



UADY

**FACULTAD DE
MATEMÁTICAS**

"Luz, Ciencia y Verdad"

Programa Computacional

Tuberías e Hilos Posix

Usado en la asignatura de Programación de Sistemas

Autor:

Enrique Ayala Franco

Agosto-Diciembre 2020

Programa computacional

Tuberías e hilos Posix

Descripción

En los sistemas operativos modernos, la creación de procesos y la comunicación entre ellos es fundamental para la correcta operación del sistema y para obtener una mejor eficiencia. Adicionalmente, los hilos facilitan la ejecución concurrente de tareas con lo cual se optimiza aún más el funcionamiento de las aplicaciones.

El programa computacional desarrollado ilustra el manejo y la creación de procesos, señales, hilos y la comunicación entre procesos ellos mediante tuberías, en un sistema operativo Linux/Unix, utilizando el estándar Posix. Los procesos creados simulan a dos usuarios que entablan una conversación y se envían datos de un proceso al otro, de manera similar a como funciona el comando talk (un chat).

Es requisito indispensable el uso de tuberías e hilos para comunicar fluidamente a los procesos y se deben crear dos ventanas de terminal para apreciar la conversación de forma adecuada (mediante el comando xterm). En la figura 1 se ilustra la interacción de los procesos e hilos y el funcionamiento del sistema.

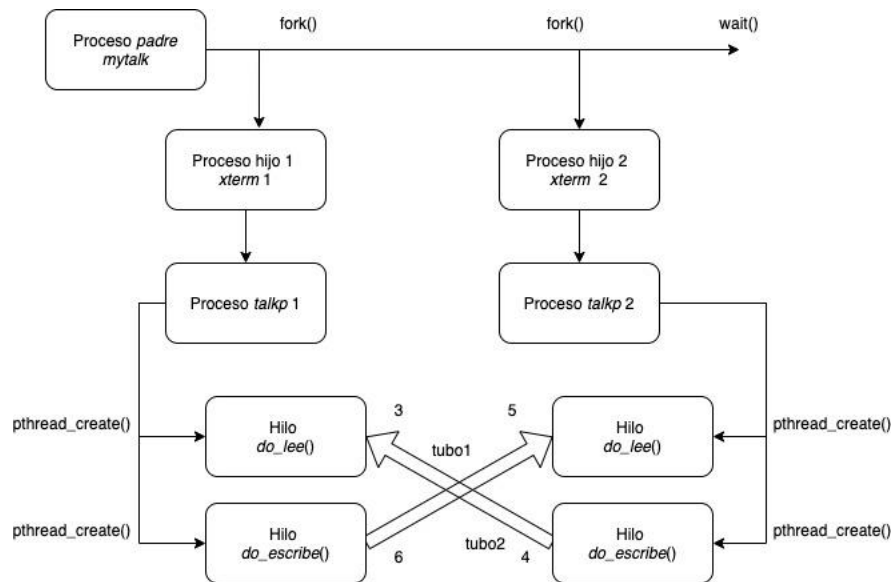


Figura 1 Árbol de procesos e hilos del sistema.

Se proporciona el código fuente del programa **mytalk** y del programa **talkp**, el cual es invocado desde mytalk, y a la par, se emplea **xterm** para abrir dos ventanas de terminal desde las cuales se escriben y se reciben los mensajes.

Las instancias del programa talkp reciben por parámetro descriptores de tuberías para establecer comunicación entre ellos.

La interfaz del programa es en línea de comando y se debe ejecutar en primer lugar el programa **mytalk** el cual crea las dos instancias del programa **talk**.

Características:

- Lenguaje de programación: C estándar.

Relación con el programa del curso:

Unidad 3. Sistemas Operativos

Objetivo: Analizar y desarrollar algún subsistema de un sistema operativo.

3.3. Subsistemas del Sistema Operativo

3.3.1. Administración de procesos, Administración de Dispositivos de I/O, Administración de memoria, Administración de archivos.

