Examen Febrero de 2004

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 muy brevemente cada una de las preguntas.

1. Clasificación de Sistemas Operativos

a) Defina sistema multiprocesador simétrico y asimétrico.

b) Describa brevemente: sistemas *batch*, sistemas de tiempo compartido, sistemas de tiempo real y sistemas distribuidos.

2. Concurrencia

a) Mencione los tres requisitos que debe cumplir una solución al problema de la sección crítica.

b) En el contexto de monitores, cuando un proceso ejecuta un *signal* sobre una condición, y hay un proceso esperándola, ¿qué debe resolverse y qué políticas pueden tomarse para ello?

c) ¿Cuándo puede decirse que las condiciones necesarias de *deadlock* también son suficientes?

3. Memoria

a) Presente las estrategias de asignación dinámica de almacenamiento.

b) De tres ventajas de la segmentación que no tenga la paginación.

c) Dado el algoritmo de reemplazo de páginas FIFO, diga qué sucede al incrementarse el número de marcos asignados.

d) Defina algoritmos de pila para reemplazo de páginas. Mencione un algoritmo de pila dado en el curso.

e) ¿Cuál es la ventaja de los algoritmos de pila?

f) En un sistema de paginación por demanda, si el hardware lo permite ¿qué utilidad tiene el bit sucio (*dirty bit*)?

g) Explique qué desventaja particular tienen los hilos implementados a nivel de usuario (sin soporte del núcleo) en un sistema de memoria virtual con paginación por demanda.

h) ¿Qué estrategia evita la hiperpaginación manteniendo a la vez el nivel de multiprogramación lo más alto posible?

4. Disco y sistemas de archivos

a) Describa el algoritmo SCAN.

b) ¿Qué desventaja presenta el algoritmo SCAN? Dé una alternativa.

c) Presente brevemente tres alternativas para gestión de espacio libre en almacenamiento secundario.

d) ¿Qué inconveniente tienen los sistemas que apoyan distintos tipos de archivos en forma nativa?

```
Solución:
      Ver libro
```

Problema 2

a) Especificar los componentes del bloque de control de procesos típico

b) Indicar la secuencia de pasos que sigue el sistema operativo al realizar un cambio de contexto entre dos procesos, teniendo en cuenta la estructura definida con anterioridad.

c) Se pide implementar un planificador/despachador de procesos para un sistema monoprocesador multitarea que cumpla con las siguientes características:

- Utilice una planificación basada en colas multinivel y retroalimentación
- Los procesos tienen asociada una prioridad discreta, del 0 al 10, siendo la prioridad 0 la mayor del sistema.
- El sistema cuenta con una mezcla variada de procesos interactivos y batch. Los procesos batch siempre entran al sistema con proridad 10.
- Nunca debe detenerse o retrasarse la ejecución de un proceso con prioridad Q, si solo existen en el sistema procesos con prioridad P, donde $P < Q$
- Se desean asignar cuantos de tiempo distintos según la prioridad de los procesos. A menor prioridad, mayor el cuanto de tiempo asignado (deben respetarse las restricciones anteriores)

d) Qué cambios debería realizar al planificador/despachador anterior, si el hardware donde debe implementarse es un sistema multiprocesador asimétrico, en donde un procesador es el encargado de ejecutar permanentemente el código del planificador.

```
Solución:

a) El Process Control Block de un proceso, contiene informacion critica para la
administracion del mismo dentro del SO. Entre otras cosas, encontramos:

      o Identificacion (Id unico de proceso)
      o Estado del proceso (Pronto, Listo, Suspendido, ...)
      o Registros, Program Counter
      o Regiones de memoria asignadas
      o Lista de archivos abiertos, informacion de comunicacion entre procesos
      o Informacion de contabilidad (tiempo de uso, recursos consumidos)
      o Punteros a otras estructuras del SO
```

b) Frente a un cambio de contexto, el sistema debe:

