Examen Marzo 2007

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

Justifique brevemente cada respuesta.

1) Dado el siguiente código escrito en lenguaje C:

```
main ()
{
    char * buffer;
    int fd_read, fd_write, count;
    buffer = (char *) malloc(256*sizeof(char));

    if ((fd_read = openread("read.txt") == -1) || (fd_write =
openwrite("write.txt") == -1))
        exit 1;

    while (count = read(fd_read,buffer,256))
        write(fd_write,buffer,count);

    closeread(fd_read);
    closewrite(fd_write);
    exit 0;
}
```

   a. ¿Cuáles invocaciones a procedimientos son un llamado a sistema (system call) o desembocan en uno.
   b. Cual es el propósito de un llamado a sistema.

2) Describa los sistemas operativos diseñados con un micro-núcleo (*microkernel*).
3) Describa el planificador de colas multinivel con feedback (*Multilevel-Feedback-Queue*).
4)
   a. ¿Qué es el tiempo de retorno (*turnaround time*)?
   b. ¿Qué entiende por rendimiento (*throughput*) de un algoritmo de planificación?
5) ¿El siguiente sistema está en *deadlock*?

Recurso	R1	R2	R3	R4
Cantidad	1	2	1	3

Procesos	P1	P2	P3
Recursos Asignados	R2	R1,R2	R3

Procesos	P1	P2	P3
Recursos Pedidos	R1	R3	R2

6) En un sistema computacional que maneja memoria virtual con direcciones de 16 bits:
   a. ¿Cuál es el tamaño del espacio virtual de los procesos en este sistema?
   b. Si se tienen un sistema de paginación de dos niveles, y las direcciones virtuales se forman con 3 bits para el primer nivel, 3 bits para el segundo nivel y 10 bits para el desplazamiento. ¿Cuántas tablas de páginas se tienen en memoria, de primer y segundo nivel, si el proceso está consumiendo 10KB de espacio virtual?
7) Compare la asignación de *frames* global vs local.
8) Describa dos métodos de acceso a un archivo.
9)
   a. ¿Qué entiende por el método polling?
   b. Describa el método de acceso directo (*Direct Memory Access*).
10)
   a. ¿Qué entiende por memoria cache?
   b. ¿Qué entiende por *Spooling*?

Solución:

Nota: Esta solución tiene respuestas que son una guía de solución. No necesariamente están completas.

1.

a. Todos ya que requieren de servicios que brinda el sistema operativo:

malloc: Es una función de la biblioteca de usuario que puede desenbocar en la extensión del área de heap del proceso.

openread y openwrite: apertura de un archivo.

read y write: Lectura y escritura de un archivo.

closeread, closewrite: cerrar un archivo abierto por el proceso.

exit: Finalización del proceso.

b. Brindar un servicio del sistema operativo.

2.

Los sistemas con micronúcleo brindan funcionalidades de gestión de procesos y memoria. A su vez, proveen de una capa de comunicación entre procesos para poder implementar los demás servicios a nivel de usuario.

3.

Es un planificador que define un conjunto de colas a las que clasifica según algún criterio (ej. Prioridad). Los procesos serán asignados (según su característica) a alguna de las colas. Cada cola puede tener su política individual de planificación.

La retroalimentación permite que los procesos sean cambiados de cola cada vez que adquieran el estado de listo. Ej.: Un proceso con alto consumo de CPU (cpu-bound) tenderá a ir a colas de menor prioridad. Los procesos con bajo consumo de CPU tenderán a avanzar a colas con mayor prioridad.

4.

a. Es el intervalo de tiempo desde que un proceso es cargado hasta que finaliza su ejecución.

b. Es el número de procesos que ejecutaron completamente por unidad de tiempo.

5.

Sí, está en deadlock.

Falta justificación.

6.

a. 2^{16}

b. Suponiendo que la memoria virtual asignada comienza en la dirección virtual 0 y termina en la dirección virtual 10k. Para representar el espacio virtual del proceso es necesario 3 tablas (1 para primer nivel y 2 para segundo nivel).

7.

En la estrategia global los fallos de página de un proceso afectan la eficiencia de ejecución de otro.

La estrategia local independiza los fallos de página por proceso, pero ocupa ***frames*** de memoria que quizás sean poco utilizados y no son reemplazados.

8.

Secuencial.

Directo.

