Tarea: Calculadora con procesos

Diego Ruiz Mora — 2202000335

9-Septiembre-2023

1 Creación de procesos para cada operación

Para iniciar, será necesario crear los procesos que nos ayudarán a realizar las operaciones correspondientes de suma, resta, multiplicación y división. En los cuales el código es bastante sencillo, como podemos apreciar en los siguientes imágenes:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

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

    printf("\n%d+%d=%d",n1,n2, n1+n2);
    printf("\n");
    sleep(5);
}
```

Figure 1: Código de suma.c

Podemos notar de manera clara que en el código recibimos los parámetros, que en este caso tenemos el primero de ellos se refiere a la longitud del vector, que es el segundo parámetro que pasamos al 'main'. Este vector es de cadenas que termina con un '**NULL**, y al inicio tiene el comando con él que se invoca el programa, que en este caso sería **./suma**, en el medio tenemos algunos otros argumentos que se utilizarán en la ejecución de las instrucciones.

Esto se ve posteriormente en el código cuando de la tercera y cuarta posición del vector de argumentos obtenemos los números que vamos a sumar. Seguido, al efectuar la operación directamente lo imprimimos. Lo mismo sucederá en los siguientes casos, ya que las 4 primeras operaciones son sencillas de implementar.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

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

    printf("\n%d-%d=%d",n1,n2, n1-n2);
    printf("\n");
    sleep(5);
}
```

Figure 2: Código de resta.c

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

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

    printf("\n%d/%d=%d",n1,n2, n1/n2);
    printf("\n");
    sleep(5);
}
```

Figure 3: Código de división.c

Como se puede apreciar cada uno de los códigos tienen una estructura parecida, solo difieren en la operación que realizan y en que como imprimen los datos, ya que tienen cada uno un operador distinto. Tal no es la situación al tener un procedimiento como los siguientes que son la permutación o la combinación, en tales caso tendremos que definir distintas funciones que nos permitan el calculo.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

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

    printf("\n%dx%d=%d",n1,n2, (n1*n2));
    printf("\n");
    sleep(5);
}
```

Figure 4: Código de multiplicación.c

Primero tendremos que recordar que la fórmula para calcular el número de permutaciones y combinaciones:

$$nPr = \frac{n!}{(n-r)!} \tag{1}$$

$$nCr = \frac{n!}{(n-r)!\ r!} \tag{2}$$

Con el conocimiento de estas ecuaciones debemos de asumir que necesitamos una función que nos calcule el factorial y a su vez otra que nos calcule el numero de permutaciones para dos números dados, entendamos en ambas ecuaciones (1) y (2), a n como el número de elementos tomados en rangos de r. Por tal razón, tendremos que mandar a llamar a la función factorial desde el cálculo de las permutaciones.

Por otro lado, en la función *main* encontramos la función **atoi()** para hacer el *casteo* de las cadenas encontradas dentro del vector de argumentos. Seguido tendremos que aplicar algunas condicionales para prevenir los posibles errores, como que se den elementos que sean igual a cero, o que tengamos que escoger grupos r mayores que el numero de elementos n (tal situación no existe en estos cálculos).

En cualquiera de los dos casos tendremos que advertir de lo que esta ocurriendo, esto se pudo haber solucionado desde la petición de los datos, pero se decidió hacer de esta forma debido a que las otras operaciones no tienen estas restricciones. Al final pondremos una pequeña detención del programa para que al momento de correrlo seamos capaces de ver que es lo que esta ocurriendo a nivel de procesos.

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int factorial(int n){
    int res=1;
    while(n>0){
        res=res*n;
        n--;
    }
    return res;
}

int permutaciones(int n, int r){
    return factorial(n)/ factorial(n-r);
}

int main(int argc, char** argv){
    int n1, n2, res;
    n1= atoi(argv[1]);
    n2= atoi(argv[2]);

    if(n1>=0||n2>=0){
        if(n1>n2){
            printf("\n%dC%d=%d",n1,n2, permutaciones(n1,n2));
        }
        else
            printf("\nEl primer numero debe de ser mayor que el segundo");
    }
    else
        printf("\nLos numeros deben de ser positivos");

