

Pontificia Universidad Javeriana

Taller 02: Fork()



Pontificia Universidad
JAVERIANA
Colombia

Tomas Alejandro Silva Correal
Juan Guillermo Pabon Vargas

Sistemas Operativos

25 de Marzo de 2025

Índice:

Objetivos.....	3
Introducción.....	3
Desarrollo.....	3
Resultados.....	11
Conclusiones.....	17

Objetivos

El objetivo de este taller es aplicar conceptos sobre procesos y comunicación entre procesos para mejor entender los temas y cómo funcionan para reforzar nuestro conocimiento sobre el tema de procesos.

El siguiente documento mostrará el desarrollo de la creación de procesos y la comunicación entre ellos. También mostrará los resultados del programa realizado.

Introducción

En este taller se van a crear dos ficheros que contienen dos diferentes arreglos con valores separados por espacios, y un programa principal en C que reciba como argumentos los 2 ficheros y además 2 valores que indican cuántos elementos hay en cada fichero.

En el programa se realizan 4 procesos fork con diferentes funciones. Posterior a esto, se mostrará el resultado del programa dado valores predeterminados en los ficheros.

Desarrollo

inicialmente, se explicarán los parámetros de entrada asignados. Se generaron 2 archivos de prueba, como es correspondiente, con nombres “archivo01” y “archivo02”, donde se encuentran los arreglos a manejar. El archivo01 cuenta con un total de 9 valores separados por comas (dando así el valor N1 - el cual define la cantidad de valores), y el archivo02 cuenta con 6 valores en total (definiendo así N2 de parámetro de entrada).

A continuación, se mostrarán los contenidos de ambos archivos, con los valores correspondientes a cada uno:



```
GNU nano 7.2 archivo01.txt
1 3 4 6 78 4 5 2 6
```

Imagen 1: fichero archivo01.txt, contiene un arreglo de valores separados por espacios



```
GNU nano 7.2 archivo02.txt
7 14 21 28 34 59
```

Imagen 2: fichero archivo02.txt, contiene un arreglo de valores separados por espacios

Fuera de lo ya mencionado, se mostrará en si el fichero principal, el cual contiene los procesos y funcionalidades requeridas para el manejo de los procesos mencionados en el enunciado.

```

/*****
 * PONTIFICIA UNIVERSIDAD JAVERIANA
 * Tomas Alejandro Silva Correal - Juan Pabon Vargas
 * Taller02 - Taller de Fork()
 *****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
typedef struct{
    char nombre[50];
    int resultado;
} Resultados;

int main(int argc, char *argv[])
{
    //Obtiene tamaño de arreglos
    int N1=atoi(argv[1]);
    int N2=atoi(argv[3]);
    //Los procesos:
    int GrandHijo;
    int Segundohijo;
    int Primerhijo;
    int Padre;
    //Obtiene los datos de nombre de archivo
    char *archivo00=argv[2];
    char *archivo01=argv[4];

```

```

/*-----
SECCIÓN DE LECTURA DE ARCHIVOS DADO EL NOMBRE DEL ARCHIVO COMO PARAMETRO DE ENTRADA
-----*/
//LECTURA DEL PRIMER ARCHIVO
FILE *arc00;
arc00 = fopen(archivo00, "r");
int *ArF00= calloc(N1, sizeof(int));
//Llena un arreglo temporal con "calloc" para los datos del archivo
for(int i=0; i<N1; i++) {
    fscanf(arc00, "%d", &ArF00[i]);
}
fclose(arc00);

//LECTURA DEL SEGUNDO ARCHIVO
FILE *arc01;
arc01 = fopen(archivo01, "r");
int *ArF01= calloc(N1, sizeof(int));
//Llena un arreglo temporal con "calloc" para los datos del archivo
for(int i=0; i<N1; i++) {
    fscanf(arc01, "%d", &ArF01[i]);
}
fclose(arc01);
/*-----
SECCIÓN DE PROCESOS CON FORK
-----*/
//Para el proceso padre (el cual usa pipe()), se crea el pipe
int pipefd[2];
if (pipe(pipefd) == -1) {
    perror("pipe");
    exit(1);
}

```

```

    //Crea variables de suma donde se guarda cada resultado de proceso
    int sumA=0;
    int sumB=0;
    int sumC=0;
    int sumD=0;

    //GrandHijo como nuevo proceso
    sleep(1);
    GrandHijo=fork(); //Genera un proceso hijo por fork()
    if(GrandHijo == 0) {
        Resultados R;
        R.resultado=0;
        close(pipefd[0]); // Deja solo para escribir
        strcpy(R.nombre, "Sumatoria de primer arreglo");
        for(int i=0; i<N1; i++) {
            R.resultado += ArF00[i];
        }
        //escribe el resultado en el pipe
        write(pipefd[1], &R.resultado, sizeof(int));
        close(pipefd[1]);
        exit(0);
    }

    //SegundoHijo como nuevo proceso
    sleep(1);
    Segundohijo=fork(); //Genera un proceso hijo por fork()
    if(Segundohijo == 0) {
        Resultados R;
        R.resultado=0;
        close(pipefd[0]); // Deja solo para escribir
        for(int i=0; i<N2; i++) {
            R.resultado += ArF01[i];
        }
    }