Falta descripción.

9.

a. Es cuando el sistema consulta constantemente el busy bit de la controladora del dispositivo. El motivo es verificar que la controladora quedo disponible para aceptar otro comando. Este sistema genera busy waiting, pero puede ser beneficioso para sistemas dedicados.

b. El DMA es un dispositivo que permite la transferencia de bloques de memoria desde un controlador a memoria RAM. El sistema carga en los registros de la controladora DMA la tarea a realizar y posteriormente la activa. La controladora DMA se encarga de controlar la transferencia desde/hacia el controlador de dispositivo hacia/desde la memoria principal sin intervención del procesador. Una vez finalizada la transferencia se interrumpe al procesador para avisar al sistema de la finalización.

10.

a. La cache es una memoria de rápido acceso que permite tener copias de datos que se encuentran en dispositivos más lentos. De esta forma, el dispositivo o sistema que accede a los datos tiene mejor velocidad de acceso.

b. El Spooling es un buffer que permite guardar información mientras son transferidos entre dos dispositivos o un dispositivo y una aplicación. El dispositivo que recibe la información no permite que sea intercalado.

Problema 2

Supóngase un sistema tipo UNIX con un sistema de archivos cuyos i-nodos contienen, además de los atributos, los punteros con información de la ubicación de los bloques de datos. Esta información tiene 10 accesos directos, 1 puntero de indirección simple, un puntero de indirección doble y 1 puntero de indirección triple. Además los bloques de este sistema de archivos tienen de tamaño 4KB. Por otro lado considérese el siguiente programa escrito en lenguaje C que ejecuta en dicho sistema UNIX:

```
main() {
   int fd1, pid1, pid2;
   char buf[10];
   fd1 = open("/usr/a", O_RDWR);
   /* ejecuto sentencias durante 20ms */
   pid1 = fork;
   if (pid1 == 0){ /* hijo 1 */
      /* ejecuto sentencias durante 20 ms */
      read(fd1, buf, 10);
      /* ejecuto sentencias durante 10 ms */
      exit();
   } else {
      pid2 = fork;
      if (pid2 == 0) { /* hijo 2 */
         /* ejecuto sentencias durante 10 ms */
         read(fd1, buf, 10);
         /* ejecuto sentencias durante 10 ms */
         exit();
      } else { /* padre */
         /* ejecuto durante 10 ms */
         exit();
      }
   }
}
```

Se pide:

a) ¿Cuántos bloques de datos podría tener como máximo un archivo de este sistema, sabiendo que los índices de direccionamiento de bloques de datos tienen 16 bits de tamaño?
b) ¿Cuántos accesos a disco supone las operaciones open del código anterior?
c) Suponiendo que tenemos un planificador de procesos con algoritmo de planificación tipo Round-Robin y con un time-slice de 10 ms, se pide dibujar el cronograma de la ejecución de los procesos del programa descrito en el enunciado, suponiendo que el sistema de archivos tiene una cache en memoria principal con los bloques mas recientemente usados que estará vacía antes de la ejecución del programa y que cada acceso a disco tarda 10 ms.

Nota: El cronograma de ejecución debe indicar para cada proceso su estado de E (ejecutando), L (listo para ejecutar) y B (bloqueado por E/S) durante toda la ejecución del mismo.

Solución:

a) Por un lado el enunciado nos dice que el tama˜no de las direcciones a los bloque de datos del disco ocupan 16 bits. Esto significa que se pueden direccionar como maximo hasta 2**16 = 64K bloques de datos. Pero por otro lado tendremos que calcular si la estructura de nodo-i del sistema de archivos impone o no mas restricciones a este numero maximo. Veamos cuantos bloques de datos pueden ser direccionados desde un nodo-i como el descrito en el enunciado. Primero tenemos que calcular cuantas direcciones de bloques de datos caben en un bloque de datos. Como los bloques de datos ocupan 4koctetos y las direcciones ocupan 2 octectos, entonces el numero de direciones de bloques de datos que caben en un bloque de datos es de 2K.

Calculemos ahora los bloques de datos que se pueden direccionar desde un nodo-i.

