Departamento de Informática

Grado en Ingeniería Informática

Operating Systems

Communication Exercises

Exercise 1 (Enero. Examen Final. Curso 2010-2011)

The Eratostene sieve is an algorithm to generate a series of consecutive prime numbers. It allows to find all the prime numbers smaller than a given natural number N. To do this, a table is formed with all the natural numbers between 2 and N and the numbers that are not primes are crossed out as follows: when a number is found An integer that has not been crossed out, that number is declared prime, and all its multiples are crossed out. The process ends when the square of the largest number confirmed as prime is greater than N.

This Exercise asks to implement the Eratosthenes sieve with processes and pipes. The algorithm should work as follows:

• A parent creates N children.

• The parent connects with child 0.

• Child i connects with children i-1 and i + 1 forming a pipeline.

• The parent generates a series of consecutive numbers 2, 3, 4, 5,… and sends them one by one to child 0. The sequence ends with the number n-1. At the end of the sequence, the parent will send a -1 to inform the children that they should finish.

• Each child stores the first number it receives in a primo_local variable and prints it on the screen. Later:

- if the number it receives is NOT multiple of primo_local, it forwards it to the next child.
- if the received number is a multiple of primo_local, it filters it and does nothing.

• When a child receives a -1, they pass it on to the next child and finish.

• The father waits for all the children to finish before finishing.

The following table shows an example of how each child created:

	Child 1	**Child 2**	**Child 3**	**Child 4**	**…**
Local prime=	2	3	5	7	
Filtered	4,6,8,10…	9,15,21,27…	25,35…	49…	

SOLUTION

Communication Exercises

a)

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

int tuberia0[2], tuberia1[2];
 int impares, pares, valor;

void Child0(){
        int valor;
       impares=-1;
        close(tuberia0[1]);
        close(tuberia1[0]);
        close(tuberia1[1]);

        do{
          read(tuberia0[0], &valor, sizeof(int));
          impar++;
        }while(valor !=0);
        printf(“impares %d\n”, impares);

}

void Child1(){
        int valor;
        pares=-1;
        close(tuberia1[1]);
        close(tuberia0[0]);
        close(tuberia0[1]);

        do{
          read(tuberia1[1], &valor, sizeof(int));
          par++;
        }while(valor !=0);
        printf(“pares %d\n”, pares);

}
```

```
int main (int argc, char *argv[]) {
        impares = pares = 0;

        pipe(tuberia0);
        pipe(tuberia1);

        if (fork()==0) { /* codigo del Child */
                Child0();
        }else{
                if (fork()==0) { /* codigo del Child */
                        Child1();
                }else{
                       while (0 ¡= scanf(“%d”,&valor)){
                               if (valor == 0)
                                 break;
                               valor = 1;
                               if (valor%2 != 0){
                           write(tuberia0[1], &valor, sizeof(int));
                               }else{
                           write(tuberia1[1], &valor, sizeof(int));
                               }
                        }
                       valor = 0;
                       write(tuberia0[1], &valor, sizeof(int));
                       write(tuberia1[1], &valor, sizeof(int));
                       while(wait(&valor)!=-1);


                }
        }
        return 0;
}
```

Communication Exercises

b)

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

int tuberia0[2], tuberia1[2];
 int impares, pares, valor;

void Child0(){
        int valor;
        impares=-1;
        signal(SIGINT,SIG_IGN);
        close(tuberia0[1]);
        close(tuberia1[0]);
        close(tuberia1[1]);

        do{
          read(tuberia0[0], &valor, sizeof(int));

          impar++;
        }while(valor !=0);
        printf("impares %d\n", impares);
}

void Child1(){
        int valor;
        pares=-1;
        signal(SIGINT,SIG_IGN);
        close(tuberia1[1]);
        close(tuberia0[0]);
        close(tuberia0[1]);

        do{
          read(tuberia1[1], &valor, sizeof(int));
          par++;
        }while(valor !=0);
        printf("pares %d\n", pares);

}
```

```
void manejador(){
        int valor = 0;
         write(tuberia0[1], &valor, sizeof(int));
         write(tuberia1[1], &valor, sizeof(int));
        while(wait(&valor)!=-1);
       exit(0);
}

int main (int argc, char *argv[]) {
        impares = pares = 0;

        pipe(tuberia0);
        pipe(tuberia1);

        if (fork()==0) { /* codigo del Child */
                Child0();
        }else{
                if (fork()==0) { /* codigo del Child */
                        Child1();
                }else{
                        signal(SIGINT,manejador);
                while (1){
                                if (valor == 0)
                                  break;
                                valor = 1;

                                scanf("%d",&valor);
                                valor = 1;
                                if (valor%2 != 0){
                           write(tuberia0[1], &valor, sizeof(int));
                                }else{
                           write(tuberia1[1], &valor, sizeof(int));
                                }
                        }
                }
        }
        return 0;
}
```

c)

