1. **(2,5 puntos)**

Las prácticas de algunos alumnos se bloquean con ciertos datos de entrada. Un profesor decide realizar un programa que ejecute un comando indicado por parámetro que imprima un mensaje de error cuando pase un minuto de ejecución o la salida del comando supere el Megabyte de tamaño.

De manera que el programa con un comando bien implementado se ejecutaría:

```
# ./auto_ejec comando_bueno
entrada 1
entrada 2
...
entrada n
```

Si el comando se bloquea más de un minuto (60 segundos) la sesión es:

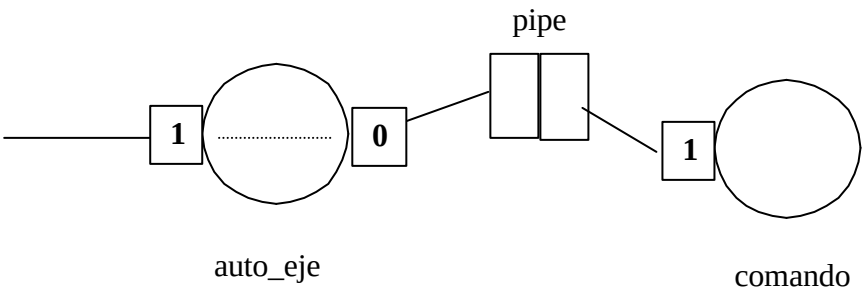
```
#./auto_ejec comando_bloqueado
entrada 1
entrada 2
entrada 3
ERROR: comando se bloquea.
```

Si el comando queda en un bucle infinito, la sesión queda:

```
# ./auto_ejec comando_infinito
entrada 1
entrada 1
entrada 1
...
ERROR: salida mayor de un mega.
```

El comando a ejecutar no necesita ningún parámetro.

Para controlar la salida del comando, el profesor piensa en el siguiente diseño:



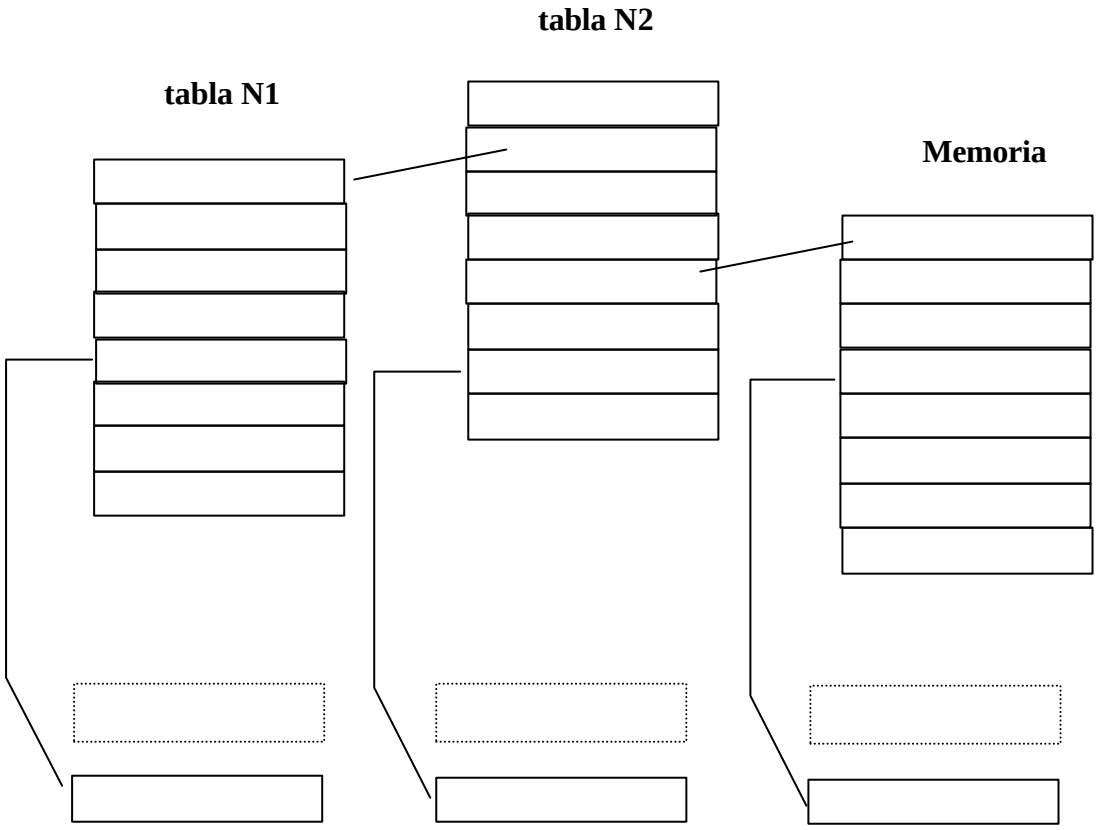
Para implementar el comando el profesor piensa en el siguiente código:

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

void fin_minuto ( int sno ) ;

int main ( int argc, char *argv[] ) {
        int err;
        int pd[2];
        pid_t cpid;
        int leido, leidos;
        char buf[1024];

        /* uso */
        if (argc < 2){
                printf("uso: %s comando\n",argv[0]);
                exit(1);
        }

        /* RELLENAR 1 : crear pipe */

        cpid = fork();                  /* fork */
        switch (cpid)   {
                case -1:                /* error fork */
                        perror("fork:");
                        exit(3);
                        break;
                case 0:
                        /* RELLENAR 2 : conectar salida al pipe */
                        execvp(argv[1],argv+1);         /* ejecutar comando */
                        perror("execvp:");
                        exit(4);
                        break;
                default:
                        /* RELLENAR 3 : conectar entrada del pipe */
                        signal(SIGALRM, fin_minuto);    /* cronometrar 60 segundos */
                        alarm(60);
                        /* RELLENAR 4 : leer y escribir, máximo 1 mega */
                        alarm(0);                       /* fin cronómetro */

                        if (leidos >= 1024*1024)        /* comprobar leidos */
                        {
                                printf("ERROR: salida mayor de un mega.\n");
                                exit(5);
                        }
                        break;
        }

        /* fin main */
        return 0;
}

void fin_minuto ( int sno )
{
        /* RELLENAR 5 : imprimir "ERROR: comando se bloquea.\n" y finalizar la ejecución*/
}
```

**Se pide** ayudar a nuestro profesor, completando las secciones de códigos indicadas a rellenar, justificando brevemente. Basta escribir el código distinguiendo las 5 zonas.

2. **(2,5 puntos)**

Suponga un sistema con memoria virtual paginada usando dos niveles de tablas de páginas, direccionamiento de 32 bits con páginas de 4 kbytes de tamaño.
Los atributos asociados a las páginas son: M (modificado), X(ejecutable), RO(sólo lectura), P(presente) y COW(Copiar en la escritura). Este último atributo indica que si na marcada con COW = 1, primero ha de duplicar la página y después modificar esta duplica.




A) Se desea que la tabla de segundo nivel ocupe una página, es decir, 4Kbytes.

- ¿Cuántos bits son necesarios para direccionar esta tabla?
- ¿Sería posible incluir esta tabla de 2º nivel en memoria virtual y no en

- ¿Cuánto ocupa en memoria principal la tabla de nivel 1?
- ¿Es posible tener en Memoria virtual la rutina de tratamiento de fallo de página? ¿Por qué si? / ¿Por qué no?

B) Se desea implementar la llamada al sistema:

mm_FastMemCpy ( void * dst, void * src, int npages ) ;

Para poder tener a partir de la dirección indicada en 'dst' los mismos datos que están a partir de 'src'. Se garantizan que ambas direcciones (src y dst) son el comienzo de una página y npages es un número de páginas válido.

| Nombre y Apellidos | NIA |
|---|---|
| | |

**Se pide:** Escriba y describa en pseudocódigo la función descrita, usando la siguiente tabla:

| **Pasos** | **Comentario** |
|---|---|
| Acceder a ... | Es necesario comprobar ... |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

**3. (2,5 puntos)**

Cuatro alumnos están jugando en clase en el descanso a un juego de cartas que implica coordinación (no saltarse los turnos) y exclusión mutua (mientras uno roba una carta o la cambia con el compañero el resto debe esperar a que acabe para no quitarse la carta de la mano).

Se plantea tomar como base esta experiencia para implantar una arquitectura de comunicación entre procesos que permita realizar una práctica de inteligencia artificial en la que cada alumno codificará su estrategia dentro de las siguientes funciones aisladas:

```
Proceso_jugador()
{
        while ( 1 ) {
                Pensar_la_jugada();   // va calculando la mejor opción
                Jugar();              // intercambia rápidamente las cartas
        }
}
```

Todos los procesos serán hijos de un proceso principal denominado partida, y por tanto heredaran tras el fork y exec correspondiente una serie de semáforos inicializados de la siguiente manera:

```
#define NUMERO_DE_JUGADORES     4
#define NUMERO_DE_CARTAS        40
#define SIGUIENTE ((IDENTIFICADOR_DE_JUGADOR + 1) % NUMERO DE JUGADORES)
Semáforo esperar [NUMERO_DE_JUGADORES];
User_ID baraja [NUMERO_DE_CARTAS];
```

Se pide:

1. Intercalar en el código propuesto para proceso_jugador() las ordenes wait y signal necesarias sobre los semáforos esperar[IDENTIFICADOR_DE_JUGADOR] y esperar[SIGUIENTE] para asegurar el correcto funcionamiento del mismo, es decir, que cada jugador deba esperar a su turno antes de poder ejecutar jugar()

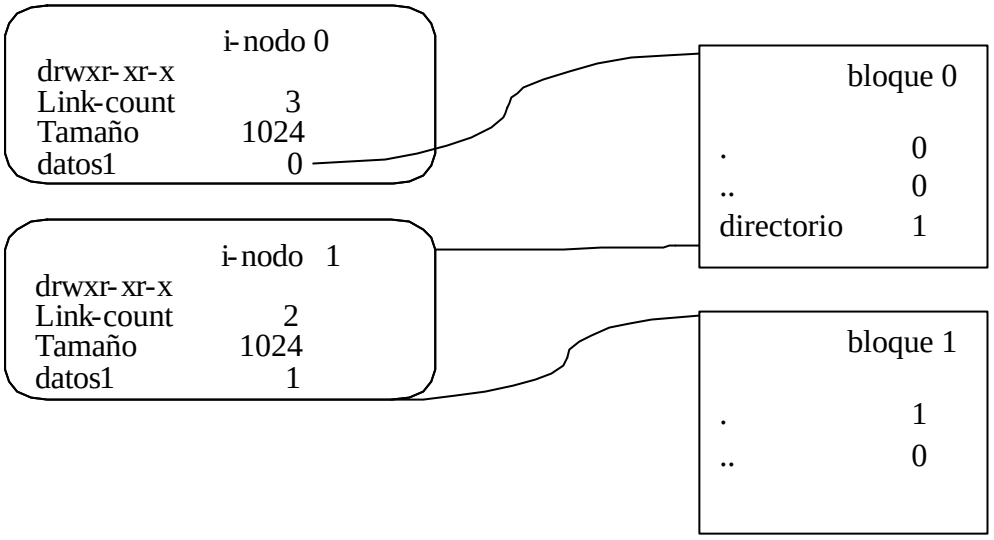
2. ¿Cómo debe estar inicializado el array de semáforos para que el funcione, es decir, empiece el primer jugador y ceda el turno al siguiente y así sucesivamente, quedando bloqueado cada cual hasta que le toque?

3. En el caso de que jugar() no se ejecute de forma atómica, ya que tendrá que modificar al menos dos entradas de la tabla baraja… ¿Habría que incorporar algo para garantizar la exclusión mutua en la ejecución de jugar()? De ser necesario, escribir el código. De no serlo, explicar por que.

4. **(2,5 puntos)**

El sistema de ficheros de la figura muestra el resultado del disco /dev/hda3 una vez montado sobre el directorio /usr/extra.

Se trata de una partición muy pequeña en la que tan solo hay 10 inodos y 50 sectores de datos de 1K cada uno de ellos.




Sobre ese disco y con CWD /usr/extra, se ejecutan los comandos siguientes:

```
rmdir directorio
touch fichero1
ln fichero1 fichero2
ln –s ./fichero2 fichero3
```

nota: touch actualiza la fecha de un fichero, y si no existe lo crea vacio.

Se pide:

1.) Dibujar la estructura del disco al finalizar las operaciones indicadas, indicando los valores de inodos y bloques según lo mostrado en el dibujo

2.) ¿Cuántos ficheros de 1K podrían crearse a partir de ese momento?

3a.) ¿Cuál seria el tamaño máximo de fichero que podríamos crear si en lugar de crear muchos de 1K creásemos solo uno pero grande?

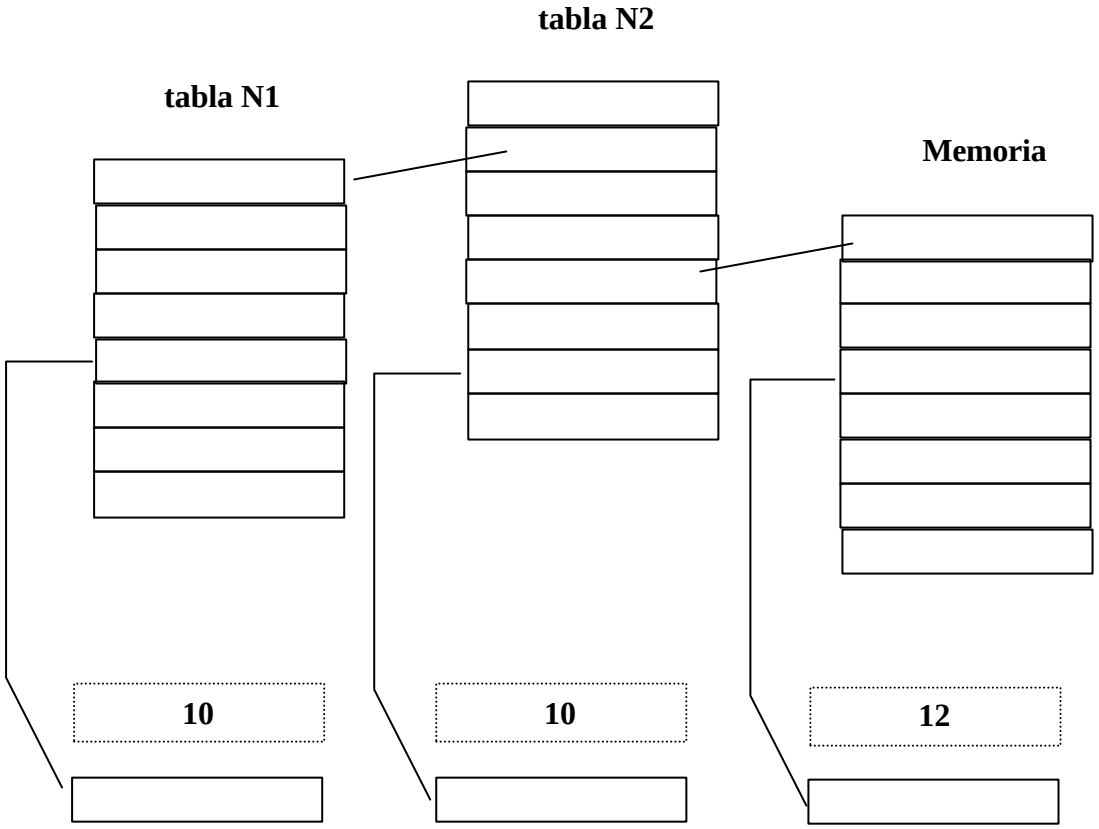
3b.) ¿Cuantos enlaces físicos (o duros) podrían hacerse a ese fichero de tamaño una vez creado el mismo? Explica la respuesta

3c.) ¿Y si en lugar de físicos los quisiésemos simbólicos, cuantos podrían crearse?

## 1 SOLUCION

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>


void fin_minuto ( int sno ) ;

int main ( int argc, char *argv[] )
{
        int err;
        int pd[2];
        pid_t cpid;
        int leido, leidos;
        char buf[1024];

        /* uso */
        if (argc < 2)
        {
           printf("uso: %s comando\n",argv[0]);
           exit(1) ;
        }

        /* pipe */
        err = pipe(pd);
        if (err < 0)
        {
                perror("pipe:");
                exit(2) ;
        }

        /* fork */
        cpid = fork();
        switch (cpid)
        {
                case -1:
                        /* error fork */
                        perror("fork:");
                        exit(3);
                        break;
                case 0:
                        /* salida al pipe */
                        close(pd[0]);
                        close(1);
                        dup(pd[1]);
                        close(pd[1]);




                        /* ejecutar comando */
                        execvp(argv[1],argv+1);
                        perror("execvp:");
                        exit(4);
                        break;

        default:
                /* entrada del pipe */
                close(pd[1]);
                close(0);
                dup(pd[0]);
                close(pd[0]);

                /* cronometrar 60 segundos */
                signal(SIGALRM, fin_minuto);
                alarm(60);

               /* leer y escribir, máximo 1 mega */
                leidos = 0;
               do {
                     leido = read(0,buf,1024);
                     if (leido <= 0)  break;

                    write(1,buf,1024);
                     leidos += leido;

                 } while(leidos < 1024*1024+1);

                /* fin cronómetro */
                alarm(0);

                /* comprobar leidos */
                if (leidos >= 1024*1024)
                {
                  printf("ERR: salida > mega.\n");
                  exit(5);
                }
                break;
        }

        /* fin main */
        return 0;
}


void fin_minuto ( int sno )
{
      printf("ERROR: comando se bloquea.\n");
      exit(6);
}
```