Con los 10 enlaces directos podemos direccionar 10 bloques. Con el enlace de indireccion simple podemos direccionar 1 bloque que contiene 2k direcciones a bloques. Con el enlace de indireccion doble podemos direccionar 1 bloque que contendra 2k direcciones de bloques conteniendo cada uno 2k direcciones a bloques de datos, esto es, 2k*2k=4M bloques de datos. Por ultimo con el enlace de indireccion triple podemos direccionar 1 bloque que contendra 2k direcciones de bloques conteniendo cada uno 2k direcciones a bloques que a su vez contendran cada uno de ellos 2k direcciones a bloques de datos, esto es, 2K*2K*2K=8G bloques de datos. Como se puede ver los nodos-i nos permitiria direccionar si pudiera muchos mas bloques que los impuestos por el propio tama˜no de la direccion que como hemos visto solo nos permite direccionar 64K bloques de datos. Por tanto, la respuesta a esta pregunta es 64k bloques de datos.

b) La operacion del codigo open("/usr/a",O_RDWR) tiene como objetivo traer a memoria principal el nodo-i del archivo "/usr/a". Si suponemos que el nodo-i del directorio "/" ya esta en memoria principal el numero de accesos al disco necesarios sera:

   a. 1 acceso para traer a memoria principal el primer bloque de datos del directorio "/" cuya direccion buscaremos en el nodo-i del directorio "/".
   b. 1 acceso para traer a memoria principal el nodo-i del directorio "/usr" cuya direccion en disco buscaremos en el bloque de datos del directorio "/".
   c. 1 acceso para traer a memoria principal el primer bloque de datos del directrio "/usr" cuya direccion buscaremos en el nodo-i del directorio "/usr".
   d. 1 acceso para traer a memoria principal el nodo-i del archivo "/usr/a" cuya direccion en disco buscaremos en el bloque de datos del directorio "/usr"

   Por tanto el numero de acceso a disco sera de 4.

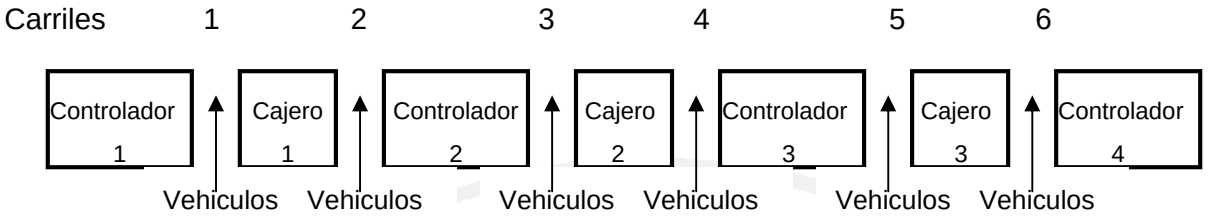
c) El calendario de ejecución queda como:

```
        0  10  20  30  40  50  60  70  80  90  100 110 120 130 140
Tiempo  |---|---|---|---|---|---|---|---|---|---|---|---|---|---|
Padre   |*B*|*B*|*B*|*B*|-E-|-E-|+L+|+L+|-E-|^T^|^T^|^T^|^T^|^T^|
Hijo1   |.......................|-E-|+L+|+L+|-E-|+L+|-E-|^T^|^T^|
Hijo2   |.......................|+L+|-E-|*B*|+L+|-E-|^T^|^T^|^T^|
```

Primero el padre se bloquea (*B*) durante 40ms haciendo la operacion open, luego ejecuta (-E-) durante 20 ms, en el instante 60 crea los dos procesos hijo. Por round robin ejecuta primero el proceso hijo1 y luego el proceso hijo2, mientras los otros quedan en estado listo para ejectuar (+L+). En el instante 80 el proceso hijo2 se bloquea haciendo E/S. En ese mismo instante el proceso padre recupera la CPU, ejecuta y luego termina (^T^). En el instante 90 le toca el "time slice" al proceso hijo1 que tendrıa que bloquearse para hacer e/s pero como se encuentra en la cache (memoria principal) el bloque a leer, no tiene que bloquearse ejecutando en cambio duarante 10 ms. Luego le toca el turno al proceso hijo2 que despues de ejecutar termina y finalmente ya solo queda el proceso hijo1 en el instante 110 que ejecuta y luego termina.

Problema 3

Se desea modelar un peaje de una carretera de una sola vía. El peaje además de cobrar el ticket controla los documentos de los autos y camiones. Los puestos están distribuidos de esta manera:



Las cajas y los controladores atenderán para los dos carriles que tienen a su costado, salvo las de las puntas que solamente atienden autos del carril que tienen al lado.