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

int tuberia0[2], tuberia1[2];
 int impares, pares, valor;

void Child0(){
          int valor;
         impares=-1;
          close(tuberia0[1]);
          close(tuberia1[0]);
          close(tuberia1[1]);

          do{
            read(tuberia0[0], &valor, sizeof(int));
            impar++;
          }while(valor !=0);
}

void Child1(){
          int valor;
          pares=-1;
          close(tuberia1[1]);
          close(tuberia0[0]);
          close(tuberia0[1]);

          do{
            read(tuberia1[1], &valor, sizeof(int));
            par++;
          }while(valor !=0);

}
```

```
int main (int argc, char *argv[]) {
          impares = pares = 0;

          pipe(tuberia0);
          pipe(tuberia1);

          if (fork()==0) { /* codigo del Child */
                    Child0();
          }else{
                    if (fork()==0) { /* codigo del Child */
                              Child1();
                    }else{
                             int fd=open("f1.txt",O_RDONLY);
                             close(0);
                             dup(fd);
                             close(fd);

                    while (0¡=scanf("%d",&valor)){
                                        if (valor == 0)
                                          break;
                                        valor = 1;

                                        if (valor%2 != 0){
                                  write(tuberia0[1], &valor, sizeof(int));
                                        }else{
                                  write(tuberia1[1], &valor, sizeof(int));
                                        }
                              }
                              valor = 0;
                             write(tuberia0[1], &valor, sizeof(int));
                             write(tuberia1[1], &valor, sizeof(int));
                              while(wait(&valor)!=-1);
                    }
          }
          return 0;
}
```

Exercise 2 : (Junio. Examen Extraordinario. Curso 2009-2010)

a) Code a program in C language that reads numbers from the keyboard and displays the sum of the numbers entered. The following guidelines should be followed:

- There must be 3 processes. A parent process (P), a Child process (H), and a grandchild process (N).

- The parent process (P) creates the Child process and waits for its completion.

- The process Child (H) creates the process grandson (N). Also, it reads keyboard numbers and sends them to a pipe. When the keyboard number 0 is read, it will send a SIGUSR1 signal to the grandson process (N) to terminate its execution.

- The process grandson (N) receives the numbers from the pipe and adds them in a variable. When it receives the SIGUSR1 signal, it must display the result of the addition and finish its execution.

b) Indicate the change or changes that should be added so that the execution of the reading of numbers and the addition is carried out in the background.

c) Could the Child (H) process send the SIGKILL signal to tell the grandchild (N) process that it should display the result of the addition and terminate? Give reasons for the answer.

SOLUTION

a) .

```
int total=0;

void ImprimirResultado()
{
      printf("El total de los números es %d\n",total);
      exit(0);
}
```

```
int main()
{
      int fd[2];  //Descriptores para el pipe
      int pidChild;
      int pidNieto;
      int num;
      int estado;
      struct sigaction a;

      pidChild=fork();
      if(pidChild!=0)  // Proceso padre P
      {
            wait(&estado);
      }
      else
      {
            pipe(fd);
            pidNieto=fork();
            if(pidNieto!=0)  // Proceso Child H
            {
                  scanf("%d",&num);
                  while(num!=0)
                  {
                        write(fd[1],&num,sizeof(num));
                        scanf("%d",&num);
                  }
                  kill(pid,SIGUSR1);
                  wait(&estado);
            }
            else
            {
                  a.sa_handler=ImprimirResultado;
                  a.sa_flags=0;
                  sigaction(SIGUSR1,&a,NULL);
                  while(1)
                  {
                        read(fd[0],&num,sizeof(num));
                        total+=num;
                  }
            }

      }
      return 0;
}
```

b) .

