josemhv
www.wuolah.com/student/josemhv

Modulo_2_SO.pdf

Ejemplos Ejercicios Resueltos Modulo 2

2º Sistemas Operativos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
UGR - Universidad de Granada

```c
/*
Ejercicio 1: Escribir un programa que pueda recibir 0, 1 o 2 argumentos tal que:
$ ./prog → lea de stdin y escriba en stdout
$ ./prog ent.txt → lea de ent.txt y escriba en stdout
$ ./prog ent.txt sal.txt → lea de ent.txt y escriba en sal.txt
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char * argv[]){
    int fdent, fdsal;
    fdent= STDIN_FILENO;
    fdsal= STDOUT_FILENO;

    if(argc == 2 || argc==3){
      if( (fdent=open(argv[1],O_RDONLY)) < 0){
        perror("...");
        exit(1);
      }
    }
    if(argc==3){
      if( (fdsal=open(argv[2],O_WRONLY | O_CREAT ,S_IRWXU)) < 0 ){
        perror("...");
        exit(1);
      }
    }
    if(argc > 3){
      printf("Error: ...");
      exit(1);
    }
    char bloque[512];
    int leidos;

    while( (leidos=read(fdent, bloque, 512*sizeof(char))) > 0){
      if(write(fdsal,bloque,leidos*sizeof(char)) < 0){
        perror("...");
        exit(-1);
      }
    }
    close(fdent);
    close(fdsal);
    return 0;
  }
```

```c
/*
Ejercicio 2: Si es un fichero regular añade permisos de escritura a otros.
Añadir permiso→ OR

  110101101
  000000010
  -------------
  110101111

Quitar permiso→ AND

  110101101
  111111011
  ------------
  110101001
permisos= attrs.st_mode & ~S_IWOTH
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(int argc, char * argv[]){

  if(argc != 2){
      fprintf(stderr, "error : nº de argumentos incorrectos");
      exit(-1);
  }

  struct stat attrs;
  mode_t permisos;

  if( stat(argv[1],&attrs) < 0){
    perror("...");
    exit(1);
  }

  if(S_ISREG(attrs.st_mode)){
    permisos = attrs.st_mode | S_IWOTH;
    if(chmod(argv[1],permisos) < 0){
      perror("Error: el fichero no es regular");
      exit(1);
    }
  }
}
```

```c
/*
Ejercicio 3: Añadir permisos de escritura a otros , a todos los ficheros
contenidos dentro de un cierto directorio que es el pasado por argumento.
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>

int main(int argc, char * argv[]){
  if(argc != 2){
      fprintf(stderr, "error : nº de argumentos incorrectos");
      exit(-1);
  }
  DIR * directorio;
  struct dirent * entrada;
  char ruta[512];

  if( (directorio = opendir(argv[1]) )==NULL ){
    perror(".1.");
    exit(1);
  }

  while( (entrada=readdir(directorio)) !=NULL){
    struct stat attrs;
    mode_t permisos;
    sprintf(ruta, "%s/%s", argv[1], entrada-> d_name );

    if( stat(ruta,&attrs) < 0){
      perror(".2.");
      exit(1);
    }
    if(S_ISREG(attrs.st_mode)){
      permisos = attrs.st_mode | S_IWOTH;
      if(chmod(ruta,permisos) < 0){
        perror("Error: el fichero no es regular");
        exit(1);
      }
    }
  }
  closedir(directorio);
}
```


```c
/*
EjercicioFork1: Crea 5 hijos que muestren un mensaje y el padre muestre otro.
Además el mensaje mostrado por el padre siempre será el último.
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>

int main(){

  int i;

  pid_t pid=1;
  for(i=0; i<5 && pid!=0 ;i++)
    pid=fork();

  if(!pid){
    printf("Soy yo, Soy hijo:)\n");
  }
  else{
    for(i=0; i<5; i++)
      wait(NULL);

    printf("Soy un padre feliz\n");

  }
}
```