Manual de Usuario

Requisitos para la ejecución

Sistema Operativo Linux, Unix o Macintosh.

Memoria RAM de 1GB mínimo.

Operación

Copiar el archivo ejecutable principal mytalk.c (se proporcionan los ejecutable para sistema Macintosh), en un directorio e iniciar el programa, el cual invocará al programa talkp.c.

Iniciar el programa y los procesos

>./mytalk

```
MacBook-Pro-de-Enrique:sol_ada9_hilos Enrique$ ./mytalk
Ids de tuberías utilizadas:
4 3
6 5

Esperando por finalización de hijos
Ctrl+C terminar y cerrar todo
```

Figura 2 Ejecución del programa principal.

El programa principal genera dos ventanas y dos procesos talkp, como se ve en la figura 3. Desde las terminales, los procesos hijos intercambian mensajes.

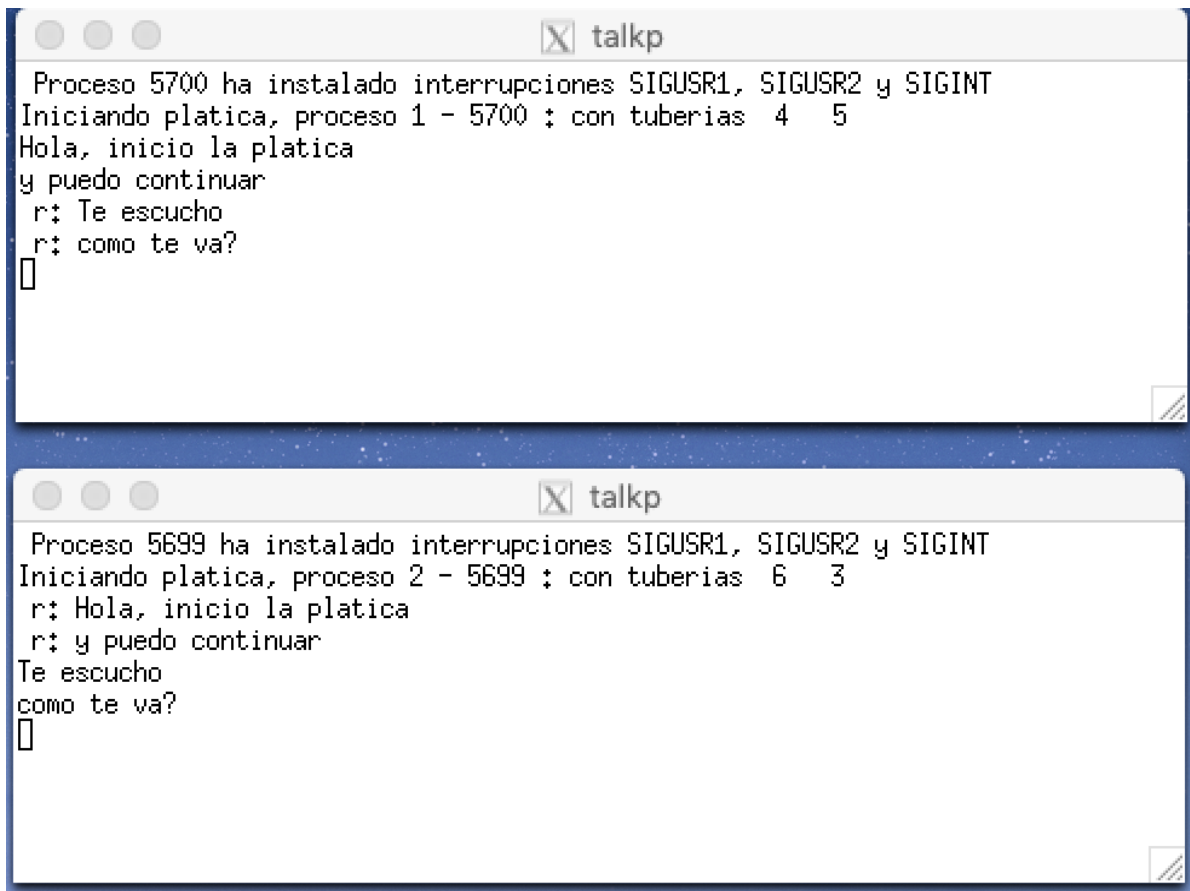


Figura 3 Terminales generadas con xterm, e inicio de procesos talkp que intercambian mensajes.

