INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

SISTEMAS OPERATIVOS

CORTÉS GALICIA JORGE

MENDOZA PARRA SERGIO

OLVERA AGUILA LEONARDO DANIEL

PAZ SÁNCHEZ BRANDON

2CM7

PRÁCTICA 5. ADMINISTRADOR DE PROCESOS EN LINUX Y WINDOWS (2)

LUNES, 13 DE NOVIEMBRE DE 2017

Contenido

Competencias

El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux y Windows a través de la creación de nuevos procesos por copia exacta de código y/o por sustitución de código para el desarrollo de aplicaciones concurrentes sencillas.

Desarrollo

Seccion Linux

1. Introduzca los siguientes comandos a traves de la consola del sistema operativos Linux:

Ps

El comando ps permite comprobar el estado de los procesos activos en un sistema y mostrar información técnica sobre los procesos. Estos datos son útiles para tareas administrativas, como la determinación de la manera de definir las prioridades del proceso. Según las opciones utilizadas, el comando ps proporciona la siguiente información:

- Estado actual del proceso
- ID de proceso
- ID de proceso principal
- Identificador del usuario
- Clase de programación
- Prioridad
- Dirección del proceso
- Memoria utilizada
- Tiempo de CPU utilizado

Ps-fea

Muestra un proceso en específico.
ps [modificadores] [condición]
ps –fea condición
El (–fea):

- -f Mostrar columnas de usuario, PID, PPID, CPU, STIME, TTY, TIME, y COMMAND
- -e Seleccionar todos los procesos del usuario.
- -a Lista los procesos de todos los usuarios.

2. A través de la ayuda en línea que proporciona Linux, investigue para que se utiliza el comando **ps** y mencione las opciones que se pueden utilizar con dicho comando. Además, investigue el uso de las llamadas al sistema **fork (), execv (), getpid (), getppid ()** y **wait ()** en la ayuda en línea, mencione que otras funciones similares a execv () existen, reporte sus observaciones.

ps [modificadores] [condición]
Donde ***modificadores*** es opcional, y puede tomar los siguientes valores:
Los siguientes modificadores no toman el parámetro condición:

- **-A**: Muestra todos los procesos (de todos los usuarios en el sistema).
- **-a**: Muestra todos los procesos de una [tty] determinada.
- **-d**: Muestra todo excepto los líderes de la sesión.
- **-e**: Muestra todos los procesos (equivalente a -A).
- **T**: Muestra todos los procesos de la terminal actual.
- **a**: Muestra todos los procesos de la terminal actual incluyendo los de otros usuarios.
- **g**: Muestra todos los procesos incluyendo grupos líderes (obsoleta excepto en sunOs).
- **r**: Muestra solamente los procesos corriendo.
- **x**: Muestra los procesos en un estilo BSD (sin controlar la [TTY]).

Los siguientes modificadores toman el parámetro condición:

- **-N**: Muestra todos los procesos excepto los que encajan con la condición (equivalente a --deselect).
- **-C**: Muestra los procesos que tienen como nombre la condición.
- **-G**: Muestra los procesos que tienen como grupo (nombre de grupo o id) la condición.
- **-P**: Muestra los procesos que tienen como [Identificador de proceso] la condición.
- **-S**: Muestra los procesos que tienen como sesión la condición.
- **-U**: Muestra los procesos que tienen como usuario (nombre de grupo o id) la condición.

Existen distintos modificadores admitidos según la versión del comando ps que se esté usando en el sistema (BSD, POSIX, GNU, etc.)

fork ()
Los procesos se crean a través de la llamada al sistema fork. Cuando se realiza dicha llamada, el kernel duplica el entorno de ejecución del proceso que llama, dando como resultado dos procesos. El proceso original que hace la llamada se conoce como proceso padre, mientras que el nuevo proceso resultado de la llamada se conoce como proceso hijo. La diferencia en los segmentos de datos de ambos procesos es que la llamada fork retorna un 0 al proceso hijo, y un entero que representa el PID del hijo al proceso padre. Si la llamada fracasa, no se crea el proceso hijo y se devuelve -1 al proceso padre. Una vez ejecutada con éxito la llamada fork y devueltos los valores de retorno ambos procesos continúan su ejecución a partir de la siguiente instrucción al fork.

wait ()

Wait es una llamada al sistema del sistema operativo UNIX, estandarizada en POSIX (y otros).