**2 SOLUCION**




A) Se desea que la tabla de segundo nivel ocupe una página, es decir, 4Kbytes.

- ¿Cuántos bits son necesarios para direccionar esta tabla? **10**
- ¿Sería posible incluir esta tabla de 2º nivel en memoria virtual y no en memoria física? **Sí**
- ¿Cuánto ocupa en memoria principal la tabla de nivel 1? **2^10=4Kb**
- ¿Es posible tener en Memoria virtual la rutina de tratamiento de fallo de página?. ¿Por qué si?. ¿Por qué no?. **No, porque si no se encuentra presente la rutina de tratamiento de excepción en memoria, quedaría en un bucle infinito.**

B) mm_FastMemCpy ( void * dst, void * src, int npages ) ;

| **Pasos** | **Comentario** |
|---|---|
| Acceder a la tabla de página de Nivel 1 y Nivel 2 correspondiente a la dirección origen. | Es necesario comprobar que es una página válida y asignada al proceso. |
| Acceder a la tabla de página de Nivel 1 y Nivel 2 correspondiente a la dirección destino. | Es necesario comprobar que es una página válida y asignada al proceso. |
| Copiar tantas entradas como el parámetro 'npages' indique, de la tabla de Nivel dos en la que está la dirección 'src' a la tabla de nivel dos en que está la dirección 'dst'. | Puede realizarse en un bucle o copiando la zona de tabla de páginas de nivel dos en que están los 'npages' descriptores a partir del asociado a 'src'. |
| Establecer el atributo COW (Copy On Write) a uno | Permite retrasar la copia de contenidos al momento de actualización de datos. |

