

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.

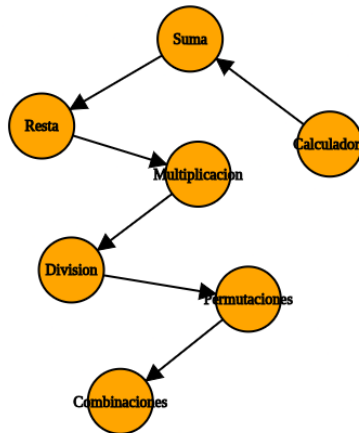
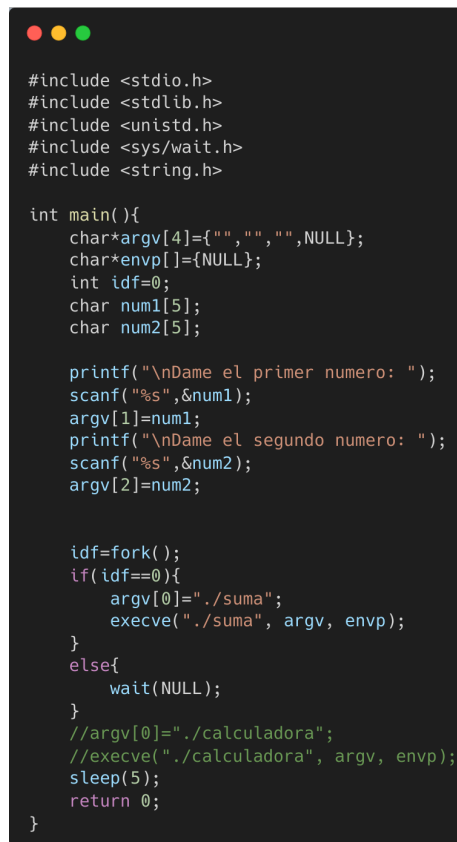


Imagen 1: Estructura



```

#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.

```

#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

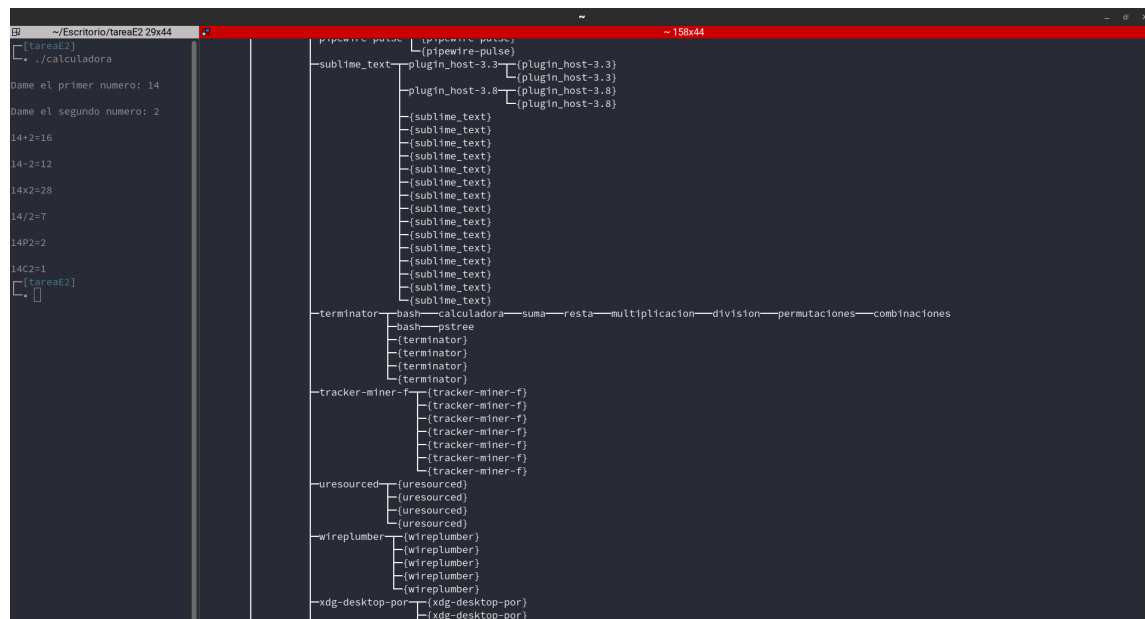


Imagen 4: Estructura generada

2 Segunda Estructura

Para realizar la siguiente estructura mostrada en *imagen 5* modificaremos solamente tres códigos que serán: *calculadora.c*, *division.c* y *resta.c*. A los dos últimos solamente haremos un hijo como en el caso de la estructura pasada y en caso de *calculadora* crearemos cuatro hijos.