- Permite a un proceso padre esperar hasta que termine un proceso hijo.
- El entero apuntado por el argumento *status* será actualizado con un código que indica el estado de terminación del proceso hijo.
- Devuelve el identificador del proceso hijo ó -1 en caso de error.

getpid ()

Getpid devuelve el identificador de proceso del proceso invocador. (Esto se utiliza a menudo por las rutinas que generan nombres de archivos temporales únicas.)

getppid ()

Getppid devuelve el identificador de proceso del padre del proceso invocador.
Estas llamadas al sistema devuelven el identificador de proceso ya sea del padre o del hijo.

execv ()

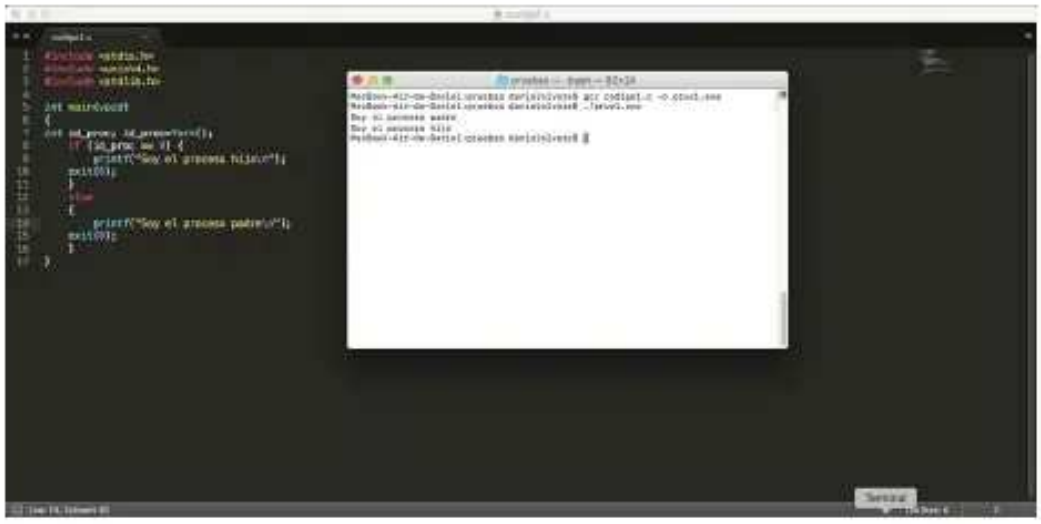
Reemplaza la imagen en memoria de un proceso por el de un archivo del filesystem.

3. Capture, compile y ejecute los dos programas de creación de un nuevo proceso por copia exacta de código que a continuación se muestra. Observe su funcionamiento y experimente con el código.

Primer código

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

int main(void)
{
int id_proc; id_proc=fork();
   if (id_proc == 0) {
         printf("Soy el proceso hijo");
   exit(0);
   }
   else
   {
```

```
        printf("Soy el proceso padre");
    exit(0);
    }
}
```



Pantalla 1. Compilaciσn en la terminal del programa codigo1.c

Segundo Código

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(void)
{
int id_proc; id_proc=fork();
    if (id_proc == 0) {
        printf("Soy el proceso hijo"); }
    else {
        printf("Soy el proceso padre"); }
        printf("Mensaje en ambos");
exit(0); }
```

Pantalla 2. Compilacion en la terminal del programa codigo2.c

4. Programe una aplicación que cree seis procesos (por copia exacta de código). El primer proceso se encargará de realizar la suma de dos matrices de 10x10 elementos tipo entero, el segundo proceso realizará la resta sobre esas mismas matrices, el tercer proceso realizará la multiplicación de las matrices, el cuarto proceso obtendrá las transpuestas de cada matriz y el quinto proceso obtendrá las matrices inversas. Cada uno de estos procesos escribirá un archivo con los resultados de la operación que realizó. El sexto proceso leerán los archivos de resultados y los mostrará en pantalla cada uno de ellos. Programe la misma aplicación sin la creación de procesos, es decir de forma secuencial. Obtenga los tiempos de ejecución de las aplicaciones, compare estos tiempos y dé sus observaciones.

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

#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <string.h>

void suma(int a[10][10],int b[10][10]);
void resta(int a[10][10],int b[10][10]);
void multiplicacion(int a[10][10],int b[10][10]);
void transpuesta(int a[10][10],int b[10][10]);
void inversa(int a[10][10],int b[10][10]);

