9.0

Practica 4
Sistemas Operativos

Universidad Autónoma Metropolitana – Azcapotzalco
15 de junio de 2018

Índice

1. Introducción

La mayoría de los sistemas operativos identifican a un proceso mediante un identificador único lo cual se conoce como pid (process identifier) el cual es un numero entero. Cuando un proceso crea un proceso con "fork."este proceso se le llama comúnmente "proceso padrez al nuevo proceso creado como "proceso hijo". Cuando se crea el proceso hijo este puede obtener sus recursos directamente del sistema operativo, o bien, obtenerlos a partir de su padre ya sea teniendo una partición de estos o compartirlos (como lo es la memoria). Además un proceso puede cargar un nuevo programa gracias a la llamada execv esta carga un archivo binario en memoria por lo que destruye el proceso donde este se llame y inicia la ejecución de este nuevo programa[1].

Cuando se crea un nuevo proceso existen dos formas en la que se ejecutan, ahora los dos programas, la primera de estas se refiere a que el proceso padre se ejecuta concurrentemente con su hijo que en este caso es cuando se ejecuta la función fork sin tomar en cuenta ninguna señal que podría emitir el proceso hijo, en la segunda forma se da cuando el proceso una vez que llama a la función fork el proceso padre espera hasta que el hijo termine su ejecución para que este continué con su propia ejecución, esto se puede lograr con la función wait.

2. Desarrollo

Para el desarrollo de la practica se realizo un programa el cual tendrá la opción de elegir una cierta operación, una vez elegida se creara un proceso con fork el cual posteriormente ejecutara un execv para llamar al programa que puede realizar dicha operación, en la llamada de execv se pasaron dos parámetros los cuales son los operandos que el programa procesara. Cabe señalar que el llamado a fork es muy necesario ya que si execv tiene éxito en llamar al programa que se necesite, entonces este proceso sera remplazado por el programa que se llame por lo cual es claro que esto no se puede hacer en el programa principal. A continuación en el código 1 se muestra la implementación de lo explicado anteriormente.

Código 1: Programa Principal.

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

int main()
{
    int statusChild = 0;
    int fid = 0;
    char * envp[] = {NULL};
    char ** argv;
    char * exeOp[] = {  "./suma",
                        "./resta",
                        "./mult",
                        "./div",
                        "./perm",
                        "./comb"};
    int op = -1;
    while(1){
        printf( "Seleccione una opcion:\n"
                "\t1.- Suma con dos operandos\n"
                "\t2.- Resta con dos operandos\n"
                "\t3.- Multiplicacion con dos operandos\n"
                "\t4.- Division con dos operandos\n"
                "\t5.- Permutaciones sin repeticion\n"
                "\t6.- Combinaciones\n"
                "\tPresione '0' para salir\n");
        scanf("%d", &op);
        if(op == 0) break;
        if(op >= 1 && op <= 6){
            int n, m;
            if (op == 5 || op == 6) printf("Asuncion: n >= m\n");
            printf("Introduzca el primer operando 'n'\n");
            scanf("%d", &n);
            printf("Introduzca el segundo operando 'm'\n");
            scanf("%d", &m);
            int size_n = (n==0) ? 1 : ((int) log10(n)) + 1;
            int size_m = (m==0) ? 1 : ((int) log10(m)) + 1;
            argv = (char**) malloc(sizeof(char*) * 2);
            argv[0] = (char*) malloc(size_n);
            argv[1] = (char*) malloc(size_m);

            fid = fork();
            if(fid == 0){
                sprintf((char*)(&argv[0][0]), "%d", n, size_n);
                sprintf((char*)(&argv[1][0]), "%d", m, size_m);
                int ret = execve(exeOp[op - 1], argv, envp);
                if(ret == -1) printf("Error\n");
                exit(0);
            }else{
                waitpid(fid, &statusChild, 0);
            }

            free(argv[0]);
            free(argv[1]);
            free(argv);

        }else{
            printf("Introduzca una opcion valida\n\n");
        }
    }

    return 0;
}
```

Posteriormente se ejecuto el programa haciendo que este se

bloqueara en ciertos instantes, usando scanf, con la finalidad de poder observar en el árbol de procesos como se creaba el nuevo proceso, para que después este fuese remplazado por el programa que se llamaba y una vez que se obtenían los resultados este mismo proceso finalizaba teniendo de nueva cuenta únicamente al programa principal. Esta se puede apreciar de mejor forma en la Figura 1 en la que se tienen los arboles de procesos obtenidos en los instantes antes mencionados.

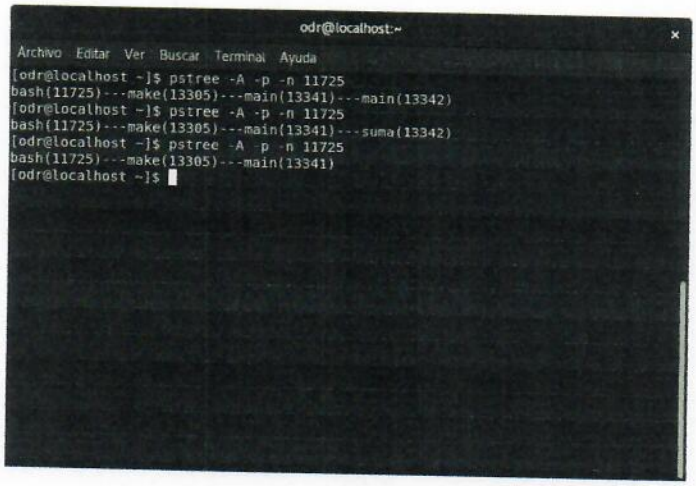

Figura 1: Arboles de Procesos.

3. Conclusión

Es importante poder controlar la ejecución que se da en los procesos pues de esta manera se podrían diseñar mas programas de esta forma, aunque para esto se tiene que conocer bien lo que hacen las funciones que tienen que ver con procesos, ya que de otra forma se pueden tener errores que no tienen que ver con el manejo del lenguaje, si no mas bien tendrian que ver con las llamadas al sistema que hacen estas.

Referencias

1. . G. G. Silberschatz A., Galvin P., *Fundamentos de Sistemas Operativos*, 7.ª ed. McGraw-Hill, 2006, pp. 83–86.

Falta código