```
int total=0;

void ImprimirResultado()
{
      printf("El total de los números es %d\n",total);
      exit(0);
}

int main()
{
      int fd[2];  //Descriptores para el pipe
      int pidChild;
      int pidNieto;
      int num;
      int estado;
      struct sigaction a;

      pidChild=fork();
      if(pidChild!=0)  // Proceso padre P
      {
            ~~wait(&estado);~~ //En background el padre no espera por la finalización
del Child
      }
      else
      {
            pipe(fd);
            pidNieto=fork();
            if(pidNieto!=0)  // Proceso Child H
            {
                  scanf("%d",&num);
                  while(num!=0)
                  {
                        write(fd[1],&num,sizeof(num));
                        scanf("%d",&num);
                  }
                  kill(pid,SIGUSR1);
                  wait(&estado);
            }
            else
            {
                  a.sa_handler=ImprimirResultado;
                  a.sa_flags=0;
```

```
                    sigaction(SIGUSR1,&a,NULL);
                    while(1)
                    {
                          read(fd[0],&num,sizeof(num));
                          total+=num;
                    }
              }

        }
        return 0;
}
```

c) .

If the Child (H) process sends SIGKILL to the grandson (N) process, N would terminate immediately without allowing time for the result of the sum to be displayed on the screen. Also, due to signal system restrictions, N cannot be armed to receive the SIGKILL signal and therefore could not change the default behavior either. Therefore, the answer is COULD NOT.

Exercise 3

Let be a system controlled by a computer with UNIX as the Operating System. The system to be controlled is made up of a sensor and an actuator. The computer should read the sensor periodically and operate the actuator depending on the sensor value.

The software to control this system consists of two processes called "sensor" and "actuator". The "sensor" process first creates the "actuator" process by establishing a pipe between the two. Then the "sensor" process will call a "read_sensor ()" function with a periodicity of 1 second that will return the sensor value as an integer. Then the "sensor" process will send the "actuator" process the value read through the pipe.

The "actuator" process initially creates a data.dat file. Then it begins to read from the pipeline the data sent by the "sender" process. Next, it does the following with the data: first it calls the “actua (data)” routine, passing it as a parameter the data; and second, it writes the data in the data.dat file (The “actua (data)” routine will activate the actuator depending on the parameter).

It is requested to write in C the processes "sensor" and "actuator" clearly specifying the calls to the system.

Note: It is assumed that the “leer_sensor ()” and “actuador” routines are already written.

.

SOLUTION:

```
/* *****  Código programa sensor *********** */
#include <signal.h>
#define STDIN      0
#define STDOUT     1
#define TRUE       1
main()
{
    int valor, nula();
    int fd[2];
    pipe(fd);
    if (fork() != 0)
    {
        while (TRUE)
        {
            signal(SIGALRM, nula);
            alarm(1);
            pause();
            valor = leer_sensor();
            write(fd[1], &valor, sizeof(int));
        }
    }
    else
    {
        close(STDIN);
        dup(fd[0]);
        close(fd[1]);
        close(fd[0]);
        execl("actuador", "actuador", 0);
        exit(1);
    }
}

int nula() {
    return(0);
}

/* ******* Código del programa actuador ******* */
#define STDIN 0
#define O_WRONLY 1
#define TRUE 1
main()
{
    int valor;
    int fd;
    fd = open("datos.dat", O_WRONLY);
    while (TRUE) {
        read (STDIN, &valor, sizeof(int));
        actua(valor);
        write(fd, &valor, sizeof(int));
    }
}
```

EXERCISE 4

Eratostene sieve is a algorithm of generating a series of consecutive prime numbers. In this exercise you are required to implement the Eratostene sieve with ***processes and pipes***. The algorithm works in the following way :

- A father creates N children.
- The father connects to the child 0.
- Child i connects to children i-1 and i+1.
- The father generates a series of consecutive numbers 2, 3, 4, 5,..... and sends them one by one to child 0. The sequence is terminated by the number -1.
- Each child stores the first number to receive in a variable local_prime. Subsequently, if the number it receives is not a multiple of local_prime, it forwards it to the next child.
- When a child receive a -1, it forwards it to the next children and terminates
- The father waits for the termination of all children before it exits.

SOLUTION:

```
#define N 10

int main()
{
  int p[N][2];
  int i;
  // CREATE PIPES
  for (i=0;i<N;i++)
    pipe(p[i]);

  for (i=0;i<N;i++){
    if (!fork()) {
      int local_prime, y;
      close(p[i][1]);
      close(p[i+1][0]);

      // FIRST NUMBER TO BE RECEIVED IS A PRIME NUMBER
      read(p[i][0], &local_prime, 4);
      printf(" CHILD %d received PRIME NUMBER %d\n", getpid(), local_prime);

      while (1) {
           read(p[i][0], &y, 4);