void main()
{
    FILE *f;
    clock_t start,end;
    float total;
    srand(time(NULL));
    int status=0,a[10][10],b[10][10],c1,c2,i,j,k,y;
```

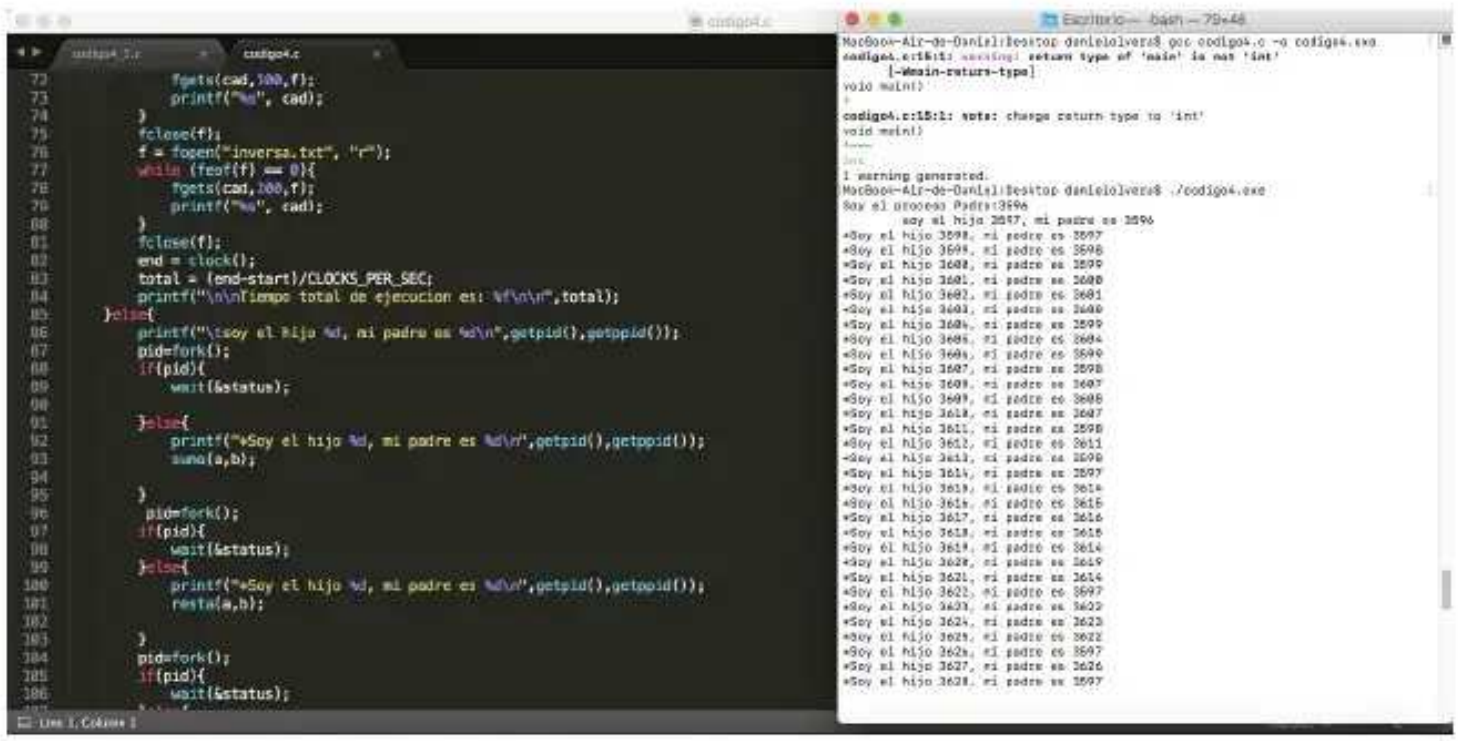
```
char cad[1000];
pid_t pid;
start = clock();
for(c1=0;c1<10;c1++){
    for(c2=0;c2<10;c2++){
        a[c1][c2]=rand() % (10);
        b[c1][c2]=rand() % (10);
    }
}
printf("Soy el proceso Padre:%d\n",getpid());
pid=fork();
if(pid){
    wait(&status);
    printf("\n\nMatriz A:\n\n");
    for(c1 = 0; c1 < 10; c1++){
        printf("|");
        for(c2 = 0; c2 < 10; c2++){
            printf("%d\t",a[c1][c2]);
        }
        printf("|\n");
    }
    printf("\n\nMatriz B:\n\n");
    for(c1 = 0; c1 < 10; c1++){
            printf("|");

        for(c2 = 0; c2 < 10; c2++){
            printf("%d\t",b[c1][c2]);
        }
        printf("|\n");
    }

f = fopen("suma.txt", "r");
while (feof(f) == 0){
    fgets(cad,100,f);
    printf("%s", cad);
}
fclose(f);
f = fopen("resta.txt", "r");
while (feof(f) == 0){
    fgets(cad,100,f);
    printf("%s", cad);
}
fclose(f);
f = fopen("multiplica.txt", "r");
while (feof(f) == 0){
    fgets(cad,100,f);
    printf("%s", cad);
}
fclose(f);
f = fopen("transpuesta.txt", "r");
while (feof(f) == 0){
    fgets(cad,100,f);
    printf("%s", cad);
}
fclose(f);
f = fopen("inversa.txt", "r");

while (feof(f) == 0){
    fgets(cad,100,f);
```

```
            printf("%s", cad);
        }
        fclose(f);
        end = clock();
        total = (end-start)/CLOCKS_PER_SEC;
        printf("\n\nTiempo total de ejecucion es: %f\n\n",total);
    }else{
        printf("\tsoy el hijo %d, mi padre es %d\n",getpid(),getppid());
        pid=fork();
        if(pid){
            wait(&status);