**3 SOLUCION**

1)

```
Proceso_jugador()
{
        while ( 1 ) {
                Pensar_la_jugada();
                Wait esperar[IDENTIFICADOR_DE_JUGADOR];
                Jugar();
                Signal esperar[SIGUIENTE];
        }
}
```

2)

Semáforo esperar [NUMERO_DE_JUGADORES] = { 1, 0, 0 …. 0 , 0 };

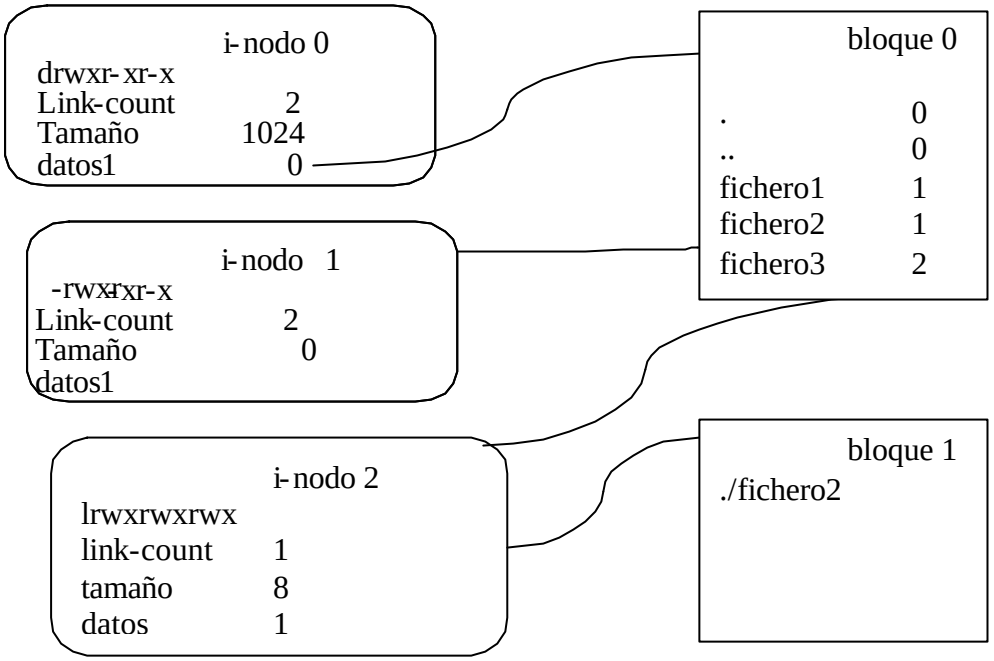
Como no podemos garantizar cual de los jugadores termina primero de pensar la jugada, solo un semáforo debe estar sin bloquear (podrían inicializarse todos a cero si antes de hacer los fork se hiciese un signal a uno de ellos).

3)

No hace falta crear ningún semáforo binario tipo mutex, ya que garantizamos con la perfecta secuenciación de accesos a la función jugar() que uno y solo uno entra en esa sección critica a la vez.

**4 SOLUCION**

1. Dibujar la estructura del disco al finalizar las operaciones indicadas.




2. ¿Cuántos ficheros de 1K pueden crearse a partir de ese momento?
Tantos como i-nodos, y solo quedan 10 – 3 = 7

3. ¿Cuál seria el tamaño máximo de fichero que podríamos crear si en lugar de crear muchos de 1K creásemos solo uno pero grande?
Tantos K como bloques libres quedan: 50 - 2 = 48 K

4. Cuantos enlaces físicos podrían hacerse a ese fichero de tamaño máximo una vez creado el mismo. Explica la respuesta
En el directorio que existe, pueden introducirse enlaces fisicos con distinto nombre pero apuntando al mismo i-nodo hasta que se llene el directorio.

5. ¿Y si en lugar de físicos los quisiésemos simbólicos, cuantos podrían crearse?
Ninguno, ya que necesitaríamos al menos un bloque de datos por enlace simbolico, y el disco esta ya lleno.