UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)
DOCUMENTO DEL SISTEMA DE CALIDAD

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
6	NOMBRE DE LA PRÁCTICA	**Comunicación entre procesos "Tuberías"**	2

1. INTRODUCCIÓN

Aunque muchas tareas pueden realizarse en procesos aislados, la gran mayoría requieren la intervención de más de un proceso. Para que dichos procesos cooperantes lleven a buen término una tarea común es necesario algún tipo de comunicación entre ellos. Linux ofrece la función ***pipe( )*** para abrir un canal de comunicación entre dos procesos. A lo largo de esta práctica se estudiará la forma en que la función ***pipe()*** trabaja.

2. OBJETIVO (COMPETENCIA)

El alumno aprenderá y practicará la forma de intercambiar información entre procesos diferentes, empleando los mecanismos de tuberías que proporciona el sistema operativo Linux.

3. FUNDAMENTO

El sistema operativo Linux permite que procesos diferentes intercambien información entre ellos. Para ello proporciona diferentes mecanismos que pueden utilizarse en función de las características de la comunicación:

1. Para procesos que se están ejecutando bajo el control de una misma máquina: *tuberías*, *semáforos*, *memoria compartida* y *colas de mensajes*.
2. Para procesos que se ejecutan en máquinas separadas: *sockets*.

Tuberías

Una tubería **(*pipe)*** se puede considerar como un canal de comunicación entre dos procesos. Los mecanismos que se utilizan para manipular tuberías son los mismos que para archivos, con la única diferencia de que la información de la tubería no se almacena en el disco duro, sino en la memoria principal del sistema.

Así pues, este mecanismo de comunicación consiste en la introducción de información en una tubería por parte de un proceso (similar a la escritura en un archivo de disco). Posteriormente otro proceso extrae la información de la tubería (similar a la lectura de información almacenada en un archivo de disco) de forma que los primeros datos que se introdujeron en ella son los primeros en salir. Este modo de funcionamiento se conoce como **FIFO** (**F**irst **I**n, **F**irst **O**ut; el primero en entrar es el primero en salir).

La comunicación mediante tuberías es de tipo ***half-duplex***, es decir, en un instante dado, la comunicación solamente puede tener lugar en un sentido. Si se quiere que un proceso **A** pueda simultáneamente enviar y recibir información de otro **B**, en general se debe recurrir a crear dos tuberías, una para enviar información desde **A** hacia **B** y otra para enviar desde **B** hacia **A**.

Clasificación de tuberías

Se pueden distinguir dos tipos de tuberías dependiendo de las características de los procesos que pueden tener acceso a ellas:

1. **Sin nombre**: Solamente pueden ser utilizadas por los procesos que las crean y por los descendientes de éstos.
2. **Con nombre o *fifo***: Se utilizan para comunicar procesos entre los que no existe ningún tipo de parentesco.

Además, cualquier sistema operativo Lunix garantiza que una tubería puede contener en cualquier momento un mínimo de 4096 bytes.

Tuberías sin nombre

Al igual que un archivo en disco se crea con la instrucción *open*, las tuberías sin nombre también han de ser creadas. Para tal fin se dispone de la llamada *pipe* al sistema.

```
#include <unistd.h>
int pipe (int descriptor[2]);
```

Si la llamada funciona correctamente devuelve el valor 0 y crea una tubería sin nombre. En caso contrario, devuelve –1 y deja en *errno* el código del error. Cada elemento del array *descriptor* se utiliza como si fuera un descriptor de un archivo de disco, correspondiendo el elemento *descriptor[0]* con la salida de la tubería (para escritura) y *descriptor[1]* con la entrada (para lectura). Así pues, el acceso a la tubería para comunicar información entre dos procesos tiene lugar como sigue:

1. Introducción de datos en la tubería: escritura en el "archivo" *descriptor[1].*
2. Lectura de datos de la tubería: lectura del "archivo" *descriptor[0]*.

