Reporte: Creación de Estructuras con una Calculadora

Diego Ruiz Mora — 2202000335

13-Septiembre-2023

1 Primera Estructura

Para generar la estructura de la *imagen 1* tendremos que hacer un pequeño cambio al código donde tendremos que generar solamente un hijo en el código de calculadora y quitar la condición en la que se hacia la petición de una operación a ejecutar. Esto se ve en la *imagen 2*, que después de la creación del hijo con la instrucción **idf=fork()**, al hijo le asignamos el nuevo código que en este caso será el de el proceso *suma* como lo hicimos en la tarea, solamente que solo se creará a un hijo en este caso.

Por otro lado, para el caso de los procesos hijos es algo parecido, en general notaremos que es la misma modificación para el resto de los códigos, donde buscaremos que cada uno de los procesos tenga un hijo, al que le asignara el código del siguiente proceso, ese siguiente proceso creará un hijo y a su vez le dará el código respectivo del siguiente, así de manera simultanea.

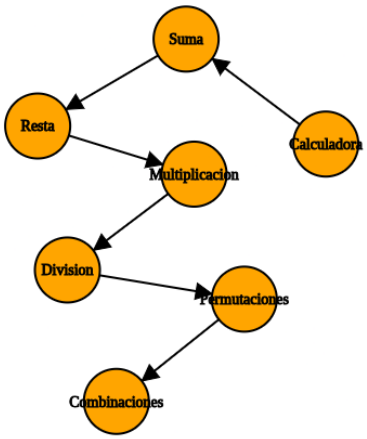

Imagen 1: Estructura

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

int main(){
    char*argv[4]={"","","",NULL};
    char*envp[]={NULL};
    int idf=0;
    char num1[5];
    char num2[5];

    printf("\nDame el primer numero: ");
    scanf("%s",&num1);
    argv[1]=num1;
    printf("\nDame el segundo numero: ");
    scanf("%s",&num2);
    argv[2]=num2;


    idf=fork();
    if(idf==0){
        argv[0]="./suma";
        execve("./suma", argv, envp);
    }
    else{
        wait(NULL);
    }
    //argv[0]="./calculadora";
    //execve("./calculadora", argv, envp);
    sleep(5);
    return 0;
}
```

Imagen 2: Código de calculadora.c

Para ver de manera más fiel de lo que hablamos, tendremos que recurrir a la *imagen 3*, donde veremos el código respectivo para la *suma* y como es que creamos un hijo y después le asignamos a este hijo el código de la siguiente operación.

Y como bien dijimos esto se repetirá hasta que se llegue al último proceso que es *combinaciones* que es el que no se verá modificado en su código. También es importante denotar que en los demás códigos incorporemos una instrucción **wait(NULL)** que nos permitirá esperar a que los procesos hijos dejen de ejecutar para que los procesos padres terminen. Esto nos permitirá visualizar el árbol de procesos.

Veremos en la *imagen 4* que es correcta esta forma de realizar el código ya que la estructura generada es la deseada, donde cada uno de los procesos tiene un hijo que es el siguiente proceso a ejecutar. De igual forma al ver el código ejecutado notaremos que los valores que pasan son los mismo, ya que esto no se modificó, aunque si se podría lograr modificando el vector de argumentos en cada proceso antes de asignar el código al siguiente hijo.

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


int main(int argc, char** argv){
    char*envp[]={NULL};
    int n1, n2,idf;
    n1= atoi(argv[1]);
    n2= atoi(argv[2]);

    printf("\n%d+%d=%d",n1,n2, n1+n2);
    printf("\n");

    idf=fork();
    if(idf==0){
        argv[0]="./resta";
        execve("./resta", argv, envp);
    }
    wait(NULL);

}
```