- Guardar la informacion asignada del proceso en memoria (o almacenamiento secundario)
- Descargar el stack y los registros
- Cargar desde memoria o almacenamiento secundario, la informacion del proceso entrante. Esto es, los archivos abiertos, registros, program counter, etc.

c) Para implementar el planificador despachador, construimos primero una estructura formada por:

- 11 colas de procesos, donde cada una esta asociada a una prioridad especifica (0 a 10), siendo 10 la cola batch
- Cuando un proceso entra al sistema, el despachador lo coloca en la cola con la prioridad asociada.
- Se implementa un round robin entre colas, de manera tal que un proceso de una cola ejecuta, si las colas de mayor prioridad no estan con elementos
- Los cuantos de tiempo son menores a mayor prioridad (menor numero)
- En el caso de los procesos batch, aplicamos un algoritmo FIFO

Tendremos entonces las funcionalidades:

```
// Llega un proceso al sistema
procedure iniciaProceso(PCB p)
begin
     p.estado = PRONTO;
     cola(p.PRIORIDAD).insert(p);
     planificador();
end;

// Interrumpir proceso
procedure interrumpirProceso(PCB p)
begin
     p.estado = SUSPENDIDO;
     // Sacamos el proceso del ppo y lo metemos al final de la cola
     cola(p.PRIORIDAD).insert(p);
     cola(p.PRIORIDAD).insert(p);
     planificador();
end;
```

```
// Llega un proceso al sistema
procedure finalizaProceso(PCB p)
begin
     cola(p.PRIORIDAD).remove(p);
     planificador();
end;

// Se ejecuta el planificador, verficando que hay que correr.
// Bajo un evento de planificacion, se ejecuta este metodo
procedure planificador()
begin
     for i = 0 to 9 do
          q = cola(i);
          if not vacia(q) then
               for j = 0 to size(q) do
                    PCB pj = q(j);
                    if pj.estado = PRONTO then
                              // Cambiamos el contexto a este proceso, y finalizamos la
ejecucion
                         cambiarContextoA(pj);
                         return;
                    endif
               endfor
          endif
     endfor

     // Corremos el primer proceso de la cola batch (si hay)
     if size(cola(10)) then
         PCB pb = cola(10)(0);
         cambiarContextoA(pb);
     endif
end;
```

Problema 3

Se desea modelar un servicio de fotocopiadora de leyes en el cual los clientes piden una ley que luego fotocopiarán y devolverán inmediatamente. En el local donde se presta el servicio no pueden entrar más de 25 clientes a la vez y sólo existe una copia de cada ley para prestar las cuales se piden por su número de ley. La cantidad de leyes no es acotada.

NOTAS:

- No se podrán implementar tareas auxiliares
- Las tareas clientes se crean dinámicamente sin poderse identificar dentro de un array de tareas.
- Se dispone de la función fotocopiar que deberá ser ejecutada por el cliente.
- Se deberá prestar especial atención a no "despertar/desbloquear" tareas innecesariamente.

a) Modelar en Ada las tareas Cliente y Prestador de leyes

b) Modelar usando mailboxes las tareas Cliente y Prestador de leyes. Explicar la semántica de los mailboxes utilizadas en esta solución.

==========

Solución:

a) usando ADA

```
task body CLIENTE is
begin
   PRESTADOR.pedir_ley(NRO_LEY, cola_espera);
   if (cola_espera <> -1) then
      PRESTADOR.esperar_en_cola[cola_espera];
   end if
   fotocopiar();
   PRESTADOR.devolver_ley(NRO_LEY);
end

task body PRESTADOR is

   /*
      Existe una estructura que contiene un mapeo entre los números de leyes en
      uso y las colas en las que esta está esperando. Soporta las funciones:

      cantidad_ley_en_uso(NRO_LEY) --> retorna la cantidad de personas que
                                       esperan la ley
      incrementar_ley_en_uso(NRO_LEY) --> incrementa la cantidad de personas que
                                          esperan la ley
      decrementar_ley_en_uso(NRO_LEY) --> decrementa la cantidad de personas que
                                          esperan la ley
      obtener_cola_espera(NRO_LEY) --> retorna la cola donde tengo que esperar
                                       por la ley
   */

   entry pedir_ley(in NRO_LEY: integer; out: cola_espera: integer);
   entry devolver_ley(in NRO_LEY: integer);
```

```
   entry esperar_cola[1..25];

begin
   loop
      select
         when cantidad_adentro < 25
         accept pedir_ley(NRO_LEY, cola_espera) do
            if cantidad_ley_en_uso(NRO_LEY) > 0 then
                cola_espera = obtener_cola_espera(NRO_LEY)
          else
              cola_espera = -1
            incrementar_ley_en_uso(NRO_LEY)
            end if
            cantidad_adentro++
         end accept
      or
         when cantidad_adentro > 0
         accept devolver_ley(N:in integer) do
          NRO_LEY= N
       end accept
       decrementar_ley_en_uso(NRO_LEY)
         cantidad_adentro--
         if cantidad_ley_en_uso(NRO_LEY) > 0 then
          accept esperar_en_cola[obtener_cola_espera(NRO_LEY)];
         end if
      end select
   endloop
end


b) usando mailboxes

--- Solucion 1

Asumimos mailboxes infinitos (los mailboxes son siembre con resolución FIFO) por
lo que el Send es no bloqueante
El Receive es bloqueante (o sea si mailbox vacio)

Uso los siguientes mailboxes:

  mbTOKEN -> Mailbox con los tokens para entrar a pedir la ley. Los mensajes
             tienen el número de cola en la que espero (1..25)
  mbCLIENTE_PEDIDO -> Mailbox con los pedidos de los clientes al prestador. Los
                      mensajes tienen el número de ley, el tipo de
                      pedido, y la cola donde esperar
  mbCLIENTE_RESPUESTA[1..25] -> Mailbox con las respuestas del prestador con la
                                cola donde tienen que esperar
  mbCLIENTE_ESPERA[1..25] -> Mailbox donde espero a que la ley esté disponible

void cliente() {
   int NRO_LEY = NNNNNM;

   // Permiso para entrar. Me dan un número de canal (de los 25 que hay)
   receive(mbTOKEN, <idx>);

   // Pido la ley
   send(mbCLIENTE_PEDIDO, <NRO_LEY,PEDIR,idx>);

   // Recibo si la tengo o si esta prestada
   receive(mbCLIENTE_RESPUESTA[idx], <cola_espera>);

   // En algún momento, la ley la voy a recibir por acá. No importa el msg
   receive(mbESPERA[cola_espera], null);

   // acá tengo la ley
   fotocopiar();

   // Devuelvo la ley
   send(mbCLIENTE_PEDIDO, <NRO_LEY, DEVOLVER>);
```

```
   // Me voy del salón así que devuelvo mi numero de "entrada"
   send(mbTOKEN, <idx>);
}


void prestador() {
   // Esto sirve para que entre 25, a los 25 canales que tengo
   for (int i = 0; i < 25; i++) {
      send(mbTOKEN, <i>);
   }

   // Eternamente escucho pedidos y devoluciones
   while (true) {
      // Recibimos un pedido
      receive(mbCLIENTE_PEDIDO, <NRO_LEY, OPERACION, idx>);
      if (OPERACION == PEDIR) {
         // Verificamos si otro pidió la ley, si la esta usando y en que canal
         // esperar la respuesta (-1 si la ley no esta en uso)
         int cola_espera = ley_en_uso(NRO_LEY)
         if (cola_espera == -1) {
            cola_espera = obtener_nueva_cola_espera()

