Ejercicios Propuestos Evaluación Continua
Memoria

Ejercicio 1: Queremos traducir las direcciones virtuales, generadas por un proceso, a direcciones reales en un sistema que soporta memoria virtual paginada. Cada página tiene un tamaño de 2 K y la tabla de páginas del proceso (simplificada, sin bits de control) es la siguiente:

Número de página	Número de marco
0	7
1	4
2	-
3	10
4	8

El proceso hace referencia a las siguientes direcciones virtuales (hexadecimal):

a) 0x1873
b) 0x2033
c) 0x1089

Traducir estas direcciones virtuales a direcciones físicas, indicando, para cada una de ellas, el número de marco y el desplazamiento dentro de ese marco en el que se encuentra físicamente la dirección referenciada.

Ejercicio 2: Tenemos un sistema con memoria virtual en el que la asignación de marcos es global y se utiliza una paginación bajo demanda pura. Existen cinco marcos de página y tenemos dos procesos **A** y **B**.

La cadena de referencia de páginas para el proceso **A** es:

7 15 17 13 15 11 15 11 9 9

La cadena de referencia de páginas para el proceso **B** es:

3 7 8 6 7 5 7 5 5 5

Suponemos que la primera página que se carga es la primera del proceso A, después la primera del proceso B y así continúan turnándose los dos procesos. Además se considera que no llegan procesos nuevos al sistema mientras A o B se están ejecutando.

Se pide **determinar el número de fallos de páginas** que se producen si aplicamos el algoritmo de reemplazo de páginas LRU.

Ejercicio 3: El siguiente algoritmo ordena ascendentemente un array de enteros mediante el método de inserción. Aunque no se escribe el código completo, el número de elementos del array y los elementos del array se toman de un fichero.

```
int main( ) {
        int  i, j, n, *a=NULL, kmin;
        FILE *fin=NULL, *fout=NULL;
        char nombre[80] ;
              .  //  Código:
              .  //      lectura de valor de n
              .  //      reserva de memoria para el array a
              .  //      apertura de ficheros
              .  //      lectura de los a[i] de fichero

        // Algoritmo de inserción:
        for (i =0; i<n; i++) {
            kmin = i;
            for ( j = i+1; j < n; j++)
                if ( a[kmin] > a[j] )
                    kmin=j;
            swap ( &a[i], &a[kmin]);
        }
        .   // Resto del código
} // End main
```

El programa se ejecuta en un computador de 8 MB de memoria principal destinada a la ejecución de programas de usuario (en estos 8 MB no se incluye la memoria ocupada por el sistema operativo). Además, la gestión de memoria se realiza mediante paginación bajo demanda con páginas de tamaño 4 KB y el algoritmo de reemplazo de páginas utilizado es LRU (least recently used). Suponiendo que el código del programa ocupa una página completa y que mientras que se ejecuta este programa no se ejecuta ningún otro:

a. Estima el número de fallos de páginas que se produce cuando la ejecución se realiza con un tamaño de array de $n = 4 * 2^{20}$ elementos y sabiendo que cada dato tipo entero ocupa 4 bytes.

b. En este caso la MMU (memory management unit) dispone de una TLB (translation lookaside buffer o buffer de traducción adelantada) con 16 entradas. Suponiendo un array de tamaño $n = 32 * 2^{10}$ elementos, estima el número de fallos en la TLB al ejecutar el programa.

Ejercicio 4: El tamaño de la memoria principal de un computador dado es de 40 MB, y ésta se gestiona con un esquema de memoria segmentada simple, donde todo proceso consta de 3 segmentos: el de código, el de datos y el de pila. El algoritmo de búsqueda de espacios libres sigue la política del *mejor ajuste*, realizando compactación cuando sea necesario.

El sistema operativo implantado en el computador es un sistema multiprogramado que ocupa en memoria 10 MB. El planificador de procesos está basado en cuatro niveles de prioridad:

1. Procesos de tiempo real que deben ejecutarse inmediatamente según la política FCFS, expulsando a cualquier proceso que esté en ejecución y tenga menor prioridad. Estos procesos se ejecutan hasta que terminen.
2. Procesos normales de usuario que se ejecutan en un planificador con retroalimentación de tres niveles, donde el cuanto de tiempo del planificador es 2 u.t.

Para simplificar el esquema del planificador, no vamos a tener en cuenta el tiempo que el sistema operativo consume en realizar el cambio de contexto cuando el proceso que estaba ejecutándose abandona dicho estado.

