Examen Diciembre de 2005

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

Para las siguientes partes, conteste justificando brevemente, cada una de las preguntas.

1. En un equipo multiprocesador que dispone de 4 procesadores, se instaló un sistema operativo basado en multiprocesamiento simétrico. ¿Los procesos del sistema operativo pueden ejecutar en cualquiera de ellos, o son restringidos a ser ejecutados en algún procesador específico?

   De estar restringido, describa como se organiza la estructura de los procesadores.

   R: Al ser un sistema operativo simétrico (SMP) los procesos del sistema operativo pueden ejecutar en cualquier procesador sin restricciones.
2. Explicar los sistemas con estructura basada en microkernel.

   R: En una arquitectura basada en microkernel, el sistema operativo se ocupa solo de unas pocas funciones, reduciendo el núcleo a su mínima expresión. El resto de funciones pasan a estar en el espacio de usuario. Se debe de proveer una interfase de comunicación entre los procesos de usuarios y los procesos que proveen los servicios del sistema.
3. ¿Cuáles son las principales actividades del sistema operativo relacionadas con manejo de memoria principal? ¿y cuáles son las relacionadas con el manejo de volúmenes y archivos?

   R:
   - Manejo de Memoria principal
     - Mantener registros de qué partes de la memoria están siendo utilizados y por quién.
     - Decidir qué procesos serán cargados en memoria cuando quede espacio libre.
     - Asignar y quitar espacio de memoria de acuerdo a la necesidad.
   - Manejo de volúmenes y archivos:
     - Definir, eliminar, incorporar volúmenes. Administración de volúmenes.
     - Creación y eliminación de archivos.
     - Creación y eliminación de directorios.
     - Soportar primitivas para la manipulación tanto de archivos como de directorios.
     - Mapeo de archivos en el almacenamiento secundario.
     - Respaldos de archivos en medios de almacenamiento estables (no volátiles).
4. Cuando los procesos realizan pedidos a dispositivos de entrada/salida. Además de la cola de procesos bloqueados, ¿en qué otra cola del sistema son alojados mientras esperan que el dispositivo realice el pedido?

   R: Cada dispositivo de entrada/salida posee de una cola de espera. Un proceso al ejecutar una operación de entrada/salida sobre un dispositivo, es puesto en la cola respectiva al dispositivo.
5. ¿Cuál es el motivo de que en el PCB (Process Control Block) se reserve lugar para los registros de la CPU?

   R: El lugar es reservado debido a que cuando un proceso se le quita el recurso procesador, es necesario guardar su estado de ejecución. De esta forma, cuando es elegido nuevamente para hacer uso del procesador los registros son copiados del PCB a los registros del procesador.
6. En un sistema monoprocesador, que gran ventaja tiene, a nivel de procesos de usuario, un sistema que soporte hilos (threads) a nivel de sistema, frente a uno que no lo haga.

R: Aprovechar que ejecuten otros hilos del mismo procesos mientras alguno de ellos se haya bloqueado en una operación de entrada/salida. Esto es posible ya que el sistema operativo reconoce los hilos como unidades de ejecución independientes.

7. En un sistema con planificador de colas multinivel, en donde a cada nivel se le asigna un algoritmo de planificación round-robin con diferente quantum y teniendo mayor prioridad las colas con quantum más chico, ¿qué tipos de procesos (alto consumo CPU o alto consumo de operaciones de E/S) clasificarían para colas con quantum más bajo y cuáles otros para colas con quantum más alto.

   R: La prioridad para el uso de un recurso es inversamente proporcional al uso que se hace del mismo. Por lo tanto, los proceso que tengan un alto consumo de CPU deberán ir a colas de menor prioridad (según la letra, quantum mayor), mientras que los procesos que hagan mayor uso de E/S tendrán mayor prioridad de acceso al procesador (colas con quatum menor). Cuando el planificador seleccione un proceso de alto consumo de CPU (una cola de baja prioridad), este ejecutará hasta que no aparezca un proceso de alto consumo de E/S (que seguramente demore poco en la CPU). De esta forma, se logra que los procesos que requieran poca CPU ejecuten lo antes posible y liberen el recurso CPU rápidamente de forma de tener un mayor desempeño general del sistema.
8. A nivel del sistema operativo, ¿qué permite la instrucción SWAP provista por el hardware?

   R: Poder implementar la sincronización entre procesos, debido a la atomicidad que presenta esta instrucción.
9. ¿Qué entiende por carga dinámica (dynamic loading)? Explique el concepto y sus beneficios.

