Ejercicio 4.1 - Tuberías

In [ ]:

```
%%bash
rm -rf *.c *.o *.elf tuberia
/*
Daniel Ledesma Ventura
Badr Guaitoune Akdi
*/
```

Ejercicios

Ejercicio 1 - tubAnon.c Estudia y ejecuta el programa `tubAnon.c` en el que los procesos padre e hijo se comunicarán a través de una única tubería sin nombre. Describe el comportamiento del programa.

1. ¿Cómo se finaliza su ejecución?
2. ¿Qué ocurre si creamos la tubería justo después de la llamada a `fork()` (antes del `switch`).

Respuesta:

---

1. La ejecución se finaliza cuando se introduce una cadena de caracteres en un buffer,en la que la posicion del buffer 0 es igual al acarater q, es decir se cumple que buffer[0] == 'q';Hay que decir que antes de parar el proceso las tuberias se comunican para pasarse el buffer introducido en este caso el caracter 'q';
2. Si se creamos las tuberias despues del fork los descriptores de los extremos no estarian conectado,puesto que cuendo se realiza la llmada fork se crea otro proceso hijo que hereda o comparte los descriptores del padre que tine abiertos o creados en ese momento pero despues del fork ya no comparten ningun descriptor abierto o creado despues de la llamada, en resumen no comparten la memoria que alamcena la tuberia (tuberias[2]).

In [ ]:

```
%%writefile tubAnon.c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

int main() {
  int pid;
  int tuberias[2];
  char buffer[256];
  int fin;
  fin = 0;

  if ( pipe(tuberias) == -1 ) {
    perror("pipe");
    exit(-1);
  }

  pid = fork();
  switch (pid) {
    case -1:
      perror("fork");
      exit(-1);
      break;
    case 0:
      printf("Lector de la tuberia\n");
      close(tuberias[1]);
      while (!fin) {
        read(tuberias[0],buffer,256);
        if (buffer[0] == 'q')
          fin = 1;
        else
          printf("Mensaje recibido(%i): %s\n",getpid(),buffer);
      }
```

```
      close(tuberias[0]);
      exit(0);
      break;
    default:
      printf("Escritor de la tuberia\n");
      close(tuberias[0]);
      while(!fin) {
        scanf("%s",buffer);
        write(tuberias[1],buffer,strlen(buffer)+1);
        if (buffer[0] == 'q')
          fin = 1;
        }
        close(tuberias[1]);
        exit(0);
        break;
  }
}

/*
badr@badr:~/Escritorio/ASO/Ejercicio4/Ejercicio4.1$ ./tubAnon.elf
Escritor de la tuberia
Lector de la tuberia
BADR
Mensaje recibido(4285): BADR
GUAITOUNE
Mensaje recibido(4285): GUAITOUNE
q
*/
```

Este programa se debe de ejecutar en terminal, ejemplo de salida:

```
gcc tubAnon.c -o tubAnon.elf
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$ ./tubAnon.elf
Escritor de la tuberia
Lector de la tuberia
Hola
Mensaje recibido(121): Hola
Adiós
Mensaje recibido(121): Adiós
q
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$
```

Ejercicio 2 - tubNombre.c Estudia y ejecuta el programa `tubNombre.c` que reproduce el comportamiento del programa `tubAnon.c`, pero utilizando tuberías con nombre para realizar la comunicación entre padre e hijo. ¿Se crea algún fichero nuevo en el sistema de ficheros, de qué tipo? Modifica el código para que el proceso escritor reciba una señal `SIGPIPE`.

Respuesta:

SI, se crea un fichero especial con mkfifo ./tuberia mkfifo construye un fichero especial FIFO con el nombre camino. modo especifica los permisos del FIFO. Son modificados por la máscara umask del proceso de la forma habitual: los permisos del fichero recién creado son (modo & ~umask).