Imagen 3: Código de suma.c

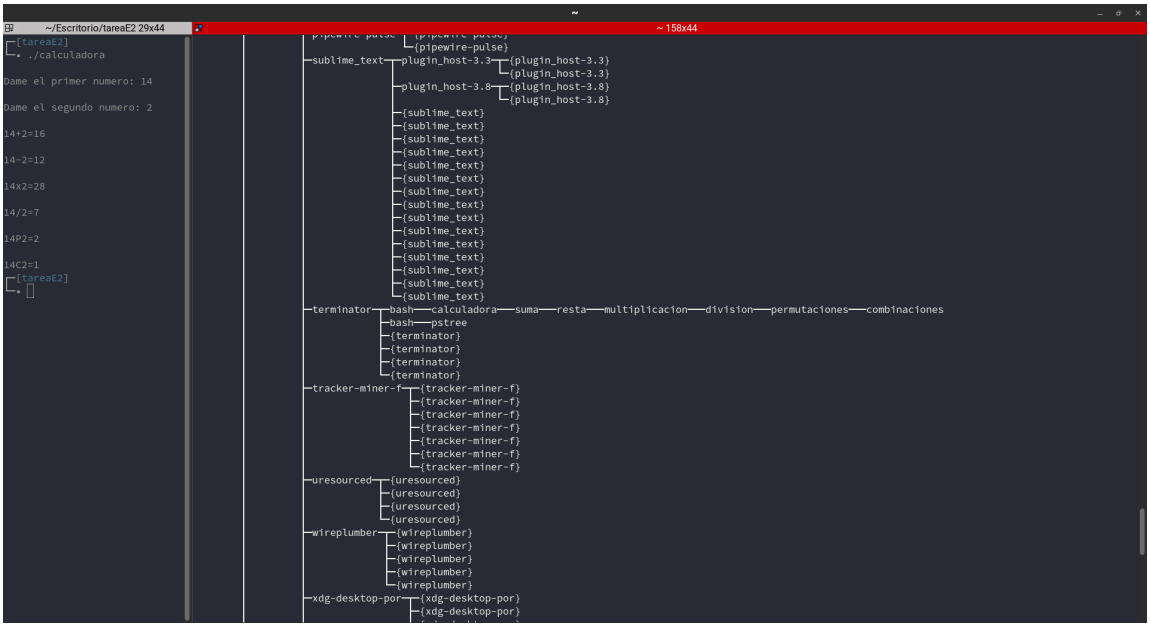

Imagen 4: Estructura generada

2 Segunda Estructura

Para realizar la siguiente estructura mostrada en *imagen 5* modificaremos solamente el código de *calculadora* ya que es el único que genera hijos, de hecho la estructura es muy parecida a lo que se llamaría un *abanico*.

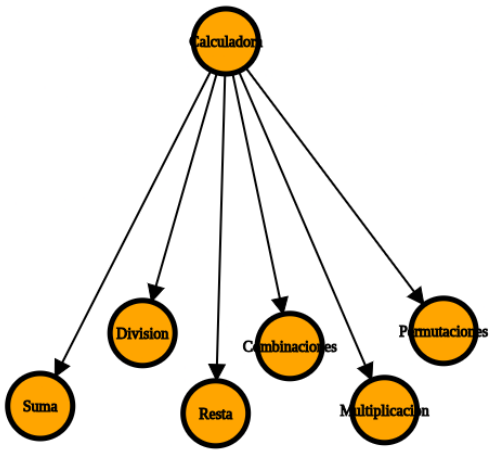

Imagen 5: Estructura

Revisaremos el código de la *imagen 6*, respectivo a *calculadora* , donde encontraremos seis instrucciones **fork()**, cada una asignada a un identificador diferente para no tener problemas al momento de crear cada uno de los hijos, claro que pudimos ahorrarnos algunas, pero es más sencillo de esta forma.

Dentro de cada una de las instrucciones encontraremos la asignación del nombre del proceso al primer elemento del vector de argumentos, seguido encontraremos la asignación del código a este respectivo hijo generado lineas antes. Básicamente el proceso es el mismo, podríamos incluso facilitar la creación con el uso de una estructura de repetición.

Por último agregamos una instrucción **sleep()** para lograr que la ejecución tarde un poco en ejecutar y nos permita observar los resultados de la ejecución, lo cual es posible observar en la *imagen 7* , donde es claro que la estructura generada es la deseada.
Es importante ver el como se entregan los resultados en la terminal, ya que estos no se resuelven de manera ordenada debido a las condiciones de carrera, a pesar de que los procesos se hayan ejecutado con cierta lógica de orden dentro del código.

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

int main(){
    char*argv[4]={"","","",NULL};
    char*envp[]={NULL};
    int idf,idf1, idf2, idf3, idf4, idf5;
    char num1[5];
    char num2[5];

    printf("\nDame el primer numero: ");
    scanf("%s",&num1);
    argv[1]=num1;
    printf("\nDame el segundo numero: ");
    scanf("%s",&num2);
    argv[2]=num2;


    idf=fork();
    if(idf==0){
        argv[0]="./suma";
        execve("./suma", argv, envp);
    }

    idf1=fork();
        if(idf1==0){
            argv[0]="./division";
            execve("./division", argv, envp);
        }

    idf2=fork();
        if(idf2==0){
            argv[0]="./resta";
            execve("./resta", argv, envp);
        }

    idf3=fork();
        if(idf3==0){
            argv[0]="./combinaciones";
            execve("./combinaciones", argv, envp);
        }

    idf4=fork();
        if(idf4==0){
            argv[0]="./multiplicacion";
            execve("./multiplicacion", argv, envp);
        }

    idf5=fork();
        if(idf5==0){
            argv[0]="./permutaciones";
            execve("./permutaciones", argv, envp);
        }

    //argv[0]="./calculadora";
    //execve("./calculadora", argv, envp);
    sleep(5);
    return 0;
}
```

Imagen 6: Código calculadora.c

Imagen 7: Estructura generada