```

```

    }
    //escribe el resultado en el pipe
    write(pipefd[1], &R.resultado, sizeof(int));
    close(pipefd[1]);
    exit(0);
}

//PrimerHijo como nuevo proceso
sleep(1);
Primerhijo=fork(); //Genera un proceso hijo por fork()
if(Primerhijo == 0){
    Resultados R;
    R.resultado=0;
    close(pipefd[0]); // Deja solo para escribir
    for(int i=0; i<N1; i++){
        R.resultado += ArF00[i];
    }
    for(int i=0; i<N2; i++){
        R.resultado += ArF01[i];
    }
    //escribe el resultado en el pipe
    write(pipefd[1], &R.resultado, sizeof(int));
    close(pipefd[1]);
    exit(0);
}

sleep(1);
//Padre como nuevo proceso
Padre=fork(); //Genera un proceso hijo por fork()
if (Padre == 0) {
    close(pipefd[1]); // Deja para solo leer
    Resultados R;
    R.resultado=0;
    int resultado;
    for (int i = 0; i < 3; i++) {
        read(pipefd[0], &resultado, sizeof(int));
        if(i == 0)
            printf("Resultado de la sumatoria del primer arreglo %d\n", resultado);
        else if(i == 1)
            printf("Resultado de la sumatoria del segundo arreglo %d\n", resultado);
        else{
            printf("Resultado de la sumatoria de ambos arreglos %d\n", resultado);
        }
    }
    //Cierra el pipe
    close(pipefd[0]);
    exit(0);
}
// Padre espera a todos
for (int i = 0; i < 4; i++) {
    wait(NULL);
}
//Libera memoria dinamica por medio de "Free"
free(ArF00);
free(ArF01);
return 0;
}

```

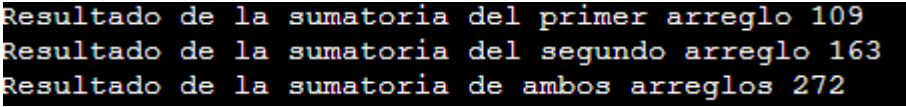
Imagen 3: Programa en C con los 4 procesos y comunicación entre ellos.

De lo visto en el programa, hay algunos elementos a tener en cuenta. El programa está dividido en dos secciones; “Lecturas de archivos” y “Fork()”, cada una cubriendo lo que su título indica (Iniciando con la carga de datos de parte de los archivos, y pasando a las funcionalidades de tipo fork(), que se solicitaban en el enunciado). hay un total de 4 procesos fork(), listados con nombres GrandHijo, SegundoHijo, PrimerHijo y Padre, todos utilizan el comando sleep() junto a fork() para generar procesos hijos que luego se

comunicaran por medio de un pipe, el cual permitirá escritura de los tres primeros procesos, y la lectura del cuarto, el cual se encarga de la escritura de los resultados en pantalla.

Resultados

A continuación se muestra el resultado del programa usando el comando:
`./taller_procesos 9 archivo01.txt. 6 archivo02.txt`



```
Resultado de la sumatoria del primer arreglo 109
Resultado de la sumatoria del segundo arreglo 163
Resultado de la sumatoria de ambos arreglos 272
```

Imagen 4: Resultado del programa C con los ficheros y valores como entrada

Viendo los resultados, y tomando en cuenta los valores de entrada, es necesario explicar el camino que se tomó para llegar a este punto. Después de guardar los datos de tamaño y valores correspondientes, se inicia el proceso `GrandHijo`, el cual suma los valores del primer arreglo, tras esto entran los procesos `SegundoHijo` y `PrimerHijo` que se encargan de la suma total de elementos en el segundo arreglo, al igual que la suma final de ambos arreglos. Todos los procesos utilizan el comando `write()` como forma de pasar los datos a la última de las funcionalidades, la cual imprime los valores tras leerlos del pipe.

Conclusiones

En el taller se crearon 4 procesos usando `fork()`. Cada vez que se usa el `fork()`, se crea un nuevo proceso hijo que es una copia del proceso padre. Cada proceso fue creado en diferentes formas para que puedan cumplir una función diferente.

Como los procesos son relacionados, se pudo haber usado un pipe sin nombre porque solo pueden establecer comunicación entre procesos relacionados. En el caso de este taller, se decidió usar un named pipe que también sirve como para que procesos relacionados puedan comunicarse, pero también sirve para que dos procesos que no están relacionados puedan comunicarse.

Ya que el proceso padre creó el named pipe, los procesos hijos pueden usar el pipe para comunicarse con los otros procesos para llegar a los resultados.

Con el taller se pudo aprender como usar el `fork()` para crear nuevos procesos y cómo usar un pipe para que los procesos puedan comunicarse entre sí.