   R: El dynamic loading permite que una rutina no sea cargada en memoria principal hasta tanto no se utilice. Entre los beneficios se puede mencionar que no se cargan innecesariamente rutinas que no son invocadas. A su vez, los binarios en el sistema ocupan menos espacio ya que se pueden crear bibliotecas dinámicas que varios programas referencien.
10. En un sistema con segmentación de 4096 bytes de memoria física, se tienen asignados los siguientes segmentos:

Base	Largo (bytes)
100	150
300	1024
40	20
2600	512
1358	128

    ¿Cuál sería la asignación en memoria física de los tamaños de segmentos que aparecen en la lista de abajo, según las estrategias de asignación First-fit, Best-fit y Worst-fit? Asuma que los pedidos de segmentos llegan en orden de izquierda a derecha, la memoria empieza en la dirección 0 y los segmentos que no pueda ubicar son descartados.
    Lista de segmentos nuevos: 128, 40, 1024, 32, 40, 45.
    R: (base, largo)
    First Fit: (1486,128), (0,40), descartado 1024, (60,32), (250,40), (1614, 45)

    Best Fit: (3112,128), (0,40), (1486,1024), (1324,32), (60,40), (250, 45)

    Worst Fit: (1486, 128), (1614, 40), descartado 1024, (3112,32), (3144,40), (1654, 45)
11. En un sistema con memoria virtual, ¿cuál es la función del dispositivo de hardware MMU (memory-management unit)?

    R: Realizar la traducción de direcciones de memoria virtuales a físicas.

Problema 2

Se tiene desarrollada una aplicación que realiza tres procedimientos cuya ejecución es independiente entre sí. Los prototipos de estos procedimientos son los siguientes:

- `void Procedimiento1(void);`
- `void Procedimiento2(void);`
- `void Procedimiento3(void);`

Suponga que los procedimientos tienen las siguientes fases de ejecución y que se usa E/S por DMA:

- Procedimiento 1: 25 ms de CPU, lectura de dispositivo1 25 ms, 25 ms de CPU
- Procedimiento 2: 30 ms de CPU, lectura de dispositivo1 35 ms, 20 ms de CPU
- Procedimiento 3: 50 ms de CPU

Esta aplicación se ha desarrollado utilizando dos modelos diferentes, cuyo código se detalla a continuación:

```
/* Modelo 1 – Un único proceso */
1 void main (void)
2 {
3      Procedimiento1();
4      Procedimiento2();
5      Procedimiento3();
6      exit(0);
7 }
```

```
/* Modelo 2 – Tres procesos concurrentes */
1 void main (void)
2 {
3      if (fork()!=0)
4      {
5             if (fork()!=0)
6                    Procedimiento1();
7             else
8                    Procedimiento3();
9      }
10     else
11     {
12            Procedimiento2();
13     }
14     wait(NULL);
15     exit(0);
16 }
```

Se consideran los siguientes supuestos:

- El sistema operativo emplea un algoritmo de planificación basado en prioridades (en caso de que existan varios procesos listos para ejecutar, se seleccionará el más prioritario). En el caso del Modelo 2 el proceso padre tiene prioridad 1, el primer proceso hijo prioridad 2 y el segundo proceso hijo prioridad 3 (números más bajos indican mayores prioridades).
- Se tomará como t=0 el instante en el que comienza la ejecución de la línea 3 en el Modelo 1 y la ejecución de la línea 3 en el Modelo 2.
- El SO no ejecuta ningún otro proceso y no se tendrá en cuenta ningún tipo de interrupción de reloj.
- El tiempo de ejecución de los servicios del sistema operativo es el siguiente (estos tiempos incluyen la planificación y activación del siguiente proceso):
    - Ejecución de la llamada al sistema fork: 20 ms
    - Ejecución de la llamada al sistema para la lectura de dispositivo: 10 ms
    - Ejecución de cualquier otra llamada al sistema: 10 ms

  - Ejecución de la rutina que atiende la interrupción del fin de la operación de E/S: 15 ms
- Considere cualquier otro tiempo despreciable.

Se pide resolver las siguientes cuestiones, justificando adecuadamente cada respuesta:

a) ¿Qué tiempo total tarda en ejecutar el Modelo 1?
b) ¿Qué árbol de procesos surge de la ejecución del Modelo 2? Indique que proceso llama a cada procedimiento.
c) ¿Qué sucede si estando en ejecución un proceso menos prioritario termina una operación de E/S de un proceso más prioritario?
d) En el Modelo 2, ¿quién está ejecutando en el instante 110 ms?
e) ¿Qué tiempo total tarda en ejecutar el Modelo 2?
f) En el Modelo 2, ¿qué pasa con la invocación a la llamada wait() de la línea 14 en el segundo proceso hijo creado?

Solución:

a) El tiempo total corresponde a la suma de los tiempos consumidos por los procedimientos indicados, mas el tiempo utilizado por el sistema operativo para realizar las llamadas al sistema y las atenciones de rutinas de interrupción necesarias.

