UNIDAD 1. PROGRAMACIÓN MULTIPROCESO

1. CONTROL DE PROCESOS EN LINUX

EJERCICIOS GUIADOS

Para cada uno de los ejercicios guiados debes hacer lo siguiente: escribe el programa en el editor gedit, compilalo y ejecutalo en el terminal y comprueba el resultado obtenido.

Para entregar: crea un documento en Writer donde añadas una captura de pantalla del resultado de cada uno de los ejercicios realizados. Explica brevemente qué hace cada uno de los programas. El documento lo debes entregar en PDF con el nombre "Ejercicios guiados_p1_nombreyapellidos".

EJERCICIO 1. Nombralo como ejemploSystem.c.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  printf("Ejemplo de uso de system():");
  printf("\n\tListado del directorio actual y envío a un fichero:");
  printf("%d",system("ls > ficsalida"));
  printf("\n\tAbrimos con el gedit el fichero...");
  printf("%d",system("gedit ficsalida"));
  printf("\n\tEste comando es erróneo: %d",system("ged"));
  printf("\nFin de programa....\n");
}
```

EJERCICIO 2. Nombralo como ejemploExec.c.

```
#include <unistd.h>
#include <stdio.h>
void main()
{
        printf("Ejemplo de uso de exec():");
        printf("Los archivos en el directorio son:\n");
        execl("/bin/ls", "ls", "-l",(char *)NULL);
        printf("¡¡¡ Esto no se ejecuta !!!\n");
}
```

EJERCICIO 3. Nombralo como ejemploPadres.c.

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

void main(void)
{
   pid_t id_pactual, id_padre;

   id_pactual = getpid();
   id_padre = getppid();

   printf("PID de este proceso: %d\n", id_pactual);
   printf("PID del proceso padre: %d\n", id_padre);
}
```

Creación y ejecución de procesos

EJERCICIO 4. Nombralo como ejemploFork.c.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
void main() {
  pid_t pid, Hijo_pid;
  pid = fork();

  if (pid == -1 ) //Ha ocurrido un error
  {
    printf("No se ha podido crear el proceso hijo...");
    exit(-1);
  }
  if (pid == 0 )  //Nos encontramos en Proceso hijo

   {
     printf("Soy el proceso hijo \n\t
             Mi PID es %d, El PID de mi padre es: %d.\n",
             getpid(), getppid() );
   }
   else    //Nos encontramos en Proceso padre
   {
    Hijo_pid = wait(NULL); //espera la finalización del proceso hijo
    printf("Soy el proceso padre:\n\t
            Mi PID es %d, El PID de mi padre es: %d.\n\t
            Mi hijo: %d terminó.\n",
            getpid(), getppid(), pid);
    }
    exit(0);
}
```

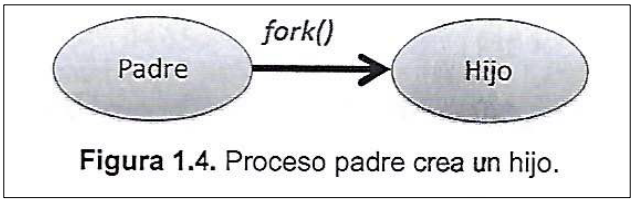

Figura 1.4. Proceso padre crea un hijo.

Comunicación entre procesos

EJERCICIO 5. Nombralo como ejemploWriteRead.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(void)
{
  char saludo[] = "Un saludo!!!\n";
  char buffer[10];
  int fd, bytesleidos;

  fd=open("texto.txt",1);//fichero se abre solo para escritura

  if( fd == -1 )
  {
   printf("ERROR AL ABRIR EL FICHERO...\n");
   exit(-1);
  }

  printf("Escribo el saludo...\n");
  write(fd,saludo, strlen(saludo));
  close(fd); //cierro el fichero

  fd=open("texto.txt",0);//el fichero se abre solo para lectura
  printf("Contenido del Fichero: \n");

  //leo bytes de uno en uno y lo guardo en buffer
  bytesleidos= read(fd, buffer, 1);
  while (bytesleidos!=0){
       printf("%1c", buffer[0]);  //pinto el byte leido
       bytesleidos= read(fd, buffer, 1);//leo otro byte
  }
  close(fd);

}
```

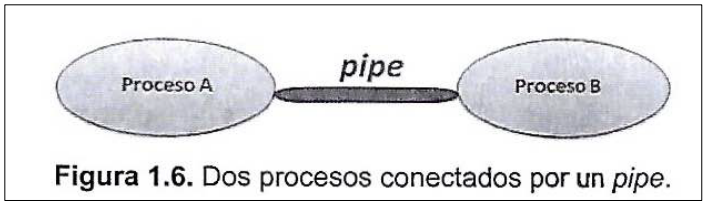

Figura 1.6. Dos procesos conectados por un *pipe*.