            // Incremento en uno la cantidad de procesos en espera por esa ley
            incrementar_ley_en_uso(NRO_LEY, cola_espera)

            // Le avisamos al cliente que la ley va a estar pronta en la cola
            indicada
            send(mbCLIENTE_RESPUESTA[idx], <cola_espera>)

            // La ley ya esta disponible, la mandamos
            send(mbESPERA[cola_espera], null)
         } else {
            // Le avisamos al cliente que la ley va a estar pronta en la cola
            indicada
            // En este caso la ley no esta disponible
            send(mbCLIENTE_RESPUESTA[idx], <cola_espera>)
         }
      } else {
         // Estamos devolviendo una ley. Hay que ver si en alguna cola alguien
         // la esta esperando. Si es así, se la pasamos a este
         int cola_espera = ley_en_uso(NRO_LEY)

         int cnt_pendiente = decrementar_ley_en_uso(NRO_LEY, cola_espera)
         if (cnt_pendiente > 0) {
            // La ley ya esta disponible, la mandamos un aviso al que este
            // esperando por ahí
            send(mbESPERA[cola_espera], null)
         }
      }
   }
}


--- Solución 2

Asumo mailboxes infinitos (los mailboxes son siembre con resolucion FIFO) por lo
que el Send es no bloqueante
El Receive es bloqueante (osea si mailbox vacio)

Uso los siguientes mailboxes:

  mbENTRADA -> Mailbox utilizado para permitir la entrada a la sala a solo 25
               clientes
  mbMUTEX -> Mailbox utilizado para mutoexcluir una zona critica
  mbCLIENTE_PEDIDO -> Mailbox con los pedidos de los clientes al prestador. Los
                      mensajes tienen el numero de ley, el tipo de
                       pedido, y la cola donde esperar
```

```
  mbCLIENTE_RESPUESTA -> Mailbox donde me comunican la casilla donde esperar por
                         la ley
  mbCLIENTE_ESPERA[1..25] -> Mailbox donde debo esperar por la ley a que este
                             disponible


typedef Struct
{
  long ley;
  int cantidad;
} Item;

Item pedidos[25];

void cliente() {
   int NRO_LEY = Dame_Nro_Ley();

   // Permiso para entrar
   receive(mbENTRADA, null);

   // Obtengo mutex
   receive(mbMUTEX, null);

   // Pido la ley
   send(mbCLIENTE_PEDIDO, <NRO_LEY,PEDIR>);

   // Recibo idx de donde obtener el mensaje
   receive(mbCLIENTE_RESPUESTA, <cola_espera>);

   // Libero el mutex
   send(mbMUTEX, null);

   // En algún momento, la ley la voy a recibir por acá. No importa el msg
   receive(mbESPERA[cola_espera], null);

   // acá tengo la ley
   fotocopiar();

   // Devuelvo la ley
   send(mbCLIENTE_PEDIDO, <NRO_LEY, DEVOLVER>);
   send(mbESPERA[cola_espera], null);

   // Me voy del salón así que devuelvo mi numero de "entrada"
   send(mbENTRADA, null);
}


void prestador() {
   // Esto sirve para que entre hasta 25
   for (int i = 0; i < 25; i++)
   {
      send(mbENTRADA, null);
      send(mbCLIENTE_ESPERA[i], null);
      pedidos[cola_espera].cantidad = 0;
   }
   send(mbMUTEX, null);

   // Eternamente escucho pedidos y devoluciones
   while (true) {
      // Recibimos un pedido
      receive(mbCLIENTE_PEDIDO, <NRO_LEY, OPERACION>);
      if (OPERACION == PEDIR) {
         // Buscamos el idx de esta ley
         int cola_espera = buscar_idx(NRO_LEY)
         pedidos[cola_espera].ley = NRO_LEY;
         pedidos[cola_espera].cantidad++;
         // Le avisamos al cliente que la ley va a estar pronta en la cola
         // indicada
         send(mbCLIENTE_RESPUESTA, <cola_espera>)
```

```
      } else {
         // Estamos devolviendo una ley.
         int cola_espera = buscar_idx(NRO_LEY);
         pedidos[cola_espera].cantidad--;
      }
   }
}

long buscar_idx(long nro_ley)
{
  int libre = -1;
  for(int i = 0; i < 25;  i++)
  {
      if(pedidos[i].cantidad == 0)
      {
            libre = i;
      }
      else if(pedidos[i].nro_ley == nro_ley)
      {
            return i;
      }
  }
  return libre;
}
```

==========