C = CPU
D = ejecución de la E/S
SL = llamada al sistema para lectura de dispositivo
SI = llamada al sistema para atender rutina de interrupción
SF = llamada al sistema para fork
S = otras llamadas al sistema llamada

Los tiempos se indican en milisegundos

P1: 25C + **10SL** + 25D + **15SI** + 25C = 100ms
P2: 30C + **10SL** + 35D + **15SI** + 20C = 110ms
P3: 50ms

En el caso del modelo 1, solo debemos sumar los tiempos, mas diez milisegundos de la llamada a exit(0) (Asumimos que es una llamada al sistema)

Modelo1 = 100ms + 110ms + 50ms = 260ms

b) Árbol de procesos

```
pA (main – proc 1)
 |-----> pB (proc 2)
 |
 *-----> pC (proc 3)
```

c) Lo que suceda dependerá si el modelo de planificación es expropiativo o no. **En el caso expropiativo**, al ejecutar la rutina de interrupción para atender el fin de entrada salida, el proceso que estaba ejecutando es retirado del procesador para ejecutar la rutina de interrupción (es más prioritaria). Luego de esto, si el proceso que fue desalojado esta listo para ejecutar (seguramente lo este) deberá competir con el que acaba de terminar su E/S, el cual si tiene mayor prioridad y esta listo para ejecutar, será el que reciba el procesador

a continuación. **Por el contrario, en el caso no expropiativo,** el que continúa la ejecución es el proceso interrumpido. Una vez que se finalice la ejecución de este, se realizara una planificación basada en prioridades, para determinar cual es el proceso que deberá continuar su ejecución.

d) esta ejecutando pA (procedimiento 2) (justificación en parte e)

e) Ordenamos la ejecución de los procesos en el tiempo según la operación realizada

Caso expropiativo

Tiempo (ms)	Proceso A (pro1)	Proceso B (pro2)	Proceso C(pro3)
0	**(sc)fork**		
5	**(sc)fork**		
10	**(sc)fork**		
15	**(sc)fork**		
20	**(sc)fork**		
25	**(sc)fork**		
30	**(sc)fork**		
35	**(sc)fork**		
40	**CPU**		
45	**CPU**		
50	**CPU**		
55	**CPU**		
60	**CPU**		
65	**(sc)inicio i/o**		
70	**(sc)inicio i/o**		
75	i/o	**CPU**	
80	i/o	**CPU**	
85	i/o	**CPU**	
90	i/o	**CPU**	
95	i/o	**CPU**	
100	**(sc)fin i/o**		
105	**(sc)fin i/o**		
110	**(sc)fin i/o**		
115	**CPU**		
120	**CPU**		
125	**CPU**		
130	**CPU**		
135	**CPU**		
140	**(sc)wait**		
145	**(sc)wait**		
150		**CPU**	
155		**(sc)inicio i/o**	
160		**(sc)inicio i/o**	
165		i/o	**CPU**
170		i/o	**CPU**
175		i/o	**CPU**
180		i/o	**CPU**
185		i/o	**CPU**
190		i/o	**CPU**
195		i/o	**CPU**
200		**(sc)fin i/o**	
205		**(sc)fin i/o**	
210		**(sc)fin i/o**	
215		**CPU**	
220		**CPU**	
225		**CPU**	
230		**CPU**	
235		**(sc)wait**	

240		**(sc)wait**	
245			**CPU**
250			**CPU**
255			**CPU**
260			**(sc)wait**
265			**(sc)wait**
270			**(sc)exit**
275			**(sc)exit**
280		**(sc)exit**	
285		**(sc)exit**	
290	**(sc)exit**		
295	**(sc)exit**		

El tiempo total de ejecución del modelo 2 es de 295ms

Caso no expropiativo

Tiempo (ms)	Proceso A (pro1)	Proceso B (pro2)	Proceso C(pro3)
0	**(sc)fork**		
5	**(sc)fork**		
10	**(sc)fork**		
15	**(sc)fork**		
20	**(sc)fork**		
25	**(sc)fork**		
30	**(sc)fork**		
35	**(sc)fork**		
40	**CPU**		
45	**CPU**		
50	**CPU**		
55	**CPU**		
60	**CPU**		
65	**(sc)inicio i/o**		
70	**(sc)inicio i/o**		
75	i/o	**CPU**	
80	i/o	**CPU**	
85	i/o	**CPU**	
90	i/o	**CPU**	
95	i/o	**CPU**	
100	**(sc)fin i/o**		
105	**(sc)fin i/o**		
110	**(sc)fin i/o**		
115		**CPU**	
120		**(sc)inicio i/o**	
125		**(sc)inicio i/o**	
130	**CPU**	i/o	
135	**CPU**	i/o	
140	**CPU**	i/o	
145	**CPU**	i/o	
150	**CPU**	i/o	
155	**(sc)wait**	i/o	
160	**(sc)wait**	i/o	
165		**(sc)fin i/o**	
170		**(sc)fin i/o**	
175		**(sc)fin i/o**	
180		**CPU**	
185		**CPU**	
190		**CPU**	
195		**CPU**	
200		**(sc)wait**	
205		**(sc)wait**	
210			**CPU**