Una característica importante de estas tuberías es que puesto que los descriptores de archivos se heredan de padres a hijos tras una llamada a ***fork()***, un hijo de un proceso puede utilizar las tuberías sin nombre que su padre hubiera creado antes de crear al hijo. En resumen, y tal como se muestra en el ejemplo siguiente, **para que un proceso padre y su hijo se puedan comunicar mediante una tubería sin nombre, ésta debe haber sido creada antes del nacimiento del proceso hijo**. Esta idea se puede extender a toda la descendencia de un proceso y no solamente a los procesos hijo.

```
#include<string.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
void main()
{
    int filedes[2]; //Descriptor de archivo doble
   int pid;
   char cadena1[50]="Informacion pasada por el pipe\n";
   char entrada[255];
   printf("Ejemplo de pipe\n");
   printf("Proceso padre pid=%d\n",getpid());
   pipe(filedes);
   pid=fork();
   switch(pid)
   {       case -1: printf("Error");
           break;
           case 0:  printf("Hijo Enviando Cadena\n");
                    write(filedes[1],cadena1,strlen(cadena1));
                    exit(0);
           break;
          default:  printf("Preparando para leer informacion del hijo\n");
          close(filedes[1]);
          read(filedes[0],entrada,sizeof(entrada));
          printf("Datos leidos de la pipa:  %s\n",entrada);
          exit(0);
    }
}
```

4. PROCEDIMIENTO (DESCRIPCIÓN)	
A) EQUIPO NECESARIO	**MATERIAL DE APOYO**
Computadora con sistema operativo Linux.	Practica 6 del manual de sistemas operativos Plan 2003-1

1.- Escriba y verifique el funcionamiento del **Ejemplo 1.**
2.- Para este ejercicio utilizaremos dos programas de tuberías con nombre. El primer programa se llama Lector y el segundo Escritor.

Programa Lector.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>
#include <fcntl.h>
#define NOMBREFIFO "mififo"
#define TAM_BUF 80
#define TRUE 1
int main(void)
{
int fp;
char buffer[TAM_BUF];
int nbytes;
mknod(NOMBREFIFO,S_IFIFO|0660,0);
while(TRUE)
{
fp=open(NOMBREFIFO,O_RDONLY);
nbytes=read(fp,buffer,TAM_BUF-1);
buffer[nbytes]='\0';
printf("Cadena recibida: %s \n",buffer);
close(fp);
}
return 0;
}
```

Nota: Una vez escrito el programa, deberás ejecutarlo y dejarlo corriendo sobre esta terminar, luego escribe el programa Escritor que se presenta a continuación.

Programa Escritor .c

```
#include <sys/types.h>
#include <fcntl.h>
#define NOMBREFIFO "mififo"
int main(int argc, char *argv[])
{
int fp;
   int result=1;
   if(argc!=2)
   printf("uso: %s cadena \n", argv[0]);
   else if ((fp=open(NOMBREFIFO,O_WRONLY))==-1)
   perror("fopen");
   else
   {
   write(fp,argv[1],strlen(argv[1]));
   close(fp);
   result=0;
   }
   return result;
   }
```

Nota 1: Para ejecutar este programa primero abre otra terminal y ejecuta el programa escribiendo lo siguiente: **./Escritor HolaCrayola**

Nota 2: Observa lo que tienes ahora en la terminar que esta ejecutando el proceso Lector. **¿Qué sucedió?**

3. Investiga como se utiliza el mando **mkfifo** desde el shell de Linux (no como una función de C bajo Linux) y deberás realizar un ejemplo de este mando **frente a tu maestra(o)**. Recuerda que para Linux una tubería no es más que un archivo por lo tanto las puedes borrar de la misma forma en que borras un archivo.

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno demostrará su habilidad para enviar señales mediante el uso de tuberías.

6. ANEXOS

7. REFERENCIAS

http://148.231.82.20/~so