La siguiente tabla muestra los procesos que son creados en el sistema, el tiempo de ejecución estimado y el tamaño que ocupa su imagen (expresados en MBytes):

Tabla de tiempos y tamaños por procesos (Tabla 1)

Procesos	T. llegada (u.t)	Duración ráfaga (u.t.)	Tamaño (MB)
A	0	12	Código: 8 Datos: 2.0 Pila: 2.5
B	2	9	Código: 2.8 Datos: 1.4 Pila: 1.6
C	4	5	Código: 5 Datos: 1.0 Pila: 1.5
D	5	8	Código: 1.2 Datos: 1.4 Pila: 1.8
E	8	4	Código: 3 Datos: 1.6 Pila: 1.8
R1	4	10	Código: 0.8 Datos: 1.2 Pila: 1.0
R2	12	6	Código: 1 Datos: 0.5 Pila: 0.5

Los procesos A, B, C, D y E son procesos normales de usuario y los procesos R1 y R2 son procesos en tiempo real, R1 tiene un periodicidad en la petición del procesador cada 100 u.t y el proceso R2 cada 40 u.t.

Sabiendo que en la incorporación de procesos nuevos el planificador puede suspender procesos que estando ya en memoria tengan menor prioridad, realizar el siguiente estudio:

a) Seguimiento de la evolución de la memoria (tabla de segmentos) a medida que se ejecutan los procesos indicados en la tabla anterior, desde el instante t=0 hasta que se completan todos los procesos de usuario. Ejemplo: inmediatamente después de cargarse en memoria el proceso A, la tabla de segmentos es la siguiente:

Tabla de Segmentos

Base segmento (MB)	Tamaño segmento (MB)	Contenido segmento
0	10	S.O.
10	8	Código A
18	2	Datos A
20	2.5	Pila A
22.5	17.5	Libre

b) Seguimiento de la evolución de los estados de los procesos, indicando además en los procesos que están en estado listo su ubicación en las colas administradas por el sistema operativo.

Ejercicios 5: Supongamos un sistema con memoria virtual en el que se utiliza una paginación bajo demanda pura. Las direcciones que genera un proceso contienen en este orden: **número de página + desplazamiento**. Cada página tiene un tamaño de 2Kb. Las peticiones de memoria del proceso A vienen dadas por las siguientes direcciones (en hexadecimal):

22AC 44B0 4EBD 3952 4110 352B 391C 45A3 35B2 2D01 4192

a. Indique la secuencia de páginas referenciadas, correspondientes a las direcciones generadas por el proceso A.

b. En este sistema se ha asignado al proceso A tres marcos de página. Determine el número de fallos de página que se producen cuando se utiliza el algoritmo de reemplazo de páginas LRU.

Respuesta:

a. El tamaño de página es de 2 Kbits = 2^{11} bits. Como el direccionamiento se realiza a 1 byte (no me han dicho nada, supongo direccionamiento a nivel de byte), los bits necesarios para el desplazamiento vienen dados por las distintas direcciones a representar: $2^{11} / 2^{3} = 2^{8}$ ➔ 8 bits.

Cada dirección tiene 16 bits de los cuales los 8 primeros son para el número de página y los 8 siguientes para el desplazamiento (según acabamos de ver).De este modo la secuencia de páginas referenciadas en hexadecimal es: 22 , 44, 4E, 39, 41, 35, 39, 45, 35, 2D, 41 que tomándolas en decimal son: 34, 68, 78, 57, 65, 53, 57, 69, 53, 45, 65.

b. El algoritmo LRU, usado menos recientemente, sustituye la página menos usada en el pasado inmediato. Aplicándolo a nuestro problema se tiene la siguiente secuencia de páginas en los tres marcos disponibles

34	34	34	**57**	57	57	**57**	57	57	**45**	45
	68	68	68	**65**	65	65	**69**	69	69	**65**
		78	78	78	**53**	53	53	**53**	53	53
F*	F*	F*	F	F	F		F		F	F

Para calcular el número de fallos de página se tienen que tener en cuenta los tres fallos iniciales puesto que el sistema utiliza paginación bajo demanda y los propios de reemplazos de páginas generados por el algoritmo LRU. En total el número de fallos de página son: 3 iniciales + 6 reemplazo = 9 fallos de páginas.