        }else{
            printf("*Soy el hijo %d, mi padre es
%d\n",getpid(),getppid());
            suma(a,b);

        }
         pid=fork();
        if(pid){
            wait(&status);
        }else{
            printf("*Soy el hijo %d, mi padre es
%d\n",getpid(),getppid());
            resta(a,b);

        }
        pid=fork();
        if(pid){
            wait(&status);
        }else{
            printf("*Soy el hijo %d, mi padre es
%d\n",getpid(),getppid());
            multiplicacion(a,b);

        }
        pid=fork();
        if(pid){
            wait(&status);
        }else{
            printf("*Soy el hijo %d, mi padre es
%d\n",getpid(),getppid());
            transpuesta(a,b);

        }
        pid=fork();
        if(pid){
            wait(&status);
        }else{
            inversa(a,b);
            printf("*Soy el hijo %d, mi padre es
%d\n",getpid(),getppid());

        }
    }
}
```

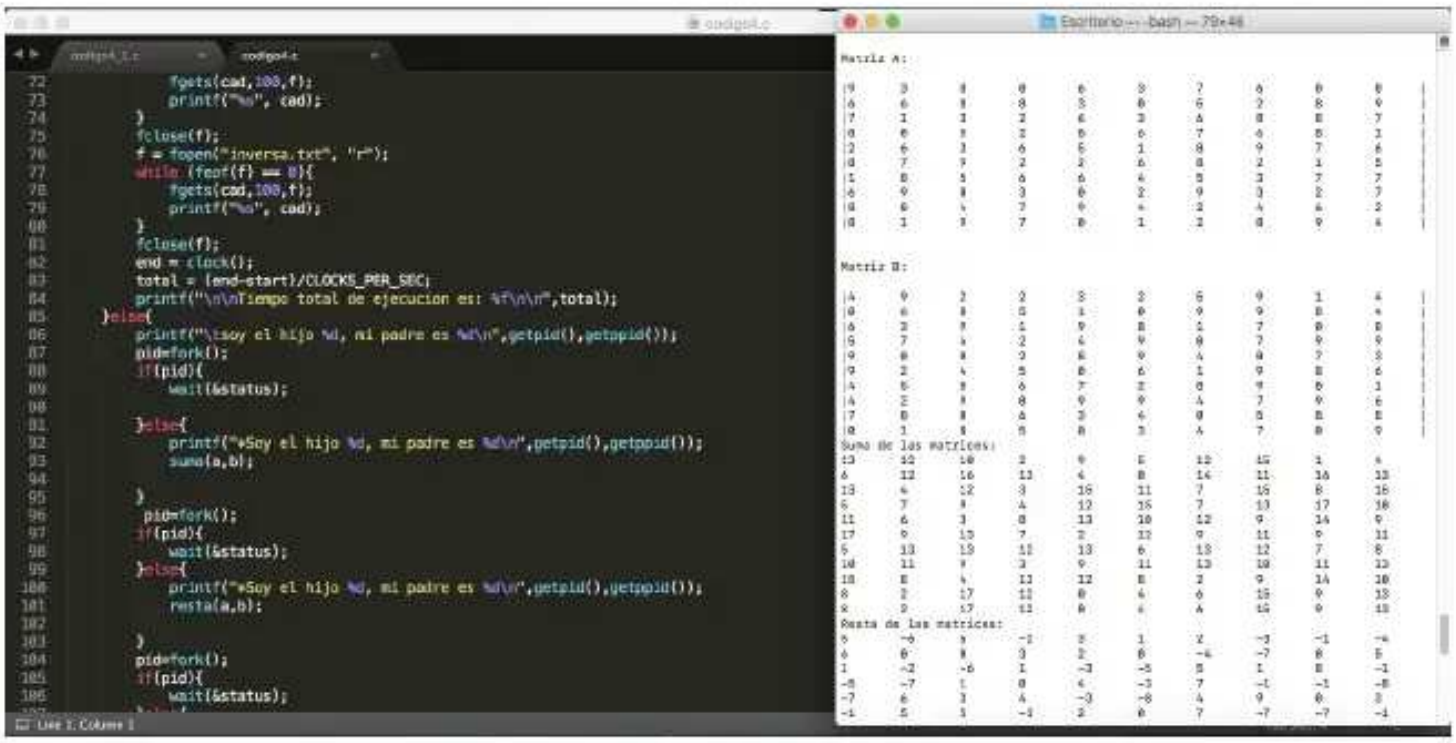
```c
void suma(int a[10][10],int b[10][10]){
    FILE *f;
    int  c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = (a[c1][c2] + b[c1][c2]);
        }
    }
    f = fopen("suma.txt","w+");
    fprintf(f, "%s\n", "Suma de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void resta(int a[10][10],int b[10][10]){
    FILE *f;
    int c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = a[c1][c2] - b[c1][c2];
        }
    }