215			**CPU**
220			**CPU**
225			**CPU**
230			**CPU**
235			**CPU**
240			**CPU**
245			**CPU**
250			**CPU**
255			**CPU**
260			**(sc)wait**
265			**(sc)wait**
270			**(sc)exit**
275			**(sc)exit**
280		**(sc)exit**	
285		**(sc)exit**	
290	**(sc)exit**		
295	**(sc)exit**		

El tiempo total de ejecución del modelo 2 es de 295ms

f) Al ejecutar el wait, esta llamada al sistema espera por la terminación de algún hijo del proceso B. Como el parámetro pasado es NULL, no tendremos información del estado de terminación del proceso hijo.

Problema 3

Se desea modelar en Ada el comedor de una empresa.

En el mismo hay una mesa con capacidad para 12 comensales (empleados y clientes de la empresa).

El comedor dispone de 2 microondas que usan los comensales para calentar su comida. Nadie puede quedarse esperando por un microondas si hay uno libre.

El comensal se asegurará que haya lugar en la mesa de forma que no se le enfríe la comida luego de haberla calentado. A los comensales no se les debe enfriar la comida.

La empresa dispone de una limpiadora, que limpiará, lo más pronto posible, el lugar del comensal que se retira de la mesa.

Los comensales jamás se sientan en un lugar que no fue higienizado previamente.

Se dispone de las siguientes funciones y procedimientos:

- **Tiempo_calentar()**: devuelve la cantidad de segundos que el comensal desea calentar su comida. Debe ser ejecutado por el comensal.
- **Calentar (x)**: Calienta la comida del microondas durante x segundos. Debe ser ejecutado por el microondas.
- **Limpiar()**: Limpia un lugar de la mesa. Debe ser ejecutado por la limpiadora.
- **Comer()**: Come la comida. Debe ser ejecutado por el comensal.
- **Sentarse()**: Ubica al comenzal en un sitio en las condiciones necesarias para comenzar a comer.

Se desea modelar el comensal, cada uno de los microondas, la mesa y la limpiadora.

Notas:

- En caso de necesitarse se puede usar una tarea auxiliar como máximo.

Solución:

```
micros = array [0..1] of task microondas;

task microondas is
  entry usar(t: in integer)
  entry numero(x: in integer)
end microondas

task body microondas is
  soy: integer;

begin
  accept numero(i: in integer)
     soy = i;
  end

  loop
     accept usar(t: in integer)
        calentar(t);
     end
     adm_micro.liberar(soy);
  end
end microondas


task adm_micro is
  entry micro_libre(x: out integer)
  entry liberar (x: in integer)
end adm_micro

task body adm_micro is
  cnt_ocup, i: integer
  libres: array(0..1) of boolean

begin
  for i in 0..1 loop
     micros[i].numero(i);
      libres[i] = true;
  end
  cnt_ocup = 0;

  loop
    select
        when cnt_ocup < 2 =>
           accept micro_libre(x: out integer)
               -- Libre devuelve el indice del primer micro libre
               i = libre(libres);
               x = i;
           end
        libres[i] = false;
           cnt_ocup = cnt_ocup + 1;
     or
           accept liberar(x: in integer)
               i=x;
           end
           libres[i] = true;
           cnt_ocup = cnt_ocup - 1;
```

```
    end
  end
end adm_micro


task comensal is
end comensal

task body comensal is
  m,t: integer;

begin
  mesa.hay_lugar();
  adm_micro.micro_libre(m);
  t = tiempo_calentar();
  micros[m].usar(t);
  sentarse();
  comer();
  mesa.dejar_lugar();
end comensal


task mesa is
  entry hay_lugar()
  entry dejar_lugar()
  entry lugar_limpiar()
  entry lugar_limpio()
end mesa

task body mesa is
  lugares,sucios: integer

  lugares = 12;
  sucios = 0;

begin
  loop
      select
        when lugares > 0 =>
           accept hay_lugar();
           lugares = lugares - 1;
     or
           when sucios > 0 =>
           accept lugar_limpiar();
     or
           accept lugar_limpio();
           sucios = sucios - 1;
           lugares = lugares + 1;
     or
           accept dejar_lugar();
           sucios = sucios + 1;
      end
  end
end mesa


task limpiadora is
end limpiadora

task body limpiadora is

begin
  loop
    hacer_su_vida();
```

```
    mesa.lugar_limpiar();
    limpiar();
    mesa.lugar_limpio();
  end
end limpiadora
```