Función Fork devuelve 0 en el proceso hijo , el PID del hijo en el proceso padre.

```c
/*
ejercicioFork2:
    $ ./prog entrada.txt salida.txt
    -Se crea un hijo→ leer el contenido de argv[1] y lo escribe argv[2]
    -Espera a que el hijo termine
    -Crea otro hijo que lea argv[2] y lo muestra por pantalla
    -Espera a que termine
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>

int main(int argc, char * argv[]){

  int fdent, fdsal;
  char bloque[512];
  int leidos;
  pid_t hijo=1;
  hijo=fork();
  if(!hijo){
    if( (fdent=open(argv[1],O_RDONLY)) < 0){
      perror("...");
      exit(1);
    }
    if( (fdsal=open(argv[2],O_WRONLY | O_CREAT | O_TRUNC ,S_IRWXU)) < 0 ){
      perror("...");
      exit(1);
    }
    while( (leidos=read(fdent, bloque, 512*sizeof(char))) > 0){
      if(write(fdsal,bloque,leidos*sizeof(char)) < 0){
        perror("...");
        exit(-1);
      }
    }
    close(fdent);
    close(fdsal);
    exit(0);
  }
  wait(NULL);
  hijo=fork();
  if(!hijo){
    if( (fdsal=open(argv[2],O_RDONLY)) < 0){
      perror("...");
      exit(1);
    }
    while( (leidos=read(fdsal, bloque, 512*sizeof(char))) > 0){
      if(write(STDOUT_FILENO,bloque,leidos*sizeof(char)) < 0){
        perror("...");
        exit(-1);
      }
    }
    close(fdent);
    close(fdsal);
    exit(0);
  }
  wait(NULL);
}
```

```c
/*
Ejercicio1Dup2: crear un duplicado del fichero
escribir en el a través de la salida estandar
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>


int main(int argc, char * argv[]){

  int fd;
  if((fd=open("datos.txt",O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU)) <0){
    perror("Error");
    exit(1);
  }
  if(dup2(fd,STDOUT_FILENO)<0){
    perror("Error");
    exit(1);
  }
  printf("Duplicado");

  close(fd);
}
```

```c
/*
Ejercicio1Exec: Haz un programa que haga un hijo que haga un ls
 del directorio que se pasa por argumento y el padre espera a
 que termine el hijo y muestra un mensaje de finalizacion.
  $./prog /home
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>


int main(int argc, char * argv[]){

  if(argc != 2){
    printf("Error: numero de parametros incorrectos");
    exit(1);
  }
  int i;
  pid_t pid=1;
  pid=fork();

  if(!pid){
    if(execlp("ls","ls",argv[1],NULL)==-1){
      perror("Error en el execlp");
      exit(1);
    }
  }
  else{
      wait(NULL);
      printf("Soy el padre y ya he terminado\n");
  }
}
```

```c
/*
Ejercicio1Pipe: cat /etc/passwd | cut -d -f1
 Lecura -> f[0]
 Escritura -> f[1]
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>

int main(){

  int fd[2];

  if(pipe(fd)<0){
    perror("ERROR en el pipe");
    exit(1);
  }
  pid_t pid1=fork();
  if(pid1==0){
    close(fd[0]);
    if(dup2(fd[1],STDOUT_FILENO)<0){
      perror("Error");
      exit(1);
    }
    if(execlp("cat","cat","/etc/passwd", NULL)==-1){
      perror("Error en el execlp1");
      exit(1);
    }
  }
  pid_t pid2=fork();
  if(pid2==0){
    close(fd[1]);
    if(dup2(fd[0],STDIN_FILENO)<0){
      perror("Error");
      exit(1);
    }
    if(execlp("cut","cut","-d:","-f1", NULL)==-1){
      perror("Error en el execlp1");
      exit(1);
    }
  }

  close(fd[0]);
  close(fd[1]);
  wait(NULL);
  wait(NULL);
}
```

```c
/*
Ejercicio Mezcla Pipe,Dup2 y Exec.
Busca en el directorio actual el fichero dado
como argumento, y encuentra el numero de enlaces
duros que tiene
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>
#include <string.h>

int main(int argc, char * argv[]){
  if(argc != 2){
    printf("Error: numero de parametros incorrectos");
    exit(1);
  }
  int fd[2];
  if(pipe(fd)<0){
    perror("ERROR en el pipe");
    exit(1);
  }

  pid_t pid1=fork();
  if(pid1==0){
    close(fd[0]);
    ino_t inodo=0;
    int enc=0;
    DIR * directorio;
    struct dirent * entrada;
    if( (directorio = opendir(".") )==NULL ){
      perror(".1.");
      exit(1);
    }
    while( (entrada=readdir(directorio)) !=NULL  && !enc){
      if(!strcmp(entrada->d_name, argv[1])){
        enc=0;
        inodo = entrada->d_ino;
      }
  }
  if(write(fd[1],&inodo, sizeof(ino_t)) < 0){
    perror("error en el write");
    exit(1);
  }
  close(fd[1]);
  exit(0);
}
```

```c
// Continuacion de Ejercicio Mezcla Pipe,Dup2 y Exec.
  pid_t pid2=fork();
  if(pid2==0){
    close(fd[1]);
    ino_t inodo=0;
    DIR * directorio;
    struct dirent * entrada;
    struct stat st;
    if(read(fd[0],&inodo,sizeof(ino_t)) < 0){
      perror("Error lectura");
      exit(1);
    }
    if((directorio=opendir("."))==NULL){
      perror("Error Apertura");
      exit(1);
    }
    while((entrada=readdir(directorio))!=NULL){
      if(entrada->d_ino==inodo){
        if(stat(entrada->d_name,&st) < 0){
          perror("Error");
          exit(1);
        }
        printf("Enlaces duros: %ld\n",st.st_nlink );
      }
    }
    closedir(directorio);
    close(fd[0]);
    exit(0);
  }
  close(fd[0]);
  close(fd[1]);
  wait(NULL);
  wait(NULL);
}
```

```c
/*
ManejadorSeñales1
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>
#include <string.h>

void manejador(int s) {
  printf("Podria ser hoy, pero va a ser que no ... \n");
}

int main(int argc, char * argv[]){

  struct sigaction sa;
  sa.sa_handler= manejador;

  if(sigaction(SIGINT,&sa,NULL) <0){
    perror("Error en el sigaction");
    exit(1);
  }
  while(1);

}
```

```c
/*
ManejadorSeñales2 : No dejar hijos zombies
*/
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <sys/wait.h>
#include <string.h>

void manejador(int s) {
  wait(NULL);
}

int main(int argc, char * argv[]){

  struct sigaction sa;
  sa.sa_handler= manejador;

  if(sigaction(SIGCHLD,&sa,NULL) <0){
    perror("Error en el sigaction");
    exit(1);
  }
  fork();
}
```