    f = fopen("resta.txt","w+");
    fprintf(f, "%s\n", "Resta de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void multiplicacion(int a[10][10],int b[10][10]){
    FILE *f;
    int c1,c2,c[10][10],i,j,k;
    for (i=0;i<10;i++){
        for (j=0;j<10;j++){
            c[i][j]=0;
            for (k=0;k<10;k++){
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
    f = fopen("multiplica.txt","w+");
    fprintf(f, "%s\n", "Multiplicacion de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
```

```
    }
    fclose(f);
}

void transpuesta(int a[10][10],int b[10][10]){
    FILE *f;
    int c[10][10],d[10][10],c1,c2;
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = a[c2][c1];
            d[c1][c2] = b[c2][c1];
        }
    }
    f = fopen("transpuesta.txt","w+");
    fprintf(f, "%s\n", "Transpuesta de la matriz A: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fprintf(f, "%s\n", "Transpuesta de la matriz B: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", d[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void inversa(int a[10][10],int b[10][10]){
    FILE *f;
    int factor,factor2,j,i,k,c1,c2;
    float c[10][10],d[10][10];
    for (j = 0; j < 10; j++){
        for (k = 0; k < 10; k++) {
            for (i = k+1; i < 10;  i++) {
                factor = a[i][k]/a[k][k];
                factor2 = b[i][k]/b[k][k];
                for (j = k+1; j < 10; j++) {
                    c[i][j] = a[i][j] - factor * a[k][j];
                    d[i][j] = b[i][j] - factor2 * b[k][j];
                }
            }
        }
    }
    f = fopen("inversa.txt","w+");
    fprintf(f, "%s\n", "Inversa de la matriz A: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%.2f\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }

    fprintf(f, "%s\n", "Inversaa de la matriz B: ");
    for(c1 = 0; c1 < 10; c1++){
```

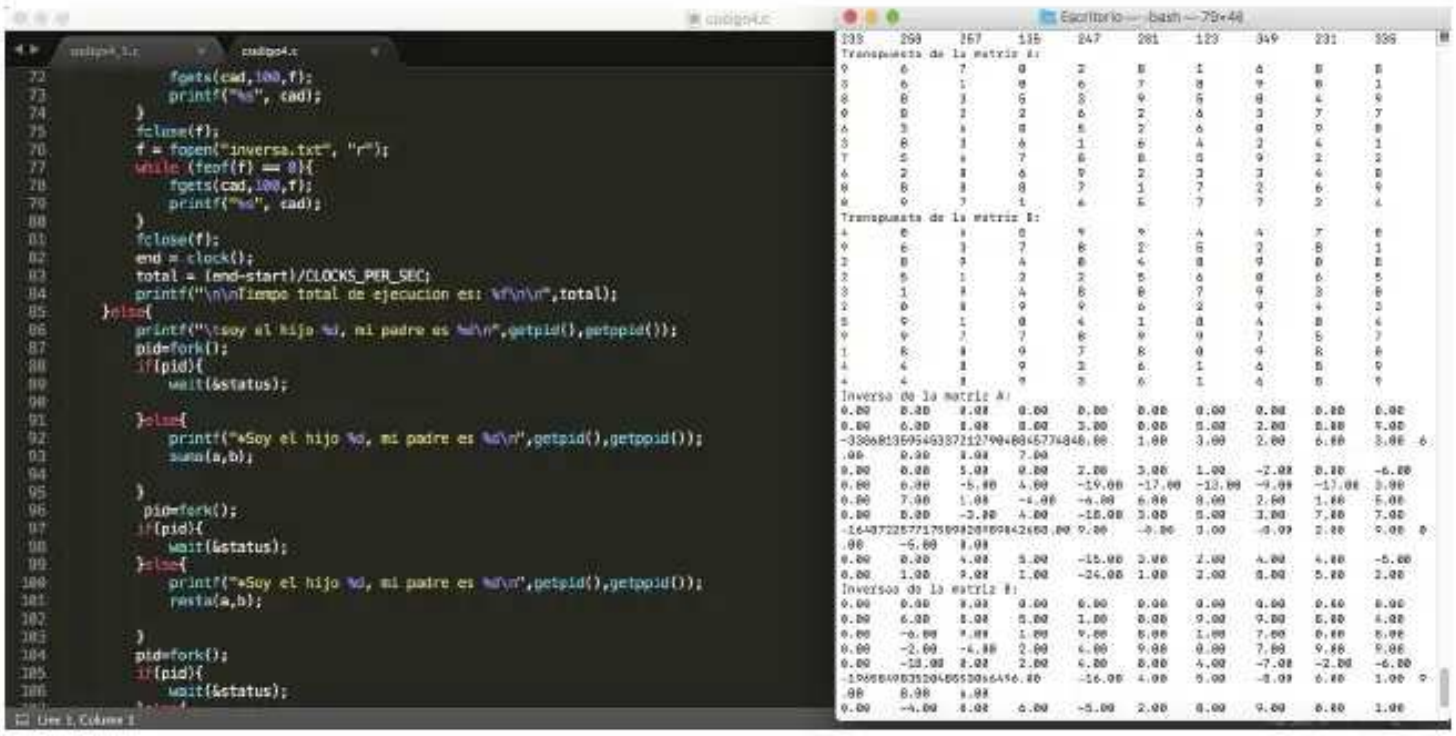
```
            for(c2 = 0; c2 < 10; c2++){
                fprintf(f, "%.2f\t", d[c1][c2]);
            }
            fprintf(f, "%s\n","");
        }
        fclose(f);
    }
```



Pantalla 3. Compilacion en la terminal del programa codigo4.c



Pantalla 4. Compilaciσn en la terminal del programa codigo4.c



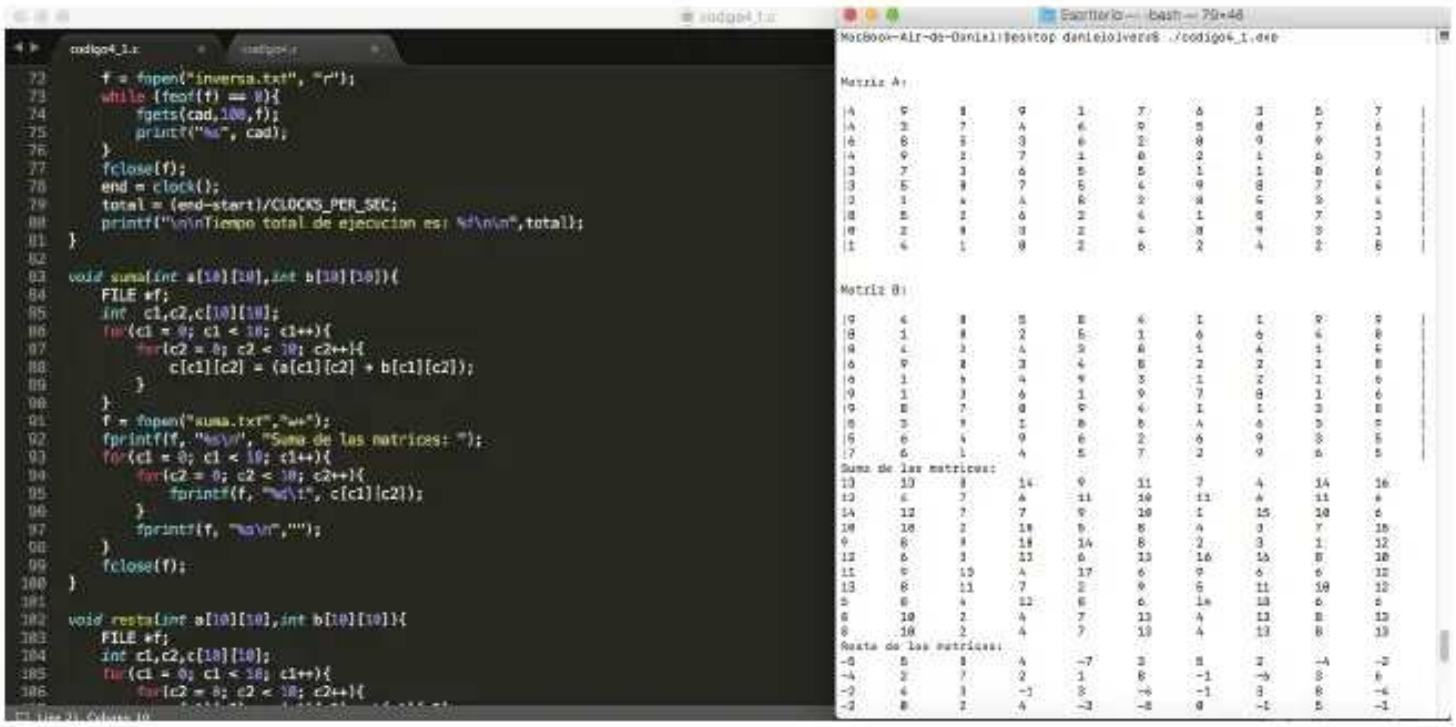
Pantalla 5. Compilaciσn en la terminal del programa codigo4.c

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

void suma(int a[10][10],int b[10][10]);
void resta(int a[10][10],int b[10][10]);
void multiplicacion(int a[10][10],int b[10][10]);
void transpuesta(int a[10][10],int b[10][10]);
void inversa(int a[10][10],int b[10][10]);

void main()
{
    FILE *f;
    clock_t start,end;
    float total;
    srand(time(NULL));
    int status=0,a[10][10],b[10][10],c1,c2,i,j,k,y;
    char cad[1000];
    //Creacion de la matriz
    start = clock();
    for(c1=0;c1<10;c1++){
        for(c2=0;c2<10;c2++){
            a[c1][c2]=rand() % (10);
            b[c1][c2]=rand() % (10);

        }
    }
    suma(a,b);
    resta(a,b);
    multiplicacion(a,b);
    transpuesta(a,b);
    inversa(a,b);
    printf("\n\nMatriz A:\n\n");
    for(c1 = 0; c1 < 10; c1++){
        printf("|");
        for(c2 = 0; c2 < 10; c2++){
            printf("%d\t",a[c1][c2]);
        }
        printf("|\n");
    }
    printf("\n\nMatriz B:\n\n");
    for(c1 = 0; c1 < 10; c1++){
        printf("|");
        for(c2 = 0; c2 < 10; c2++){
            printf("%d\t",b[c1][c2]);
        }
        printf("|\n");
    }
    f = fopen("suma.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
    }
    fclose(f);
    f = fopen("resta.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
```

```
    }
    fclose(f);
    f = fopen("multiplica.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
