4	X	0	0	0
5	X	0	0	0
6	X	0	0	0
7	X	0	0	0

(a)

	#marco	P	R	M
0	X	0	0	0
1	X	0	0	0
2	29	1	1	1
3	X	0	0	0

	#marco	P	R	M
4	X	0	0	0
5	33	1	1	1
6	X	0	0	0
7	X	0	0	0

(b)

	#marco	P	R	M
0	X	0	0	0
1	X	0	0	0
2	X	0	0	0
3	X	0	0	0

	#marco	P	R	M
4	X	0	0	0
5	33	1	1	1
6	29	1	1	1
7	X	0	0	0

(c)

	#marco	P	R	M
0	X	0	0	0
1	X	0	0	0
2	33	1	1	0
3	X	0	0	0

	#marco	P	R	M
4	X	0	0	0
5	X	0	0	0
6	29	1	1	1
7	X	0	0	0

(d)

	#marco	P	R	M
0	X	0	0	0
1	X	0	0	0
2	33	1	1	0
3	29	1	1	0

	#marco	P	R	M
4	X	0	0	0
5	X	0	0	0
6	X	0	0	0
7	X	0	0	0

(e)

Para el proceso B la evolución es similar aunque, al tener 3 marcos asignados en vez de 2, el número de fallos de página disminuye. Inicialmente las tablas de páginas están vacías, como corresponde a un sistema bajo demanda. Los tres primeros accesos, a las páginas 1, 2 y 5 provocan sendos fallos de página y actualizan los bits de presente de las entradas en las tablas de páginas. Los de 2 y 5 ponen a 1 el bit de modificación puesto que son escrituras. Supongamos que el sistema operativo le asigna los marcos 14, 38 y 60. Las tablas se quedan como muestra la figura 2(a). El orden FIFO queda 1–2–5. La escritura en la pagina 6 provoca un fallo de pagina y como no hay marcos libres, el algoritmo de reemplazo FIFO elige como victima a la pagina 1. Como no tenıa activo el bit de modificación no se envía a disco, simplemente se sobrescribe su marco. Las tablas se quedan como muestra la figura 2(b). El orden FIFO queda 2–5–6.

La lectura en la pagina 2 se sirve sin que provoque fallo de pagina, puesto que esa pagina se encuentra en memoria física. El orden FIFO no se altera.

Finalmente la lectura a la pagina 3 provoca un fallo de pagina y el reemplazo de la pagina 2, que se envía a disco, pues el primer acceso a ella fue una escritura y tiene el bit de modificación a 1. Las tablas se quedan como muestra la figura 2(c).

Si la política de reemplazo fuera la de segunda oportunidad, la evolución de las tablas para esta carga particular variaría ligeramente. En la lectura a la pagina 3 se habría elegido como vıctima a la pagina 5. Tras la inserción de la pagina 6, siguiendo la segunda oportunidad, se habrían puesto a cero los bits de referencia de las paginas 2 y 5. Con la lectura de la pagina 2 ese bit se volvería a poner a 1. Al leer la pagina 3, el algoritmo concede una segunda oportunidad a la pagina 2 y elige como vıctima a la 5, que tiene su bit de referencia a 0.

	#marco	P	R	M
0	X	0	0	0
1	14	1	1	0
2	38	1	1	1
3	X	0	0	0

	#marco	P	R	M
4	X	0	0	0
5	60	1	1	1
6	X	0	0	0
7	X	0	0	0

(a)

	#marco	P	R	M
0	X	0	0	0
1	X	0	0	0
2	38	1	1	1
3	X	0	0	0

	#marco	P	R	M
4	X	0	0	0
5	60	1	1	1
6	14	1	1	1
7	X	0	0	0

(b)

	#marco	P	R	M
0	X	0	0	0
1	X	0	0	0
2	X	0	0	0
3	38	1	1	0

	#marco	P	R	M
4	X	0	0	0
5	60	1	1	1
6	14	1	1	1
7	X	0	0	0

(c)

c) Puesto que el reemplazo es local, la planificación de CPU no afecta a la evolución de las tablas de cada proceso. Solo influye en cuando ocurren los fallos de página y los cambios, pero no en cuales son ni en el orden en que suceden. Si en vez de ser dos procesos fueran dos hebras de kernel del mismo proceso el reparto de memoria cambiaría drásticamente. En primer lugar solo habría una única tabla de páginas, en vez de dos. Segundo, al ser hebras del mismo proceso comparten la memoria, por lo cual las páginas que una cargara en memoria la otra ya se las encontraría cuando quisiera acceder a esa zona de memoria. El número de fallos de página sería menor.