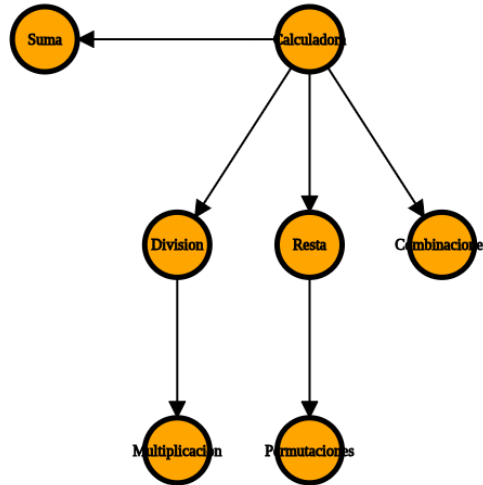


Imagen 5: Estructura

Primero revisaremos el código respectivo a *calculadora* donde, como bien lo mencionamos tenemos la creación de cuatro hijos para los procesos que son el primer nivel del árbol. Cada uno tiene un **idf** distinto, ya que lo que buscamos es identificar cada uno de ellos, ya que al asignarle el código tendremos que ser muy específicos para no incurrir en duplicados.

Es más sencillo hacerlo de esta manera, ya que si reutilizamos el identificador del proceso se vuelve una tarea complicada de hacer.

Por otro lado, notaremos que no incorporamos una llamada a sistema **wait(NULL)** que nos ayudaría a que este proceso no termine hasta que los hijos terminen, esto nos permitiría visualizar la estructura y que no tengamos conflictos de que se terminen unos antes que otros, pero curiosamente, funciona mejor sin esta instrucción, entonces la omitimos en este caso.

Después de ello tendremos que modificar los procesos que generan otros hijos, que en este caso son *division* y *resta* para lo cuál solamente tenemos que hacer un hijo al final del código, que es de suma importancia para que no se repita el código o impresiones de datos.

```

#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;
    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);
    }

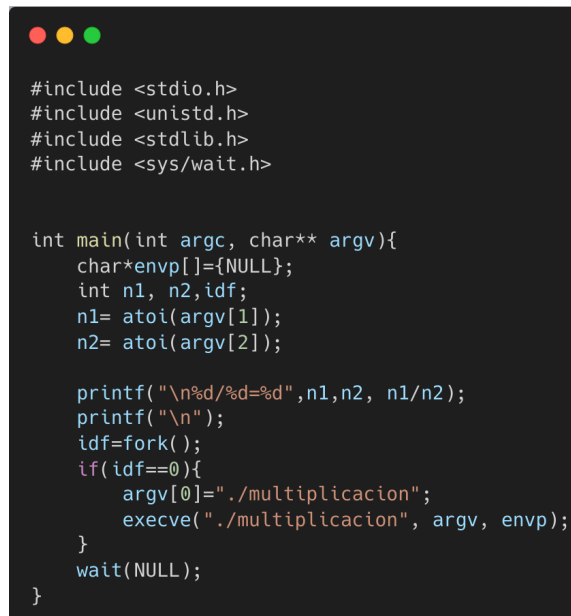
    idf2=fork();
    if(idf2==0){
        argv[0]="./resta";
        execve("./resta", argv, envp);
    }

    idf3=fork();
    if(idf3==0){
        argv[0]="./combinaciones";
        execve("./combinaciones", argv, envp);
    }
    wait(NULL);
    //argv[0]="./calculadora";
    //execve("./calculadora", argv, envp);
    sleep(2);
    return 0;
}

