Problema 1 (5.75 puntos)

A continuación se muestra un código en C que compila y funciona correctamente. El ordenador en el que se ha ejecutado el código es un ordenador con 16GBytes de RAM y 16GB de memoria swap (cada GB corresponde a 2^{30} bytes)

```c
int main(void)
{
  size_t size, nelems;
  int ret, *a;

  size   = 21474836480; // 20GBytes
  nelems = size / sizeof(int);

  printf("size = %lu   nelems = %lu\n", size,nelems);

  printf("Allocating vector\n");
  a = malloc(size);

  a[1000] = 10;

  ret = fork();

  if (ret == 0) {
     sleep(1);
     printf("Child: memory address of 'a': %lu\n", (unsigned long int) a);
     printf("Child: a[1000] = %d\n", a[1000]);
  } else {
     a[1000] = 20;
     printf("Parent: memory address of 'a': %lu\n", (unsigned long int) a);
     printf("Parent: a[1000] = %d\n", a[1000]);
     wait(NULL);
  }

  printf("Freeing vector\n");
  free(a);

  return 0;
}
```

Al ejecutar el código anterior se muestran los siguientes mensajes por pantalla. Observar que se reservan 20GBytes de memoria para un vector de enteros.

```
size = 21474836480   nelems = 5368709120
Allocating vector
Parent: memory address of 'a': 140587251462160
Parent: a[1000] = 20
Child: memory address of 'a': 140587251462160
Child: a[1000] = 10
Freeing vector
Freeing vector
```

En los siguientes apartados analizaremos paso por paso lo que hace el código. Leer primero todo el enunciado antes de comenzar a responder.

a) (0.5 puntos) Observar que el código permite reservar 20GBytes memoria aunque el ordenador sólo tiene 16Gbytes de RAM. ¿Cómo es posible esto? Comenta brevemente tu respuesta.

b) (0.75 puntos) En la línea 14 del código se hace una asignación a la posición 1000 del vector. ¿Qué es lo que sucede, técnicamente, en este momento? Es decir, ¿cómo es posible hacer esta asignación sabiendo que se han reservado 20GBytes de memoria?

c) (0.75 puntos) En la línea 16 se hace una duplicación del proceso. Observar que tanto el padre como el hijo almacenan el vector de memoria en la misma posición de memoria y pueden almacenar valores diferentes. ¿Cómo es posible ésto? Comenta tu respuesta apoyándote en un esquema si hace falta.

d) (0.75 puntos) Observa que el vector sólo se reserva una vez y, en cambio, se libera dos veces. ¿Por qué hay que liberar dos veces el vector habiéndolo reservado sólo una vez? Comenta tu respuesta.

e) (0.75 puntos, difícil) Supongamos que tanto el padre como el hijo comienzan a realizar un acceso intenso a los valores del vector para realizar cálculos matemáticos. Al cabo de un rato el sistema operativo mata los dos procesos, padre e hijo. Al aumentar la memoria swap a 32GB, sin embargo, los dos procesos funcionan correctamente y el sistema operativo no los mata. ¿Por qué funciona en un caso y en el otro no? Comentar la respuesta.

Se presenta a continuación un código similar que, en vez de usar un malloc, utiliza un archivo compartido mapeado en disco.

```
1 int main(void)
2 {
3   size_t size;
4   int ret, *a;
5
6   size = 21474836480; // 20GBytes
7   a = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANONYMOUS, -1, 0);
8
9   a[1000] = 10;
10
11  ret = fork();
12
13  if (ret == 0) {
14     sleep(1);
15     printf("Child: memory address of 'a': %lu\n", (unsigned long int) a);
16     printf("Child: a[1000] = %d\n", a[1000]);
17  } else {
18     a[1000] = 20;
19     printf("Parent: memory address of 'a': %lu\n", (unsigned long int) a);
20     printf("Parent: a[1000] = %d\n", a[1000]);
21     wait(NULL);
22  }
23
24  printf("Freeing vector\n");
25  munmap(a, size);
26 }
```

Al ejecutar el código se imprimen los siguientes mensajes por pantalla

```
Parent: memory address of 'a': 140433718976512
Parent: a[1000] = 20
Child: memory address of 'a': 140433718976512
Child: a[1000] = 20
Freeing vector
Freeing vector
```

Se pide

f) (0.75 puntos) Observar que en este caso tanto el padre como el hijo imprimen el mismo valor por pantalla. Observa que, igual que en el apartado 3, tanto el padre como el hijo indican que el vector está almacenado en la misma posición de memoria. Esta vez, en cambio, tanto padre como el hijo almacenan el mismo valor en la posición 1000. ¿Por qué? ¿Qué diferencia hay respecto el apartado 3? Comenta tu respuesta apoyándote en un esquema si hace falta.