In [ ]:

```
%%writefile tubNombre.c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

int main() {
  int pid;
  int tuberia;
  char buffer[256];
  int fin;
```

```
  fin = 0;
  mkfifo("./tuberia",0666);
  pid = fork();

  switch (pid) {
    case -1:
      perror("fork");
      exit(-1);
      break;
    case 0:
      printf("Lector de la tuberia\n");
      tuberia = open("./tuberia",O_RDONLY);
      while (!fin) {
        read(tuberia,buffer,256);
        if (buffer[0] == 'q')
          fin = 1;
        else
          printf("Mensaje recibido(%i): %s\n",getpid(),buffer);
        sleep(5);
      }
      close(tuberia);
      exit(0);
      break;
    default:
      printf("Escritor de la tuberia\n");
      tuberia = open("./tuberia",O_WRONLY);
      while(!fin) {
        scanf("%s",buffer);
        write(tuberia,buffer,strlen(buffer)+1);
        if (buffer[0] == 'q')
          fin = 1;
      }
      close(tuberia);
      exit(0);
      break;
  }
}
/*
badr@badr:~/Escritorio/ASO/Ejercicio4/Ejercicio4.1$ ./tubNombre.elf
Escritor de la tuberia
Lector de la tuberia
BADR
Mensaje recibido(4578): BADR
GUAITOUNE
Mensaje recibido(4578): GUAITOUNE
q
*/
```

Este programa se debe de ejecutar en terminal, ejemplo de salida:

```
gcc tubNombre.c -o tubNombre.elf
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$ ./tubNombre.elf
Escritor de la tuberia
Lector de la tuberia
Hola
Mensaje recibido(203): Hola
Adiós
Mensaje recibido(203): Adiós
q
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$
```

Mientras se escribe *Hola*, en otro terminal:

```
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$ ll
total 60
drwxr-xr-x  3 usuarioso usuarioso  4096 Apr 17 08:53 ./
drwxr-xr-x 14 usuarioso usuarioso  4096 Apr 15 10:14 ../

...

-rw-r--r--  1 usuarioso usuarioso  1081 Apr 17 08:48 tubNombre.c
-rwxr-xr-x  1 usuarioso usuarioso 12968 Apr 17 08:48 tubNombre.elf*
```

```
-rwxr-xr-x  1 usuarioso usuarioso 12968 Apr 17 08:48 tubNombre.elf*
prw-r--r--  1 usuarioso usuarioso     0 Apr 17 08:53 tuberia|
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$
```

In [ ]:

```
%%writefile tubNombre2.c
/*
Includes:
*/
/*************************************************************************/
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h> s
/*************************************************************************/

// Pasamos la variable de fin a ámbito global
int fin = 0;

/*
Funciones finTub (que pone fin a 1), e ini_manejador
*/
void finTub(int signo) {
/*************************************************************************/
 fin = 1;
/*************************************************************************/
}

void* ini_manejador(int signal, void* manejador) {
/*************************************************************************/
 struct sigaction act;
 struct sigaction old_act;
 act.sa_handler = handler;
 sigemptyset(&act.sa_mask);
 act.sa_flags = SA_RESTART;
 sigaction(signal, &act, &old_act);
    return NULL;
/*************************************************************************/
}

int main() {
  int pid;
  int tuberia;
  char buffer[256];

  mkfifo("./tuberia",0666);
  pid = fork();

  switch (pid) {
    case -1:
      perror("fork");
      exit(-1);
      break;
    case 0:
      printf("Lector de la tuberia\n");
      tuberia = open("./tuberia",O_RDONLY);
      while (!fin) {
        read(tuberia,buffer,256);
        if (buffer[0] == 'q') {
          fin = 1;
          printf("Hijo cierra la tuberia\n");
/*
Se cierra la tubería para generar la señal SIGPIPE al escribir
*/
/*************************************************************************/
          close(tuberia);
/*************************************************************************/
        } else {
          printf("Mensaje recibido(%i): %s\n",getpid(),buffer);
```

```
        }
        sleep(5);
      }
      exit(0);
      break;
    default:
      printf("Escritor de la tuberia\n");
/*
Preparamos la señal SIGPIPE
*/
/*************************************************************************/
   iniHandler(SIGPIPE,finTuberia);
/*************************************************************************/
      tuberia = open("./tuberia",O_WRONLY);
      while(!fin) {
        scanf("%s",buffer);
        // Con la tubería cerrada, el write activará la señal SIGPIPE
        write(tuberia,buffer,strlen(buffer)+1);
      }
      printf("Señal recibida 2/2.\n");
      exit(0);
      break;
  }
  while (wait(NULL)!= -1) {};
  printf("Esto es todo amigos\n");
}

/*
badr@badr:~/Escritorio/ASO/Ejercicio4/Ejercicio4.1$ ./tubNombre2.elf
Escritor de la tuberia
Lector de la tuberia
BADR
Mensaje recibido(5725): BADR
GUAITOUNE
Mensaje recibido(7139): GUAITOUNE
q
Hijo cierra la tuberia
q


Señal recibida 2/2.
*/
```

Este programa se debe de ejecutar en terminal, ejemplo de salida:

```
gcc tubNombre2.c -o tubNombre2.elf
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$ ./tubNombre2.elf
Escritor de la tuberia
Lector de la tuberia
Hola
Mensaje recibido(177): Hola
Adiós
Mensaje recibido(177): Adiós
q
q
Señal recibida 1/2.
Señal recibida 2/2.
```

Ejercicio 3 - tub-select.c Se pide crear un programa que creará `NHIJOS` tuberías para comunicarse con `NHIJOS` procesos hijos (cada tubería se utilizará exclusivamente para la comunicación con un solo hijo. `NHIJOS` será una constante (macro) de preprocesador que se podrá modificar en compilación.