    }
    fclose(f);
    f = fopen("transpuesta.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
    }
    fclose(f);
    f = fopen("inversa.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
    }
    fclose(f);
    end = clock();
    total = (end-start)/CLOCKS_PER_SEC;
    printf("\n\nTiempo total de ejecucion es: %f\n\n",total);
}

void suma(int a[10][10],int b[10][10]){
    FILE *f;
    int  c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = (a[c1][c2] + b[c1][c2]);
        }
    }
    f = fopen("suma.txt","w+");
    fprintf(f, "%s\n", "Suma de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void resta(int a[10][10],int b[10][10]){
    FILE *f;
    int c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = a[c1][c2] - b[c1][c2];
        }
    }

    f = fopen("resta.txt","w+");
    fprintf(f, "%s\n", "Resta de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
```

```
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void multiplicacion(int a[10][10],int b[10][10]){
    FILE *f;
    int c1,c2,c[10][10],i,j,k;
    for (i=0;i<10;i++){
        for (j=0;j<10;j++){
            c[i][j]=0;
            for (k=0;k<10;k++){
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
    f = fopen("multiplica.txt","w+");
    fprintf(f, "%s\n", "Multiplicacion de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void transpuesta(int a[10][10],int b[10][10]){
    FILE *f;
    int c[10][10],d[10][10],c1,c2;
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = a[c2][c1];
            d[c1][c2] = b[c2][c1];
        }
    }
    f = fopen("transpuesta.txt","w+");
    fprintf(f, "%s\n", "Transpuesta de la matriz A: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fprintf(f, "%s\n", "Transpuesta de la matriz B: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", d[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void inversa(int a[10][10],int b[10][10]){
```