EJERCICIO 6. Nombralo como ejemploPipe.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()

 {
  int  fd[2];
  char buffer[30];
  pid_t pid;

  pipe(fd); //se crea el pipe

  pid=fork(); //se crea el proceso hijo

  switch(pid) {
    case -1 : //ERROR
            printf("NO SE HA PODIDO CREAR HIJO...");
            exit(-1);
            break;
    case 0 : //HIJO
            printf("El HIJO escribe en el pipe...\n");
            write(fd[1], "Hola papi", 10);
            break;
    default : //PADRE
            wait(NULL); //espera que finalice proceso hijo
            printf("El PADRE lee del pipe...\n");
            read(fd[0], buffer, 10);
            printf("\tMensaje leido: %s\n",buffer);
            break;
  }
 }
```

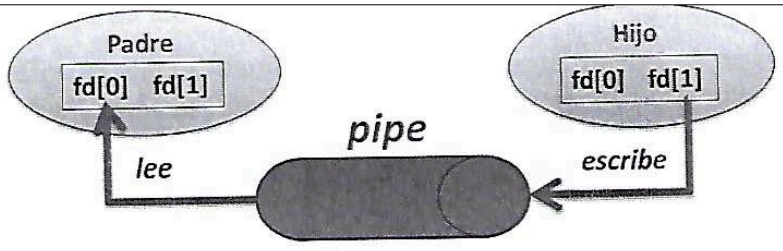

Figura 1.7. Pipe para un proceso hijo que escribe y otro padre que lee.

Cuando el flujo de información va del padre hacia el hijo:

- El padre debe cerrar el descriptor de lectura *fd[0]*.
- El hijo debe cerrar el descriptor de escritura *fd[1]*.

Cuando el flujo de información va del hijo hacia padre ocurre lo contrario:

- El padre debe cerrar el descriptor de escritura *fd[1]*.
- El hijo debe cerrar el descriptor de lectura *fd[0]*.

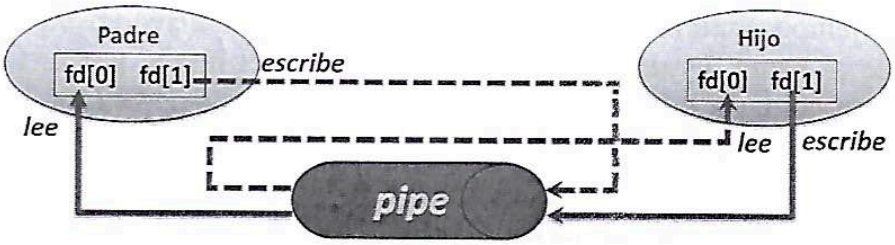

Figura 1.8. Pipe para un proceso padre e hijo que se envían datos.

EJERCICIO 7. Nombralo como ejemploPipe2.c.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
  int fd[2];
  pid_t pid;
  char saludoPadre[]="Buenos dias hijo.\0";
  char buffer[80];

  pipe(fd);   //creo pipe
  pid=fork(); //creo proceso

  switch(pid) {
   case -1 : //ERROR
        printf("NO SE HA PODIDO CREAR HIJO...");
        exit(-1);

   case 0 : //HIJO RECIBE
        close(fd[1]);//cierra el descriptor de entrada
        read(fd[0], buffer, sizeof(buffer)); //leo el pipe
        printf("\tEl HIJO recibe algo del pipe: %s\n",buffer);
        break;

   default ://PADRE ENVIA
        close(fd[0]);
        write(fd[1],saludoPadre,strlen(saludoPadre));//escribo en pipe
        printf("El PADRE ENVIA MENSAJE AL HIJO...\n");
        wait(NULL); //espero al proceso hijo
        break;
 }

  return 0;
 }
```

Sincronización entre procesos

EJERCICIO 8. Nombralo como sincronizar.c.

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
/*-----------------------------------------*/
/* gestión de señales en proceso HIJO      */
void manejador( int segnal )
{
 printf("Hijo recibe señal..%d\n", segnal);
}

/*-----------------------------------------*/
int main()
{
  int pid_hijo;
  pid_hijo = fork(); //creamos hijo

  switch(pid_hijo)
  {
     case -1:
          printf( "Error al crear el proceso hijo...\n");
          exit( -1 );
     case 0:   //HIJO
          signal(SIGUSR1, manejador); //MANEJADOR DE SEÑAL EN HIJO
          while(1) {
          };
     break;
     default: //PADRE envia 2 señales
          sleep(1);
          kill(pid_hijo, SIGUSR1);//ENVIA SEÑAL AL HIJO
          sleep(1);
          kill(pid_hijo, SIGUSR1);//ENVIA SEÑAL AL HIJO
          sleep(1);
     break;
  }
  return 0;
}
```

ACTIVIDADES

Crea un documento en Writer donde se incluya, para cada uno de los ejercicios, una captura de pantalla tanto del código del programa como del resultado obtenido al compilar y ejecutar en el terminal. El documento lo debes entregar en PDF con el nombre "Ejercicios _p1_nombreyapellidos".

EJERCICIO 9. Escribe un programa en C que liste todos los procesos activos y guarde el resultado en un fichero de salida.

EJERCICIO 10. Escribe un programa en C que cree un proceso hijo y un proceso nieto tal y como muestra la siguiente figura:

Al compilar y ejecutar el programa se debe mostrar la siguiente salida (ejemplo):

```
            Soy el proceso NIETO 4486; Mi padre es = 4485
      Soy el proceso HIJO 4485, Mi padre es: 4484.
      Mi hijo: 4486 terminó.
Soy el proceso ABUELO: 4484, Mi HIJO: 4485 terminó.
```

EJERCICIO 11. Escribe un programa en C que cree un proceso (tendremos 2 procesos, uno padre y otro hijo). El programa definirá una variable entera y le dará el valor 6. El proceso padre incrementará dicho valor en 5 y el hijo restará 5. Se deben mostrar los valores en pantalla. A continuación, se muestra un ejemplo de la ejecución:

```
Valor inicial de la variable: 6
Variable en Proceso Hijo: 1
Variable en Proceso Padre: 11
```

EJERCICIO 12. Realiza un programa en C que cree un pipe en el que el hijo envíe un mensaje al padre, es decir la información fluya del hijo al padre. La ejecución debe mostrar la siguiente salida

```
      El HIJO envía algo al pipe.
El PADRE recibe algo del pipe: Buenos dias padre.
```