    printf("\n");
    sleep(5);
}
```

Figure 5: Código de permutaciones.c

Será fácil deducir que en el caso de las combinaciones ocurre prácticamente lo mismo, con la ligera diferencia de la operación que realizaremos, en la *figura 6* tendrá el nombre de "combinaciones" asi como su repectiva ecuación 2, pero fuera de eso se le da el mismo tratamiento a los datos y cuidamos exactamente los mismos aspectos relacionados con la naturaleza de los números (que no sean cero o menores, o que el segundo de ellos no sea mayor que el primero).

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int factorial(int n){
    int res=1;
    while(n>0){
        res=res*n;
        n--;
    }
    return res;
}

int combinaciones(int n, int r){
    return factorial(n)/ (factorial(n-r)*factorial(r));
}

int main(int argc, char** argv){
    int n1, n2, res;
    n1= atoi(argv[1]);
    n2= atoi(argv[2]);

    if(n1>=0||n2>=0){
        if(n1>n2){
            printf("\n%dC%d=%d",n1,n2, combinaciones(n1,n2));
        }
        else
            printf("\nEl primer numero debe de ser mayor que el segundo");
    }
    else
        printf("\nLos numeros deben de ser positivos");
    printf("\n");
    sleep(5);
}
```

Figure 6: Código de combinaciones.c

Quizá lo más interesante y el propósito principal de esta tarea recaiga en la parte relacionada con la función *main*, donde encontraremos el uso de procesos más presente. Para ello quizá debamos de ver el código, primero que nada.

Este código lo encontramos en la *figura 7*, al inicio se hace presente la petición de los datos, que en este caso y para ahorrarnos pasos decidimos hacerlo con *cadenas*, ya que de una u otra forma los pasaremos con este tipo al vector de argumentos, lo cual se ve expresado en la asignación de los números a espacios particulares de la sección ($argv[1] = num1$). En seguida haremos la petición de la operación que queremos realizar, para ello utilizamos una estructura *if-elseif-else*, pero sin antes realizar el clonado del proceso con la instrucción **fork()** que como vimos anteriormente nos creará un proceso hijo.

A este proceso hijo le podremos pasar un cantidad de parámetros de tal manera que opere un código distinto, esto lo logramos con la instrucción **execve()**, la cual nos permite asignar el código distinto al hijo que acabamos de crear. De hecho, buscaremos asignar el código del proceso que es respectivo de la opción que se da un paso antes (suma, resta, multiplicación, división, permutaciones, combinaciones), dentro de cada una de las opciones asignaremos el identificador de los procesos, por ejemplo *argv[0]="./suma"* para el caso de la suma.

```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];
    char opc[20];

    printf("\nDame el primer numero: ");
    scanf("%s",&num1);
    argv[1]=num1;
    printf("\nDame el segundo numero: ");
    scanf("%s",&num2);
    argv[2]=num2;
    printf("\nQue quieres hacer(suma, resta, multiplicacion, divison, permutaciones, combinaciones): ");
    scanf("%s", &opc);

    idf=fork();
    if(idf==0){
        if(strcmp(opc, "suma")==0){
            argv[0]="./suma";
            execve("./suma", argv, envp);
        }
        else if (strcmp(opc, "resta")==0){
            argv[0]="./resta";
            execve("./resta", argv, envp);
        }
        else if (strcmp(opc, "multiplicacion")==0){
            argv[0]="./multiplicacion";
            execve("./multiplicacion", argv, envp);
        }
        else if (strcmp(opc, "division")==0){
            argv[0]="./division";
            execve("./division", argv, envp);
        }
        else if (strcmp(opc, "permutaciones")==0){
            argv[0]="./permutaciones";
            execve("./permutaciones", argv, envp);
        }
        else if (strcmp(opc, "combinaciones")==0){
            argv[0]="./combinaciones";
            execve("./combinaciones", argv, envp);
        }
        else{
            printf("\nNo es una opcion valida");
            exit(0);
        }
    }
    else{
        wait(NULL);
    }
    argv[0]="./calculadora";
    execve("./calculadora", argv, envp);
    return 0;
}
```

Figure 7: Código de calculadora.c

Esto tiene suma importancia ya que la instrucción **execve()** tendremos como parámetros primero el nombre del proceso que mandaremos a llamar, luego el vector de argumentos (**argv**) y por un vector de apuntadores para las variables de estado, que denominamos **envp**, dentro del vector de argumentos encontramos lo siguiente:

1. El procedimiento al que queremos mandar a llamar para copiar el código al hijo.
2. El primer número n_1 a operar.
3. El segundo número n_2 a operar.
4. NULL. Que funciona como un tipo de indicador para decir que el vector de argumentos ya termino.

Es importante notar que todas estas opciones se encuentran dentro de un **if** que nos hace modificar exclusivamente el código del proceso hijo (**idf==0**), en caso de no ocurrir esto simplemente hará un espera a que proceso padre se termine de ejecutar después de que el hijo termine su ejecución, incluso podríamos omitir este **else** ya que lo realmente importante es discriminar la ejecución de estas instrucciones solamente al proceso hijo.
Así mismo en caso de querer ingresar una opción invalida para la operación a realizar se generará una advertencia de que tal opción no es valida. Además de que ejecutará la instrucción **exit(0)** que terminará el proceso actual, que en este caso será el proceso hijo. Y no se vayan acumulando todos los procesos al llegar al final.