Cada proceso hijo tendrá el mismo comportamiento:

- Generará un número entero aleatorio entre 4 y 15.
- Llamará a `sleep` para dormir los segundos indicados por el número aleatorio.
- Escribirá en su tubería su índice de proceso hijo (recibido como argumento) y el tiempo que ha estado dormido.
- Permanecerá en un bucle repitiendo los pasos anteriores hasta que detecte que no hay lector en el otro extremo de la tubería. En ese caso, terminará su ejecución escribiendo un mensaje por la salida estándar.

Por su parte, el proceso padre quedará a la espera de actividad por cualquier tubería o por la entrada estándar. Para ello, usará la

Por su parte, el proceso padre quedará a la espera de actividad por cualquier tubería o por la entrada estándar. Para ello, usará la llamada al sistema `select()` con un *timeout* de 15s. Si pasado este tiempo no ha recibido nada por ninguna tubería (o entrada estándar), el proceso finalizará mostrando un mensaje de error por la salida estándar. Si detecta actividad en una tubería, leerá de dicha tubería y mostrará por la salida estándar los valores leídos (identificador de proceso hijo y tiempo de espera). Asimismo, si se detecta actividad en la entrada estándar, se leerá una línea. Si se lee *quit* o *exit*, cerrará sus extremos de lectura de las tuberías, esperará a que terminen todos sus hijos y se finalizará la ejecución con un mensaje de despedida.

Se deja un ejemplo de uso de *select*:

```c
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int main(void) {
  fd_set conjunto;
  struct timeval timeout;
  int cambios;
  char buffer[80];

  FD_ZERO(&conjunto);
  FD_SET(0, &conjunto);

  timeout.tv_sec = 2;
  timeout.tv_usec = 0;
  cambios = select(1, &conjunto, NULL, NULL, &timeout);

  if (cambios) {
    printf("Datos nuevos.\n");
    scanf("%s",buffer);
    printf("Datos: %s\n",buffer);
  }
  else
    printf("Ningun dato nuevo en 2 seg.\n");

  return 0;
}
```