Verificación de los identificadores de procesos

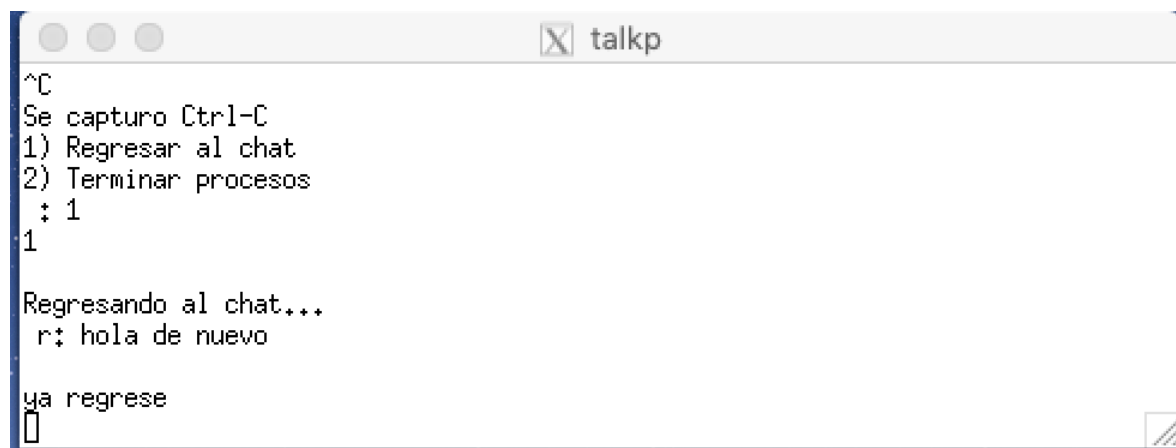
En la figura 4 se utilizó la orden **ps -fea** para listar los procesos activos. Se observan los procesos generados desde el sistema. Los identificadores de terminales son diferentes para los procesos, por ejemplo, los ids 6311 y 6310 están asociados a ttys001 y ttys002. También se ven los parámetros recibidos, los cuales corresponden a los extremos de las tuberías creadas. Los procesos xterm ejecutan de manera persistente y con parámetros a los procesos talkp.

PID	TTY	TIME	CMD
6298	ttys000	0:00.03	login -pfl Enrique /bin/bash -c exec -la bash /bin/bas
6299	ttys000	0:00.05	-bash
6306	ttys000	0:00.00	./mytalk
6307	ttys000	0:00.04	xterm -hold -e ./talkp 2 6 3
6308	ttys000	0:00.05	xterm -hold -e ./talkp 1 4 5
6311	ttys001	0:00.01	./talkp 1 4 5
6310	ttys002	0:00.01	./talkp 2 6 3
6312	ttys003	0:00.03	login -pf Enrique
6313	ttys003	0:00.02	-bash
6319	ttys003	0:00.00	ps -a

Figura 4 Lista de procesos en ejecución generados por el sistema.

Generación de señales (ejecución asíncrona)

Mediante el uso de señales, se puede generar una interrupción (Ctrl-C) y entonces se despliega un menú con opciones. Es un ejemplo de programación con mecanismos de ejecución asíncronos (ver figura 5). Se puede terminar el programa o regresar a la conversación en el punto exacto en que se dejó.



```
^C
Se capturo Ctrl-C
1) Regresar al chat
2) Terminar procesos
: 1
1

Regresando al chat...
r: hola de nuevo

ya regrese
```

Figura 5 Invocación de función de forma asíncrona mediante señales.