g) (0.75 puntos) Supongamos, de nuevo, que tanto el padre como el hijo comienzan a realizar un acceso intenso a los valores del vector para realizar cálculos matemáticos. Con la configuración original de 16GB de RAM y 16GB de swap la aplicación no peta. ¿Por qué funciona en este caso y, sin embargo, en el apartado 5 no? Razona la respuesta.

h) (0.75 puntos, difícil) Al poco rato (uno o dos minutos) de haber comenzado a realizar los cálculos matemáticos observáis que, entre otras cosas, la interfaz gráfica de las aplicaciones responden lentamente a los clics con el ratón del usuario. Además, observáis que los dos procesos sólo están consumiendo recursos de 2 CPUs, mientras que las otras 2 CPUs prácticamente no hacen nada. ¿Qué está ocurriendo? ¿Por qué reaccionan lentamente las aplicaciones a los clics con el ratón? ¿Qué está pasando? Comenta y razona tu respuesta.

Problema 2 (1.5 puntos)

A continuación se muestra un código que compila y funciona correctamente. Este código es el mismo que el que se ha utilizado en el problema 2 del parcial 1.

```
1 #define N 10
2
3 int main()
4 {
5   pid_t pid;
6   int i, value, sum, fd[2];
7
8   pipe(fd);
9
10  i = N;
11  sum = 0;
12
13  for(i = 0; i < N; i++) {
14    pid = fork();
15    if (pid == 0) {
16      value = i;
17      printf("Value %d\n", value);
18      write(fd[1], &value, sizeof(int));
19      return 0;
20    }
21  }
22
23  for(i = 0; i < N; i++) {
24    read(fd[0], &value, sizeof(int));
25    sum += value;
26  }
27
28  printf("Value of sum: %d\n", sum);
29 }
```

En una de las ejecuciones del código (ordenador con 16GB de RAM y 4 CPUs) se muestran los siguientes mensajes por pantalla.

```
Value 0
Value 4
Value 6
Value 1
Value 8
Value 5
Value 2
Value 3
Value 7
Value 9
Value of sum: 45
```

Se pide responder a las siguientes preguntas de forma breve y concisa

a) (0.75 puntos) Observar que en el código se crean los hijos en orden: primero el 0, después el 1, luego el 2, y así sucesivamente. A la hora de ejecutar el código, en cambio, los hijos se ejecutan en un orden distinto del que se han creado. ¿Puedes explicar cuál es la razón de esto? Comenta tu respuesta.

b) (0.75 puntos) Cada vez que se ejecuta el código se observa que el orden en que se imprimen los mensajes de "Value" es diferente. ¿Cómo es posible? Comenta tu respuesta.

Problema 3 (2.75 puntos)

A lo largo del curso de Sistemas Operativos 1 se ha intentado dar la visión del sistema operativo como una máquina virtual. En particular, el sistema operativo hace las tareas de árbitro, de ilusionista y ofrece las llamadas al sistema para que los procesos puedan utilizar los servicios del sistema operativo.

Centrémonos en la tarea de ilusionista y, en particular, en la planificación de procesos y el sistema de memoria virtual. Respecto la planificación de procesos y el sistema de memoria virtual:

a) (0.5 puntos) ¿Cuál es el objetivo del planificador de procesos de un sistema operativo? Comenta tu respuesta.

b) (0.75 puntos) Los sistemas de escritorio utilizan un esquema de planificación de tipo MFQ (Multilevel FeedBack Queue). ¿Cuál es la razón por la que utilizan este tipo de planificación? ¿Qué aporta al usuario esta técnica de planificación para ejecutar aplicaciones? Comenta tu respuesta.

c) (0.75 puntos) Gracias al sistema de memoria virtual una aplicación puede ejecutarse aunque no esté completamente cargada en memoria. ¿Qué es lo que hace el sistema operativo, técnicamente, para que sea posible? Comenta tu respuesta.

d) (0.75 puntos) Las librerías dinámicas son librerías que se cargan en memoria al ejecutar una aplicación. Por ejemplo, la librería `libc` de `GNU` es la librería que tiene implementadas las funciones básicas como `open`, `read`, `write`, `malloc`, `printf`, etc. Supongamos que dos o más aplicaciones utilizan la misma librería. Al ejecutar cada una de las aplicaciones, ¿se cargará en la memoria física una copia independiente para cada una de las aplicaciones que la utiliza? ¿O la comparten? Si la comparten, ¿cómo consiguen hacerlo? Comenta tu respuesta.