Por último, pero yo creo que bastante importante, es la llamada al mismo proceso calculadora, para ello se modifica el vector de argumentos en su primera posición e incorporamos la instrucción **execve** mandando a llamar al mismo proceso, para así generar un menú que se repita para hacer muchas operaciones, en este caso no nos interesa cuales sean los otros argumentos, ya que verdaderamente no tenemos como tal el recibimiento de los argumentos para la función *main* de este proceso.

En las siguientes imágenes veremos como es que opera, en este caso con el uso de la herramienta *htop* para notar de manera clara cada uno de los procesos y los argumentos que recibe.
En la *figura 8* podemos notar como en la petición de los datos aun seguimos dentro del proceso padre, después de que demos la operación y demos 'enter' debería de aparecer un nuevo proceso que tenga el nombre de la operación que se esta realizando, como en la *figura 9*, donde se ve como el proceso se llama **"./suma"** y tiene como parámetros *12* y *13* que son los valores que se pasaron.

Luego de ello, al terminar la operación tenemos que este proceso *suma* creado deja de aparecer y solamente aparece el procedimiento padre que es el producto de la ultima instrucción que manda a llamar al mismo proceso. Hay que denotar que el uso de esta instrucción sale de las instrucciones que ejecutan los hijos, por lo que solamente lo hará el padre, es como sobrescribir el programa. Cabe decir, que esta instrucción la pudimos incorporar en el **else** de la comparación **idf==0**.