Código fuente

En la carpeta se encuentran los archivos **mytalk.c** y **talkp.c**, los cuales implementan funciones para realizar la comunicación entre procesos con tuberías.

Archivo mytalk.c

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

void p1(int t1out, int t2in) {
    char st1out[10], st2in[10];
    sprintf(st1out, "%2d", t1out);
    sprintf(st2in, "%2d", t2in);

    // crear nueva terminal de control para P1
    // parametros de talkp: out (escribir), in (leer)
    if (execlp("xterm", "xterm", "-hold", "-e", "/talkp", "1", st1out, st2in, 0) < 0)
        perror("Fallo la función exec\n");

    printf("Termino p1\n");

    exit(0);
}

void p2(int t2out, int t1in) {
    char st2out[10], st1in[10];
    sprintf(st2out, "%2d", t2out); //cambiar a texto
    sprintf(st1in, "%2d", t1in);

    // crear nueva terminal de control para P2
    // parametros de talkp: out (escribir), in (leer)
    if (execlp("xterm", "xterm", "-hold", "-e", "/talkp", "2", st2out, st1in, 0) < 0)
        perror("Fallo la función exec\n");

    printf("Termino p2\n");
    exit(0);
}

int main(void) {
    int pid1, pid2, tubo1[2], tubo2[2];

    pipe(tubo1); //crear dos tuberías
    pipe(tubo2);

    printf("Ids de tuberías utilizadas: \n");
    printf(" %d %d \n", tubo1[1], tubo1[0]);
    printf(" %d %d \n", tubo2[1], tubo2[0]);
```

```

if ((pid1= fork())==0) { // creacion de proceso si == 0 es copia del Hijo

    //close(tubo1[1]);
    p2(tubo2[1],tubo1[0]);

}
else { // Padre
    if ((pid2= fork())==0) { // si == 0 es copia del segundo Hijo

        //close(tubo1[0]);
        p1(tubo1[1],tubo2[0]);

    }
    else{ //proceso del padre
        int status1, status2;
        printf("\nEsperando por finalizacion de hijos\n Ctrl+C terminar y cerrar todo\n");
        waitpid(pid1, &status1, 0);
        waitpid(pid2, &status2, 0);
        printf("Soy el padre y termino\n");
        exit(0);
    }
}

return 0;
}

```

Archivo talkp.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pthread.h>

#define BLKSIZE 1024

//prototipos de funciones
void installusrhandlers();
void usr1handler(int s);
void usr2handler(int s);
void* do_lee(void* commfd);
void* do_escribe(void* commfd);

void usr1handler(int s)
{
    fprintf(stderr, "SIGUSR1 signal caught\n");
}

void usr2handler(int s)
{
    fprintf(stderr, "SIGUSR2 signal caught\n");
}

```

```

void ctrl_handler(int s)
{
    char buf[50];
    int bytesread;
    //printf("\nSe capturo Ctrl-C\n");
    write(STDOUT_FILENO, "\nSe capturo Ctrl-C\n", 20);
    // reinstalando manejadores
    //installusrhandlers();
    //printf("1) Regresar al chat\n2) Terminar procesos\n : ");
    write(STDOUT_FILENO, "1) Regresar al chat\n2) Terminar procesos\n : ", 45);
    //int sn;
    //scanf(" %2d ", &sn); //leer caracter desde stdin
    bytesread = read(STDIN_FILENO, buf, 10);
    // printf("lei %d ", bytesread);

    if(buf[0]=='1')
    {
        fflush(stdin); // limpiar buffer
        printf("\nRegresando al chat...\n");
    }
    else if(buf[0]=='2'){
        write(STDOUT_FILENO, "\nEl proceso termino la platica\n", 31);
        sleep(3);
        kill(getppid(), SIGKILL); //terminar al proceso padre (xterm)

    }
    else { printf("Opcion no valida!\n"); }
}

void installusrhandlers()
{
    struct sigaction newact;

    newact.sa_handler = usr1handler; /* nuevo usr1 handler */
    sigemptyset(&newact.sa_mask); /* no se bloquean señales */
    newact.sa_flags = 0;
    if (sigaction(SIGUSR1, &newact, (struct sigaction *)NULL) == -1) {
        perror("No pudo instalar manejador SIGUSR1 ");
        return;
    }

    newact.sa_handler = usr2handler; /* nuevo handler */
    if (sigaction(SIGUSR2, &newact, (struct sigaction *)NULL) == -1) {
        perror("No pudo instalar manejador SIGUSR2 ");
        return;
    }
    newact.sa_handler = ctrl_handler; /* nuevo handler */
    if (sigaction(SIGINT, &newact, (struct sigaction *)NULL) == -1) {
        perror("No pudo instalar manejador SIGINT\n"); //ctrl-c
        return;
    }

    fprintf(stderr,
        " Proceso %ld ha instalado interrupciones SIGUSR1, SIGUSR2 y SIGINT\n",
        (long)getpid());
}

/* funcion-hilo para leer desde tuberia */
void* do_lee(void* commfd)
{