```

Imagen 6: Código calculadora.c

Como vemos en la *imagen 7* mediante la llamada a sistema **fork()** creamos un proceso hijo en el proceso *division* y a ese hijo le asignamos el código de *multiplicacion*, por último hacemos lo mismo que antes, utilizar la instrucción **wait()** para que este proceso termine después que el proceso hijo.



```

#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]="./multiplicacion";
        execve("./multiplicacion", argv, envp);
    }
    wait(NULL);
}

```

Imagen 7: Código division.c

Lo mismo sucederá en el caso del proceso *resta* que tendrá como hijo al proceso *permutaciones*, lograrlo es básicamente hacer lo mismo, crear un hijo y asignarle el código del proceso correspondiente. Y que no se nos olvidé colocar la instrucción que espera a que el proceso hijo termine.

Por otro lado, para el resto de procesos que son solamente hijos, agregamos una instrucción **sleep(2)** al final, esto con el fin de visualizar la estructura y que los procesos que no tienen hijos no terminen tan pronto. Realmente es el único cambio que se realiza en estos procesos, ya que fuera de ello no necesita más tratamiento.

Como veremos en la *imagen 8* al ejecutar la *calculadora* verificamos que la estructura generada es igual a la deseada, donde los hijos de medio tendrán a sus dos respectivos hijos de hecho al notar como es que entrega resultados el programa, primero se dan los resultados del primer nivel del árbol y luego se dan los resultados del segundo nivel.

De esta manera podemos llegar a las estructuras deseadas, desde mi punto de vista es más sencillo realizarlo así ya que los hijos pueden tener un código propio, esto facilita ya que no necesitamos crear toda la estructura desde *calculadora*, al contrario, podemos dividir la creación de procesos para que cada proceso cree el hijo o hijos que requiera, incluso creo que es una buena práctica realizarlo así, es parecido a la POO.

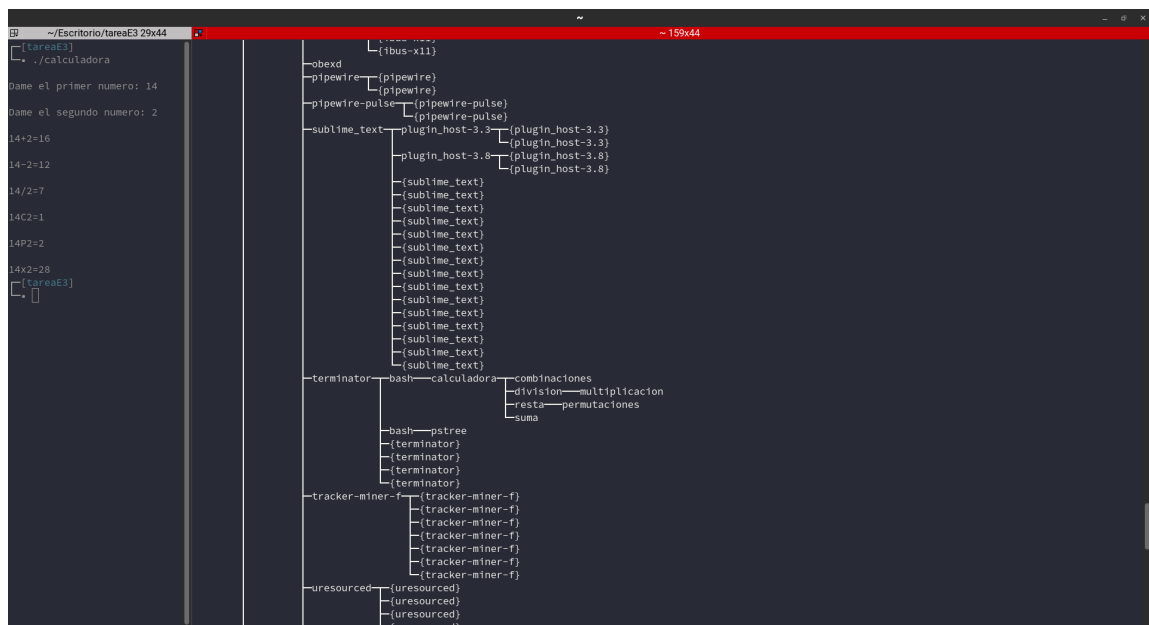


Imagen 8: Estructura generada