CC41B : Sistemas Operativos
Examen–Semestre Primavera'2001
Prof.: Luis Mateu.

Pregunta 1 (40%)

En una bolsa de comercio trabajan operadores que venden o compran acciones de la empresa ACME. Los operadores son representados mediante tareas de nSystem. Cuando un operador desea vender acciones invoca el procedimiento:

```
nTask t= vendo(precio);
```

En donde `precio` es un entero que indica el precio de venta de la acción de ACME. Este procedimiento espera hasta que aparezca un operador que compre acciones a ese precio, en cuyo caso retorna el identificador de la tarea compradora.

Cuando un operador desea comprar acciones invoca el procedimiento:

```
nTask t= compro(precio);
```

En donde `precio` es el precio máximo que el comprador está dispuesto a pagar por la acción de ACME. Si en ese instante ningún operador vende a ese precio (o menos) se retorna NULL de inmediato *sin quedarse en espera*. Si hay vendedores por ese precio, se elige como contraparte al operador que venda más barato y se retorna su identificador. La contraparte también debe retornar.

Se le pide a Ud. implementar una solución de este problema usando las tareas y mensajes de nSystem. Para ello Ud. debe programar los procedimientos `compro` y `vendo` y además una tarea coordinadora que reciba mensajes de los operadores con la siguiente estructura:

```
typedef struct {
  int tipo;   /* si es COMPRADOR o VENDEDOR */
  int precio; /* el precio especificado en la llamar a compro o vendo */
  nTask yo;   /* la tarea que realiza la llamada */
  nTask contraparte; /* la tarea con quien se hace la transaccion */
} Operacion;
```

Los campos `yo` y `contraparte` son asignados en la tarea coordinadora.

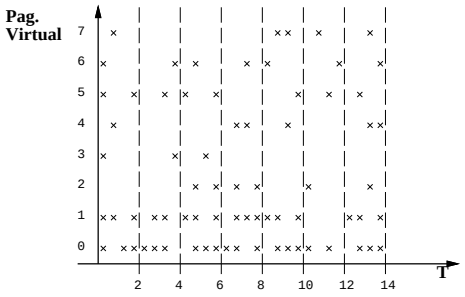
Torpedo:

```
int nSend(nTask t, void *msg);
void *nReceive(nTask *ptask, int timeout);
int nReply(nTask t, int rc);
PriQueue MakePriQueue(); /* Crea una cola de prioridad vacia */
void PriPut(PriQueue q, void *obj, int pri);
     /* Agrega obj en q con prioridad pri (menor es mejor) */
void *PriGet(PriQueue q);
     /* Entrega y extrae de q el objeto de mejor prioridad */
```

Pregunta 2 (30%)

a.- Un computador dispone de 4 páginas reales para procesos y usa la estrategia del reloj para el reemplazo de páginas. Se ejecuta un único proceso que ocupa 5 páginas virtuales (enumeradas de 1 a 5). Escriba una traza de 20 accesos de ese proceso a sus páginas virtuales de tal modo que se produzcan al menos *16 page-faults*. Haga supuestos razonables y explique.

b.- El siguiente gráfico muestra los accesos que realiza una aplicación en sus páginas virtuales.



El *working set* se calcula cada 2 unidades de tiempo. Suponiendo que el proceso nunca se va completamente a disco y que inicialmente todas sus páginas están en memoria, señale cuáles accesos pueden eventualmente producir un page-fault (indique intervalo de tiempo y página). Explique aquellos casos que le merezcan dudas.

Hint : no olvide que este proceso puede estar ejecutándose junto a otros procesos sedientos de memoria.

Pregunta 3 (30%)

a.- Un programa lee el último caracter de un archivo de 1 GB. El archivo se encuentra en una partición Unix con bloques de 1 KB. Esto se logra en C invocando el procedimiento `fseek` que permite posicionar el cursor de lectura directamente en el desplazamiento 1 GB-1. Explique cuantos bloques se deben leer de disco (como máximo) para obtener ese caracter.

b.- Suponga que el mismo programa de la parte a.- se ejecuta 2 veces seguidas. Explique por qué en la segunda ejecución no se realiza ningún acceso a disco.

c.- Se tienen 40 archivos de menos de 100 bytes cada uno, todos almacenados en una partición de 1 GB con un sistema de archivos FAT16. Calcule cuanto espacio en disco se consume para estos 40 archivos (calcule previamente de qué tamaño deben ser los bloques en la partición).