           // FORWARD A NUMBER ONLY IF IT IS NOT MULTIPLE OF THE LOCAL PRIME
NUMBER
           if ((y%local_prime) &&(i < N-1))
               write(p[i+1][1], &y, 4);
           // FINISH WHEN RECEIVING -1
           if (y == -1)
               break;
      }
      exit(0);
    }
  }

  // SENDING A STREAM OF NUMBERS
  for (i=2;i<N*10;i++)
     write(p[0][1],&i,4);
  // SENDING -1 TO TERMINATE
  i=-1;
  write(p[0][1],&i,4);

  // WAIING FOR THE CHILDREN TO TERMINATE
  for (i=0;i<N;i++){
    int status,pid;
    pid=wait(&status);
    printf(" CHILD %d terminated. \n", pid);
  }
}
```

EXERCISE 5

Write a program that uses a pipe to implement a critical section. The program creates a pipe and then makes a fork. The Child executes an infinite loop in which it reads a character from the pipe's input, writes a message, sleeps 2 seconds, and writes a character on the pipe's output. The father does the same.

SOLUTION

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

```
int  main(void) {
   int fildes[2];   /* pipe para sincronizar */
   char c;          /* caracter para sincronizar */

   pipe(fildes);
   write(fildes[1], &c, 1);   /* necesario para entrar en la seccion
critica la primera vez */
   if (fork() == 0)  {          /* proceso Child */
            for(;;) {
              read(fildes[0], &c, 1);  /* entrada seccion critica */
              // Seccion critica
              printf ("El Child entra en seccion critica\n");
              sleep (2);  //espero para que se vea que el padre no entra
              printf ("El Child sale de la seccion critica\n");
              write(fildes[1], &c, 1);  /* salida seccion critica */
              sleep (random()%2); //Espero para que no siempre entre el
mismo
            }
    } else {            /* proceso padre */
         for(;;) {
           read(fildes[0], &c, 1);   /* entrada seccion critica */
           // Seccion critica
              printf ("El padre entra en seccion critica\n");
              sleep (2);  //espero para que se vea que el Child no entra
              printf ("El padre sale de la seccion critica\n");
           write(fildes[1], &c, 1);  /* salida seccion critica */
           sleep (random()%2); //Espero para que no siempre entre el
mismo
          }
   }
 }
```

EXERCISE 6

Escriba un programa en C que implemente el pipeline: “ls | wc”.

SOLUTION

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

int main(void){
   int fd[2];
   pid_t pid;

   if (pipe(fd) < 0)  {
      perror("pipe");
      exit(-1);
  }
  pid = fork();
  switch(pid)  {
      case -1:   /* error */
         perror("fork");
         exit(-1);
       case 0:    /* proceso Child ejecuta ``ls'' */
          close(fd[0]); /* cierra el pipe de lectura */
          close(STDOUT_FILENO); /* cierra la salida estandar */
          dup(fd[1]);
          close(fd[1]);
          execlp("ls","ls",NULL);
          perror("execlp");
          exit(-1);
     default:  /* proceso padre ejecuta ``wc'' */
          close(fd[1]); /* cierra el pipe de escritura */
          close(STDIN_FILENO); /* cierra la entrada estandar */
          dup(fd[0]);
          close(fd[0]);
          execlp("wc","wc",NULL);
          perror("execlp");
        }
}
```

EXERCISE 7

Write a C program that will fork and connect Parent and Child with a pipe. The Child writes a message to the father and the latter writes it on the screen (STDOUT). To avoid race problems, Parent and Child must sleep 2 seconds.

SOLUTION

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

#define ESPERALECTURA 1
#define ESPERAESCRITURA 2
```

```
int main () {
        int             fd[2];
        pid_t           pid_Child;
        char            cadena [] ="Hola soy el Child.\n";
        char            buffer [80];
        int             num_bytes_leidos;