Fcntl → File Control
LK → lock

int fcntl(int fd, int orden(F_GETLK,F_SETLK,F_SETLKW) , struct flock)

```
struct flock{
        short l_type; [F_RDLCK, F_WRLCK, F_UNLCK]
        short l_whence; [SEEK_SET, SEEK_CUR, SEEK_END]
        off_t l_start;
        off_t l_len;
        pid_t pid;
};
```

void* mmap(void* address, size_t length, int prot, int flags, int fd, off_t offset)
ejemplo:

```
memoria= (char*)mmap(0, … , PROT_READ,MAP_SHARED,fd,0)
```

MAP_SHARED → cambios que haga son visibles a otros procesos
PROT_READ → el mapeo es de solo lectura
primer 0 → el kernel se encarga
… → tamaño de la proyeccion (fichero entero se usa de tamañao → st_size de la struct stat)

```
munmap(memoria, tamaño);
```

```
if( memoria == MAP_FAILED) { … }
```

//Falta ejemplo

1 proceso (padre/servidor) crea el fifo(crea un archivo especial con un nombre)
todos los procesos que se vayan a comunicar abren el fifo (open del fichero [RD,WR]) y usa read y write como siempre.

mkfifo(“nombre”, permisos);
unlink(“nombre”);

IPC: pipe(cauces sin nombre) y FIFO(cauces con nombre).

//Falta ejemplo

SISTEMAS OPERATIVOS
Grupo A2 (Lunes 11:30-13:30) Curso 2013/14

Prueba individual del Módulo II: API de Linux

Apellidos y nombre: .....

D.N.I: .....

Preguntas:

I) Parte obligatoria mínima (50%):

Construir un único programa que cree un proceso hijo. Ambos procesos utilizaran un cauce con/sin nombre para comunicarse (el tipo de cauce lo dejo a vuestra elección). Donde el trabajo de cada proceso es:

a) El padre:Lee el directorio actual y para cada archivo regular contenido en el mismo:
   - obtiene e imprime en pantalla su número de inodo junto con el UID del propietario.
   - escribe el nombre del archivo en el cauce.

b) El hijo: Lee del cauce el nombre del archivo escrito por el padre el lo lee entero mediante el mecanismo de proyección de archivos. Tras lo cual, muestra en pantalla su contenido.

II) Ampliación:

a) [15%] El padre se asegura que los archivos cuyo nombre va a pasar al hijo tienen permiso de lectura para el proceso, si no fuese así, lo cambia para que los pueda leer el hijo.

b) [15%] Establecer un manejador para la señal SIGPIPE en el padre en caso de que el proceso hijo finalice inesperadamente de forma que se indique el evento por pantalla.

c) [20%] El acceso al archivo por parte del proceso hijo es exclusivo, es decir, bloqueará el archivo antes de acceder al mismo y lo desbloqueará al finalizar.