In [ ]:

```c
%%writefile tub-select.c
/*****************************************************************************/
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/wait.h>
#include <time.h>
/*****************************************************************************/

#ifndef NHIJOS
#define NHIJOS 2
#endif

// TIpo de datos para las variables que enviamos/leemos por el pipe
typedef struct _dato_t {
  int index;
  int tiempo;
} _t_dato_;

int terminaHijo = 0;

void finTub(int signo) {
  terminaHijo = 1;
}
```

```
void* ini_manejador(int signal, void* manejador) {
  struct sigaction act;
  struct sigaction old_act;

  act.sa_handler = manejador;
  sigemptyset(&act.sa_mask);
  act.sa_flags = SA_RESTART;
  sigaction(signal, &act, &old_act);
  return NULL;
}

/*
Función que ejecutará cada hijo.
- index es el índice el bucle for del padre cuando se creó el hijo
- fd es el descriptor de fichero de extremo de escritura de una tubería
*/
void fhijo(int index,int fd) {
  int r;
/*
Preparar el código para capturar la señal SIGPIPE usando ini_manejador.
La función que trate dicha señal, deberá poner terminaHijo a 1
*/
/*************************************************************************/
 ini_manejador(SIGPIPE,finTub);
/*************************************************************************/
  srand(time(NULL));
  while (!terminaHijo) {
    _t_dato_ dato;
    r  = rand() % 15 + 4;
    sleep(r);
    printf("Hijo %d escribe en pipe el tras dormir %ds\n",index,r);
/*
Escribimos en la tubería el índice y el tiempo dormido.
Creamos una estructura de tipo _t_dato para incluir ambos enteros.
*/
    dato.index = index;
    dato.tiempo = r;
/*************************************************************************/
   if(write(fd,&dato,sizeof(dato))== -1){
       printf("Write en el hijo %d falla.\n");
   }
/*************************************************************************/
  }
  printf("Hijo %d: el padre dice que a terminar!!\n", index);
}

int main() {
  int pid;
  int tuberias[2];
  int tpadre[NHIJOS];
  fd_set conjunto;
  int i=0;
  struct timeval timeout;

  for (i=0; i<NHIJOS; i++) {
/*
Crear NHIJOS tuberías e hijos. Cada tubería permitirá comunicación unidireccional
entre el padre y un hijo (el hijo escribirá y el padre leerá).
*/
/*************************************************************************/
   if(pipe(tuberias)==-1){
       perror("pipe");
       exit(-1);
   }
   pid = fork();
/*************************************************************************/
    switch (pid) {
      case -1:
        perror("fork");
        exit(-1);
        break;
      case 0:
/*
Cada hijo hará una llamada a fhijo. El primer argumento será  i.
El segundo, el descriptor del extremo de escritura de su tubería.
*/
/*************************************************************************/
```

```
     close(tuberias[0]);
     fhijo(i,tuberias[1]);
/*************************************************************************/
        exit(0);
        break;
      default:
/*************************************************************************/
        close(tuberias[1]);
        tpadre[i] = tuberias[0];
/*************************************************************************/
        sleep(1); // Esperar a que el hijo empiece ...
        break;
    }
  }

  int cambios;
  while(1) {
    printf("Padre hace select.\n");
    // El proceso padre se queda en un bucle haciendo select para saber cuándo
    // debe leer de alguna tubería/entrada estándar.
    FD_ZERO(&conjunto);
    FD_SET(0, &conjunto); // incluye stdin en el conjunto
/*
Incluir resto de descriptores en el conjunto.
*/
    for (i=0;i<NHIJOS;i++) {
/*************************************************************************/
    FD_SET(tpadre[i], &conjunto);
/*************************************************************************/
    }
/*
Establecer el timeout y llamada a select
*/
/*************************************************************************/
   timeout.tv_sec = 15;
   timeout.tv_usec = 0;
   cambios = select(NHIJOS + 1,&conjunto,NULL,NULL,&timeout);
/*************************************************************************/
    if (cambios == 0) {
      printf("timeout de select\n");
    }
    else {
      // Comprobar si hay algo en la entrada estándar
      if (FD_ISSET (0, &conjunto)) {
        char* line = NULL;
        size_t size = 0;
        printf("Algo hay en la entrada estándar ... ");
        getline(&line, &size, stdin);
        line[strlen(line)-1] = '\0';
        printf("Leído: %s\n",line);
/*
Comprobar si hay que terminar.
  Si es así (porque se recibe exit), cerrar todos los descriptores de lectura para
  que se genere la señal SIGPIPE para cada hijo.
  Luego, hacer un bucle invocando "wait()" para esperar a que mueran todos los hijos.
*/
/*************************************************************************/
   if(strcmp(line,"exit") == 0){
     for(i = 0; i < NHIJOS; i++){
       close(tpadre[i]);
     }
     while(wait(NULL) == -1);
     exit(0);
   }
/*************************************************************************/
      }
/*
Comprobar si hay algo en alguna tubería. Si es así, leer e imprimir por pantalla
*/
      for (i = 0; i < NHIJOS;  ++i) {
        if (FD_ISSET (tpadre[i], &conjunto)) {
          _t_dato_ dato;
          read(tpadre[i],&dato,sizeof(dato));
          printf("Padre recibe por hijo %d con tiempo  %d\n",dato.index,dato.tiempo);
        }
      }
    }
```

```
    }
  }
}
/*
badr@badr:~/Escritorio/ASO/Ejercicio4/Ejercicio4.1$ ./tub-select.elf
Padre hace select.
Hijo 0 escribe en pipe el tras dormir 10s
Hijo 1 escribe en pipe el tras dormir 10s
timeout de select
Padre hace select.
Hijo 0 escribe en pipe el tras dormir 16s
Hijo 1 escribe en pipe el tras dormir 16s
timeout de select
Padre hace select.
exit
Algo hay en la entrada estándar ... Leído: exit
Hijo 0 escribe en pipe el tras dormir 14s
Hijo 1 escribe en pipe el tras dormir 18s
Write en el hijo -929007344 falla.
Hijo 1: el padre dice que a terminar!!
Hijo 0 escribe en pipe el tras dormir 13s
Write en el hijo -929007344 falla.
Hijo 0: el padre dice que a terminar!!
*/
```