Una vez que un cajero atendió a un vehículo, no podrá atender a otro hasta que el controlador termine de controlar dicho vehículo, y viceversa.

Por otro lado los camiones solo podrán pasar por los carriles de las puntas. Un inspector controlara cada cierto tiempo a estos controladores (los de las puntas), los que no podrán atender ningún vehículo mientras están siendo controlados y viceversa.

El control por parte del inspector a estos controladores se hará con los 2 a la vez, es decir que ninguno de los 2 deberá estar controlando vehículos, debiendo obtener primero la atención del controlador 4 antes que el controlador 1.

Por razones de ordenamiento y comodidad, los vehículos siempre pedirán ser atendidos primero por el puesto de su izquierda y luego por el de la derecha.

Se pide:

Modelar este sistema usando monitores.

Se dispone de las siguientes funciones o procedimientos:

- **pagar()** paga o cobra al vehiculo al cajero.
- **controlar_documentacion()** control de documentación del controlador al vehículo.
- **inspeccion()** control del inspector a los controladores de las puntas.
- **dame_carril(tipo_vehículo: Integer)** que retorna a que carril debe ir el vehículo, dependiendo si es auto o camión, el cual será ejecutado por el vehículo.
- **que_soy()** retorna 0 si es un auto o 1 si es un camión.

Nota: Se podrá utilizar como máximo una tarea auxiliar.

Solución:

La solución a este problema es equivalente a la solución vista en el curso sobre el problema de los 'Filósofos Comensales' pero con 6 comensales en lugar de 5 como se vio en el curso. En este caso los vehículos (y el inspector) toman el lugar de los filósofos y los controladores y cajeros toman el lugar de los cubiertos. Para armar la estructura 'circular' debe considerarse que los inspectores obtienen primero el controlador de más a la derecha y luego el de la izquierda.

```
Procedure Vehiculo
   int carril = dame_carril(que_soy()); // Devuelve un numero entre 1 y 6
   int controlador, cajero;
   int controlador = carril DIV 2;
   int cajero = (carril-1) DIV 2;
   Administrador.entrar();
   if(controlador == cajero) // Si el controlador esta a la izquierda
         Controlador.inicioControl(controlador);
         Cajero.inicioPago(cajero);
   else // Si el cajero esta a la izquierda
         Cajero.inicioPago(cajero);
         Controlador.inicioControl(controlador);
   endif;
   pagar();
   controlar_documentacion();
   if(controlador == cajero)
         Cajero.finPago(cajero);
         Controlador.finControl(controlador);
   else
         Controlador.finControl(controlador);
         Cajero.finPago(cajero);
   endif;
   Administrador.salir();
End;
```

```
Procedure Inspector
   Administrador.entrar();
   Controlador.inicioControl(3);
   Controlador.inicioControl(0);
   inspeccionar();
   Controlador.finControl(0);
   Controlador.finControl(3);
   Administrador.salir();
End;

Monitor Controlador
   boolean ocupada[0..3];
   Condition espera[0..3];

   procedure inicioControl(int i)
         if ocupada[i]
               espera[i].wait();
         endif
         ocupada[i] = true;
   end;

   procedure finControl(int i)
         ocupada[i] = false;
         espera[i].signal();
   end;

   begin
         for i = 0 to 3
               ocupada[i] = false;
         endfor
   end
end Controlador;
```

```
Monitor Cajero
   boolean ocupada[0..2];
   Condition espera[0..2];

   procedure inicioPago(int i)
         if ocupada[i]
               espera[i].wait();
         endif
         ocupada[i] = true;
   end;

   procedure finPago(int i)
         ocupada[i] = false;
         espera[i].signal();
   end;

   begin
         for i = 0 to 2
               ocupada[i] = false;
         endfor
   end
end Cajero;

Monitor Administrador
   Condition espera;
   int adentro;

   procedure entrar()
         adentro++;
         if adentro > 5
               espera.wait();
         endif
   end
```

```
    procedure salir()
          adentro--;
          if adentro >= 5
                espera.signal();
          endif
    end

    begin
          adentro = 0;
    end;
end Administrador;

main()
begin
    cobegin
          // Lanzar vehiculos, camiones e inspector
    coend
end
```