Construir el programa necesario para implementar dicha especificación realizando las suposiciones que estime necesarias y que sean lo menos restrictivas posibles.

Observaciones:
- Subir a Tutor el código fuente desarrollado. En el código debe aparecer como comentario tu nombre y apellidos.
- Entregar esta hoja rellena.

Firma: .....

```c
/*
Examen 13/14
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <dirent.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>

void manejador(int s)
{
  printf("Mi hijo ha finalizado de forma inesperada :$\n");
}

int main(){

  int fd[2];
  pid_t pid;

  if(pipe(fd)<0) {
    perror("Error al crear el cauce\n");
    exit(-1);
  }

  if((pid= fork())<0) {
    perror("Error al crear el proceso hijo\n");
    exit(-1);
  }

//padre
  if(pid){
    DIR *directorio;
    struct dirent *entrada;
    struct stat estado;
    struct sigaction senial;
    senial.sa_handler= manejador;

    if(sigaction(SIGPIPE, &senial, NULL)<0){
      perror("ERROORRRRR\n");
      exit(-1);
    }

    if(close(fd[0])<0) {
      perror("Error al cerrar el cauce de lectura en el padre\n");
      exit(-1);
    }

    if((directorio= opendir("."))==NULL) {
      perror("Error al intentar abrir el directorio actual\n");
      exit(-1);
    }
```

```
57      while((entrada= readdir(directorio))!=NULL) {
58        if(stat(entrada->d_name, &estado)<0) {
59          perror("Error al recuperar informacion del fichero\n");
60          exit(-1);
61        }
62
63        if(S_ISREG(estado.st_mode)) {
64          int ino= estado.st_ino;
65          int dev= estado.st_dev;
66          printf("Número de inodo: %d\tUID: %d\n", ino, dev);
67
68          if(estado.st_mode & S_IROTH!=S_IROTH) {
69            mode_t permisos= estado.st_mode|S_IROTH;
70
71            if(chmod(entrada->d_name, permisos)<0)  {
72              perror("No se le pudieron cambiar los permisos al archivo.\n");
73              exit(-1);
74            }
75          }
76
77          if(write(fd[1], entrada->d_name, 256)<0) {
78            perror("Error al escribir en el cauce\n");
79            exit(-1);
80          }
81        }
82      }
83
84      close(fd[1]);
85      closedir(directorio);
86    }
87
88  //Hijo
89    else{
90      int leidos1, leidos2;
91      char nombre[256];
92      int fileDirectory;
93      char bloque[256];
94      struct flock cerrojo;
95      struct stat atributos;
96      char *memoria;
97
98      if(close(fd[1])<0) {
99        perror("Error al cerrar el cauce de escritura en el hijo\n");
100       exit(-1);
101     }
102
103     while((leidos1= read(fd[0], nombre, 256))>0){
104       cerrojo.l_type= F_RDLCK;
105       cerrojo.l_whence= SEEK_SET;
106       cerrojo.l_start= 0;
107       cerrojo.l_len= 0;
```

```
109         if((fileDirectory= open(nombre, O_RDONLY))<0) {
110           perror("Error al recuperar informacion del fichero ");
111           printf("%s\n", nombre);
112           exit(-1);
113         }
114
115         if(fcntl(fileDirectory, F_SETLK, &cerrojo)<0) {
116           perror("Error al poner el cerrojo\n");
117           exit(-1);
118         }
119
120         if(stat(nombre, &atributos)<0){
121           perror("Error al recuperar informacion del fichero ");
122           exit(-1);
123         }
124
125         printf("\nInformacion sobre %s: ", nombre);
126         memoria= (char *) mmap(0, atributos.st_size, PROT_READ, MAP_SHARED, fileDirectory, 0);
127
128         if(write(STDOUT_FILENO, memoria, atributos.st_size)<0){
129           perror("Error en la escritura.\n");
130           exit(-1);
131         }
132
133         munmap(memoria, atributos.st_size);
134         printf("\n");
135         cerrojo.l_type= F_UNLCK;
136
137         if(fcntl(fileDirectory, F_SETLK, &cerrojo)<0) {
138           perror("Error al quitar el cerrojo\n");
139           exit(-1);
140         }
141       }
142
143       close(fileDirectory);
144       close(fd[0]);
145       exit(0);
146     }
147
148     wait(NULL);
149   }
150
```