```



```

int comunfd = *((int*)commfd);
ssize_t bytesread;
ssize_t byteswritten;
char buf[BLKSIZE];

/*lee desde el socket */
while( (bytesread = read(comunfd, buf, BLKSIZE)) > 0) {

    /* y Escribir en pantalla */
    write(STDOUT_FILENO, " r: ", 4);
    byteswritten = write(STDOUT_FILENO, buf, bytesread);
    if (bytesread != byteswritten) {
        fprintf(stderr,
            "Error al escribir %ld bytes, %ld bytes escritos\n",
            (long)bytesread, (long)byteswritten);
        break;
    }

} //fin while

pthread_exit(NULL);
} // fin do_lee

/* funcion-hilo para escribir a la tuberia*/
void* do_escribe(void* commfd)
{
    int comunfd = *((int*)commfd);
    ssize_t bytesread;
    ssize_t byteswritten;
    char buf[BLKSIZE];

    /*lee desde teclado */
    while(1) {
        bytesread = read(STDIN_FILENO, buf, BLKSIZE);
        if ( (bytesread == -1) && (errno == EINTR) )
            fprintf(stderr, "No pudo leer de la entrada estandar\n");
        else
            if (bytesread <= 0) break;
        else {
            /* Escribir a la tuberia */
            byteswritten = write(comunfd, buf, bytesread);

            if (byteswritten != bytesread) {
                fprintf(stderr,
                    "Error al escribir %ld bytes, %ld bytes escritos\n",
                    (long)bytesread, (long)byteswritten);
                break;
            }
        }
    }

} //fin while

pthread_exit(NULL);
} // fin do_escribe

int main (int argc, char *argv[]) {
    // parametros
    int tout=0;
    int tin=0;

```

```

int np=0;

if (argc != 4) {
    fprintf(stderr, "No se recibieron numeros de tuberias %s\n", argv[0]);
    exit(1);
}

//manejadores de señales
installusrhandlers();

np = atoi(argv[1]); //numero de proceso
tout = atoi(argv[2]); //numero de tubo - escribir
tin = atoi(argv[3]); //numero de tubo - leer

printf("Iniciando platica, proceso %d - %d : ", np, getpid() );
printf("con tuberias %2d %2d\n", tout, tin );

char *estado;
int thr_id1; /* threads ID para nuevos hilos */
int thr_id2;
pthread_t p_thread1; /* estructuras thread */
pthread_t p_thread2;

/* crear nuevos threads */

thr_id2 = pthread_create(&p_thread2, NULL, do_escribe, (void*)&tout );
thr_id1 = pthread_create(&p_thread1, NULL, do_lee, (void*)&tin );

/* el padre espera por la terminacion de los threads */
pthread_join(p_thread2, (void**)&estado);
pthread_join(p_thread1, (void**)&estado);
printf("Terminada %s la hebra \n", estado);
printf("Terminada %s la hebra \n", estado);

exit(0);
}

```