        /* Establecemos la tuberia  */
        pipe (fd);

        if ((pid_Child = fork ())== -1) {
          perror ("fork");
          exit (1);
        }
        if (pid_Child == 0 ) {
           /* El Child cierra el descriptor de entrada */
          close (fd[0]);
           /* El Child escribe en la tuberia */
          sleep (ESPERAESCRITURA);
          write (fd[1], cadena, sizeof (cadena));
          printf ("Fin escritura\n");
          exit (0);
        }
        else { /* El padre cierra el descriptor de salida */
          close (fd[1]);
          /* El padre lee de la tuberia */
          sleep (ESPERALECTURA);
          num_bytes_leidos = read (fd[0], buffer, sizeof (buffer));
          printf ("La cadena recibida por el padre es:%s:", buffer);
        }
        return (0);
}
```

EXERCISE 8

Write a C program that allows 2 programs (prog_esc and prog_lec), which by default use standard input and output to write or read the data they use, to communicate through a pipe created by the parent of both processes.

The reader program requests a string for STDIN and writes it. The writer just writes a string to the STDOUT standard output.

SOLUTION

--------------- Programa principal. Comunicador padre/Child ---------------------------------

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

int main () {
        int             fd[2];
        pid_t           pid_Child;
        /* Establecemos la tuberia  */
        pipe (fd);
        if ((pid_Child = fork ())== -1) {
                perror ("fork");
                exit (1);
        }
        if (pid_Child == 0 ) { /* el Child */
        /*  Cambio la salida estándar por la de la tuberia */
                dup2 (fd[1],1);
                close (fd[0]);
        /* prog_esc escribe sobre la salida estándar, que sera la tuberia 
*/
                execlp ("./prog_esc", "./prog_esc", NULL);
        }
        else { /* padre*/
        /*  Cambio la entrada estándar por la de la tuberia */
                dup2 ( fd[0],0);
                close (fd[1]);
        /* prog_lec lee de la entrada estándar, que sera la tuberia */
                execlp ("./prog_lec", "./prog_lec", NULL);
        }
        return (0);
 }
```

--------------- Programa lector   ---------------------------------

```
#include <stdio.h>

int main() {
    char buffer[90];

    scanf("%s",buffer);
    printf ("He leido:%s:\n",buffer);
 }
```

--------------- Programa escritor ---------------------------------

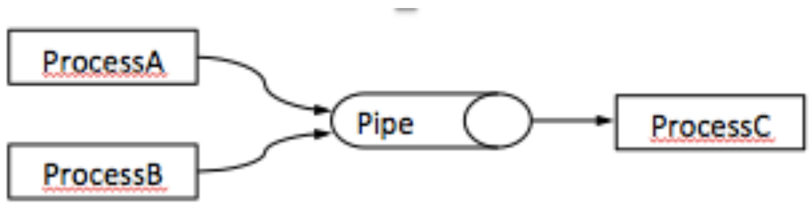
```
#include <stdio.h>

int main (){

    printf ("hola123456\n");
 }
```

Exercise 9

Write a C program that creates three processes. Two of them write the standard output in a pipe and the third reads the standard input from the pipe.



SOLUTION

```
#include <stdio.h>
int main( void )
{
    int my_pipe[2];
     int pid1, pid2;
    /* Processp1 is the parent and creates the pipe */
     if (pipe(my_pipe) < 0) {
        perror(“Error: I can’t create the pipe”) ;
        exit(0);

    }

    /* Processp2 */
switch ((pid1=fork()) {

        case -1: perror("Error creating new process ") ;

                 /* pipe closed */

                 close(my_pipe[0]) ;

                 close(my_pipe[1]) ;
             exit(0) ;

                 break ;

     case 0: /* Processchild, ProcessB */

                /* close read pipe */

                close(my_pipe[0]) ;

                /* code of ProcessB */
            /* code for writting in the pipe using my_pipe[1] */

                break ;

        default: /* ProcessC creation*/

                 switch ((pid2 = fork()) {

                     case -1: perror("Error creating new process ") ;

                              close(Pipe[0]) ;

                              close(Pipe[1]) ;

                              /* The previous process is killed*/

                              kill(pid1, SIGKILL) ;

                              exit(0) ;

                     case 0:  /* Processchild (C) reads from pipe */

                              close(Pipe[1]) ;

                              /* code for reading from the pipe*/

                              break ;

                     default: /* parent (B) writes in the pipe */
```

```
                                    close(my_pipe[0]) ;

                                    /* code for writing in the pipe*/
                               break ;

                    }

}
```

Exercise 10

1. Compile and execute the programs in directory "Pipe-2" in the following order:

   oneprocesspipe

   fatherchildpipe

   childrenpipe

   redirpipe

2. Explain shortly what each program does.

3. What happens if you uncomment the line 39 (//close(p[0]);) from childrenpipe.c program? Discover why and explain shortly.