Problema 3

Se desea modelar un juego de cartas donde participan 10 jugadores. Los jugadores deberán primero pensar la jugada y luego jugar, pero solo podrán hacerlo cuando les toque su turno (por orden de ubicación en la ronda)

Si el jugador continúa pensando más de un minuto luego de que le toca su turno el árbitro le hará perder su turno y jugará el siguiente participante.

Se dispone de los siguientes procedimientos y funciones:

- **Pensar() : integer** – que ejecutada por el jugador piensa cual será la próxima jugada y retorna un entero que representa dicha jugada
- **Jugar(integer jugada)** - que ejecutada por el jugador, realiza la jugada identificada por el entero pasado como parámetro

Se pide implementar el tipo jugador y el árbitro usando:

a) Semáforos. Se acepta que en caso que el jugador no tenga pensada la jugada al momento de su turno, el árbitro lo espere 1 minuto aunque el jugador termine de pensar antes de terminar dicho lapso.
b) ADA. En este caso no se acepta que el árbitro espere innecesariamente.

Notas:

- No se aceptaran tareas auxiliares.
- La parte a) corresponde a **60%** del ejercicio y la parte b) al **40%**.

Solución:

parte a)

```
#define MINUTO = 60000
Semaphore jugar[10];
Semaphore mutex[10];
Semaphore ya_jugo;

typedef enum {PENSANDO, PRONTO} tipo_estado;
tipo_estado estado[10];

main()
{
 init(ya_jugo, 0);
 for(int i = 0; i < 10; i++)
 {
   init(jugar[i], 1);
   estado[i] == PENSANDO;
 }
```

```
 cobegin
   Arbitro();
   Jugador(0)
   Jugador(1)
     ...
   Jugador(9)
 coend
}


Arbitro()
{
  int i = 0;
  while(TRUE)
  {
     P(mutex[i]);
     if(estado[i] == PRONTO)
     {
          V(jugar[i]);
          P(ya_jugo);
          estado[i] == PENSANDO;
     }
     else
     {
          V(mutex[i]);
          sleep(MINUTO);
          P(mutex[i]);
          if(estado[i] == PRONTO)
          {
               V(jugar[i]);
               P(ya_jugo);
               estado[i] == PENSANDO;
          }
     }
    V(mutex[i]);
    i = (i + 1)%10;
  }
}

Jugador(int i)
{
  while(TRUE)
  {
     mi_jugada = pensar();
     P(mutex[i]);
          estado[i] = PRONTO;
     V(mutex[i]);
     P(jugar[i]);
     jugar(mi_jugada);
     V(ya_jugo);
  }
}
```

---

parte b)

```
TASK arbitro
      ENTRY dame_id(id: OUT INTEGER);
      ENTRY juego[10];
      ENTRY ya_jugo;
END TASK;

TASK BODY arbitro
  INTEGER i;
BEGIN
    FOR i = 1 TO 10 LOOP
      ACCEPT dame_id(id)
            id := i;
      END;
    END FOR;

    i = 1;
    LOOP
      SELECT
            ACCEPT juego[i];
            ACCEPT ya_jugo;
      OR DELAY 60;
      END SELECT
      i = (i + 1) MOD 10;
    ENDLOOP
END TASK


TASK TYPE jugador
END TASK

TASK BODY jugador
  INTEGER i;
  INTEGER mi_jugada;
BEGIN
      Arbitro.dame_id(i);
      LOOP
            Pensar(mi_jugada);
            arbitro.juego[i]();
            Jugar(mi_jugada);
            Arbitro.ya_jugo();
      ENDLOOP
END TASK

jugador jugadores[10]; -- creo 10 jugadores
```