Este programa se debe de ejecutar en terminal, ejemplo de salida:

```
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$ gcc tub-select.c -o tub-select.elf
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$ ./tub-select.elf
Padre hace select.
Hijo 0 escribe en pipe el tras dormir 5s
Padre recibe por hijo 0 con tiempo  5
Padre hace select.
timeout de select
Padre hace select.
timeout de select
Padre hace select.
Hijo 1 escribe en pipe el tras dormir 18s
Padre recibe por hijo 1 con tiempo  18
Padre hace select.
Hijo 0 escribe en pipe el tras dormir 15s
Padre recibe por hijo 0 con tiempo  15
Padre hace select.
Hola
Algo hay en la entrada estándar ... Leído: Hola
Padre hace select.
Hijo 1 escribe en pipe el tras dormir 5s
Padre recibe por hijo 1 con tiempo  5
Padre hace select.
Hijo 1 escribe en pipe el tras dormir 5s
Padre recibe por hijo 1 con tiempo  5
Padre hace select.
Adiós
Algo hay en la entrada estándar ... Leído: Adiós
Padre hace select.
exit
Algo hay en la entrada estándar ... Leído: exit
TERMINANDO:
Hijo 0 escribe en pipe el tras dormir 4s
Hijo 1 escribe en pipe el tras dormir 14s
UPPS. write en hijo 1 falla. Nos vamos!!
Hijo 1: el padre dice que a terminar!!
Hijo 0 escribe en pipe el tras dormir 16s
UPPS. write en hijo 0 falla. Nos vamos!!
Hijo 0: el padre dice que a terminar!!
..
usuarioso:~/aso-jupyter-public/ejercicios/ejercicio4.1$
```

Ejercicio 4 - redireccion.c Escribe un programa que sea capaz de reproducir el funcionamiento de la siguiente orden.

```
ls -l | wc -l
```

Nota: Recuerda el uso de `dup` y `dup2`.

In [ ]:

```
%%writefile redireccion.c
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/wait.h>

int main (int argc, char **argv){
    int pid1, pid2;
    int tuberias[2];
    char buffer[256];
    int fin;

    fin = 0;

    if(pipe(tuberias) == -1){
        perror("pipe");
        exit(0);
    }

    pid1 = fork();

    switch(pid1){
        case -1:
            perror("fork");
            exit(0);
            break;
        case 0:
            printf("Lector de la tuberia %i", getpid());
            close(0);
            dup2(tuberias[0],0);
            close(tuberias[0]);
            close(tuberias[1]);
            execlp("wc","wc","-l", NULL);
            exit(0);
            break;
        default:
            break;
    }

     pid2 = fork();

    switch(pid2){
        case -1:
            perror("fork");
            exit(0);
            break;
        case 0:
            printf("Lector de la tuberia %i", getpid());
            close(0);
            dup2(tuberias[0],0);
            close(tuberias[0]);
            close(tuberias[1]);
            execlp("ls","ls","-l", NULL);
            exit(0);
            break;
        default:
            break;
    }
     close(tuberias[0]);
    close(tuberias[1]);
    wait(NULL);
    printf("fin");

}


/*
```

```
badr@badr:~/Escritorio/ASO/Ejercicio4/Ejercicio4.1$ ./redireccion.elf
0
finbadr@badr:~/Escritorio/ASO/Ejercicio4/Ejercicio4.1$ total 124
-rw-rw-r-- 1 badr badr  1375 may 17 20:36 redireccion.c
-rwxrwxr-x 1 badr badr 17136 may 17 20:36 redireccion.elf
-rw-rw-r-- 1 badr badr  1015 may 17 17:17 tubAnon.c
-rwxrwxr-x 1 badr badr 17168 may 17 17:40 tubAnon.elf
prw-rw-r-- 1 badr badr     0 may 17 20:22 tuberia
-rw-rw-r-- 1 badr badr  2475 may 17 20:23 tubNombre2.c
-rwxrwxr-x 1 badr badr 17504 may 17 20:22 tubNombre2.elf
-rw-rw-r-- 1 badr badr  1072 may 17 18:47 tubNombre.c
-rwxrwxr-x 1 badr badr 17504 may 17 18:36 tubNombre.elf
-rw-rw-r-- 1 badr badr  5646 may 17 20:30 tub-select.c
-rwxrwxr-x 1 badr badr 17736 may 17 20:30 tub-select.elf

*/
/*****************************************************************************/
```

In [ ]:

```
%%bash
gcc redireccion.c -o redireccion.elf
./redireccion.elf
ls -l | wc -l
```