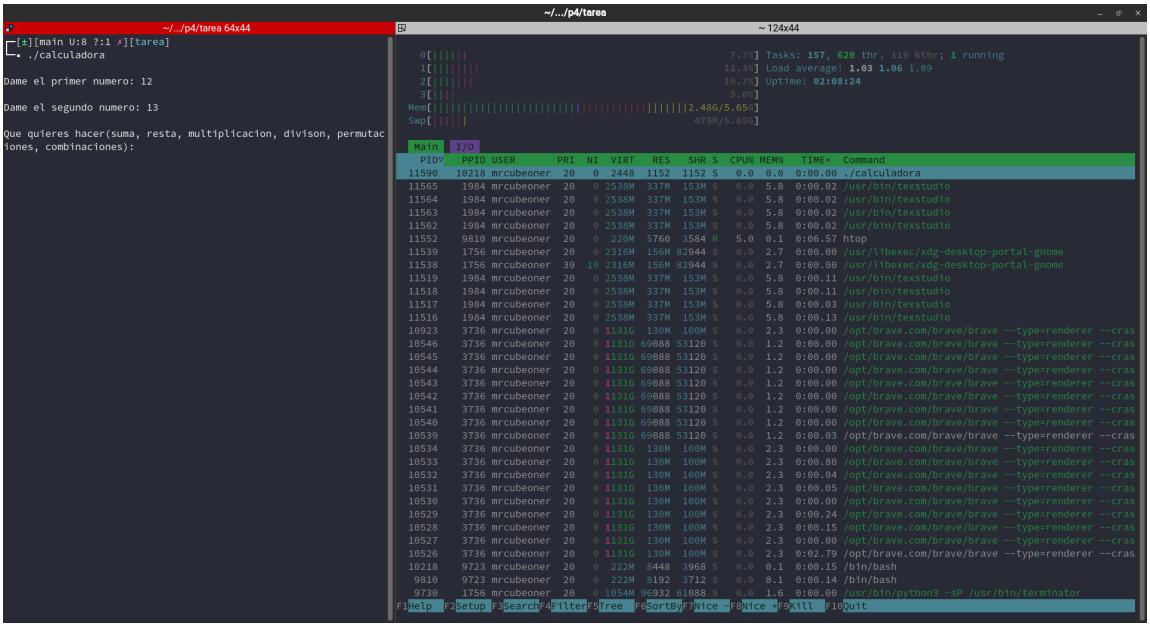

Figure 8: Prueba 1.1

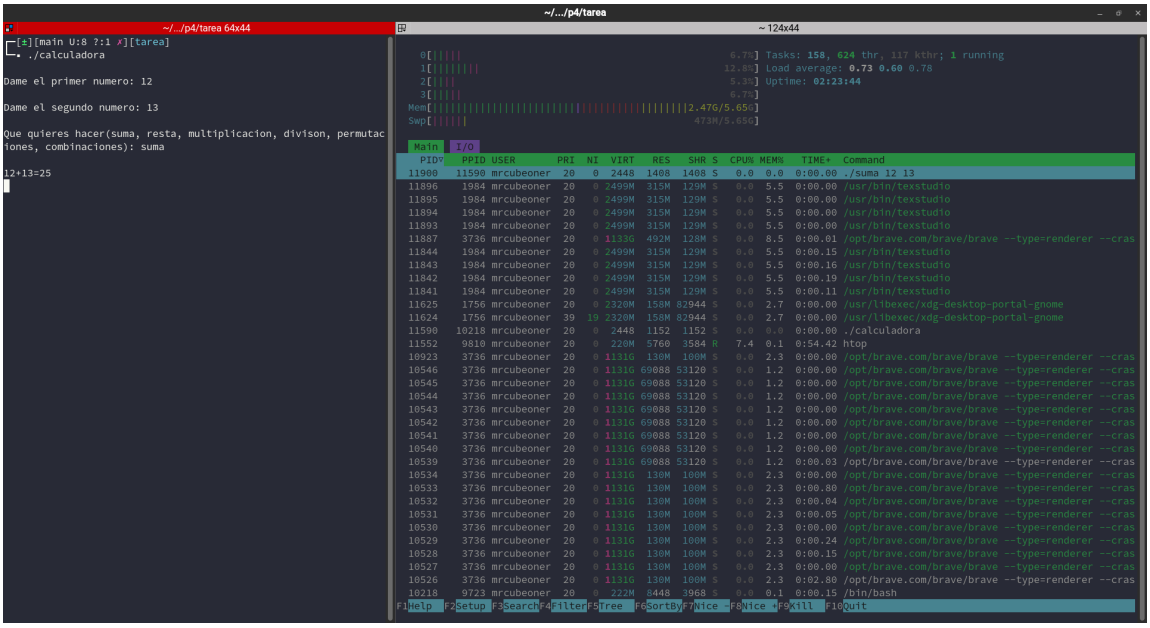

Figure 9: Prueba 1.2

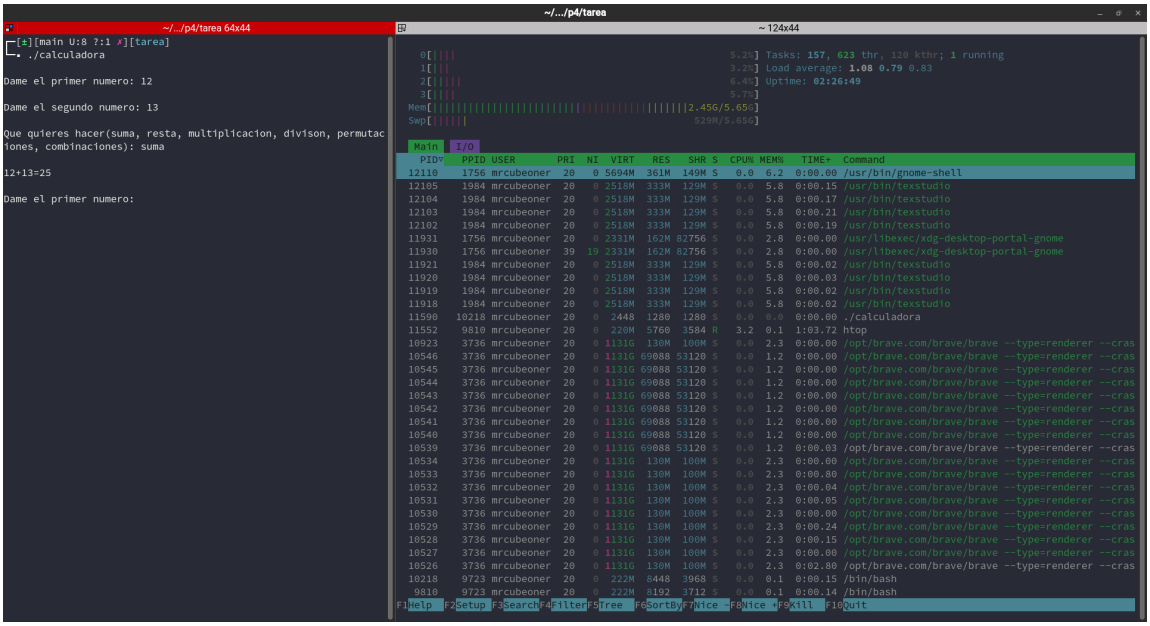

Figure 10: Prueba 1.3