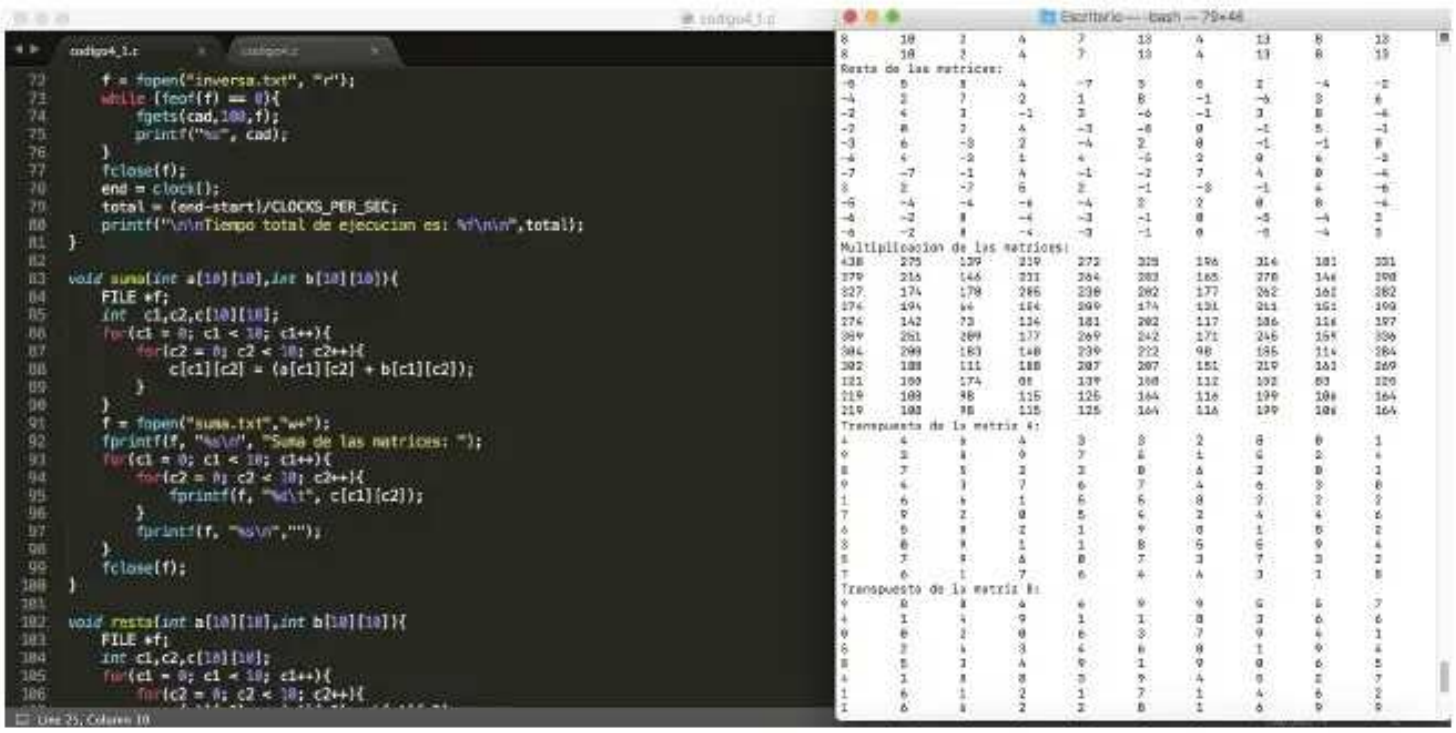
```
FILE *f;
int factor,factor2,j,i,k,c1,c2;
float c[10][10],d[10][10];
for (j = 0; j < 10; j++){
    for (k = 0; k < 10; k++) {
        for (i = k+1; i < 10;  i++) {
            factor = a[i][k]/a[k][k];
            factor2 = b[i][k]/b[k][k];
            for (j = k+1; j < 10; j++) {
                c[i][j] = a[i][j] - factor * a[k][j];
                d[i][j] = b[i][j] - factor2 * b[k][j];
            }
        }
    }
}
f = fopen("inversa.txt","w+");
fprintf(f, "%s\n", "Inversa de la matriz A: ");
for(c1 = 0; c1 < 10; c1++){
    for(c2 = 0; c2 < 10; c2++){
        fprintf(f, "%.2f\t", c[c1][c2]);
    }
    fprintf(f, "%s\n","");
}
fprintf(f, "%s\n", "Inversaa de la matriz B: ");
for(c1 = 0; c1 < 10; c1++){
    for(c2 = 0; c2 < 10; c2++){
        fprintf(f, "%.2f\t", d[c1][c2]);
    }
    fprintf(f, "%s\n","");
}
    fclose(f);
}
```



```
    f = fopen("inversa.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
    }
    fclose(f);
    end = clock();
    total = (end-start)/CLOCKS_PER_SEC;
    printf("\n\nTiempo total de ejecucion es: %f\n\n",total);
}

void suma(int a[10][10],int b[10][10]){
    FILE *f;
    int  c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = (a[c1][c2] + b[c1][c2]);
        }
    }
    f = fopen("suma.txt","w+");
    fprintf(f, "%s\n", "Suma de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void resta(int a[10][10],int b[10][10]){
    FILE *f;
    int c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
```

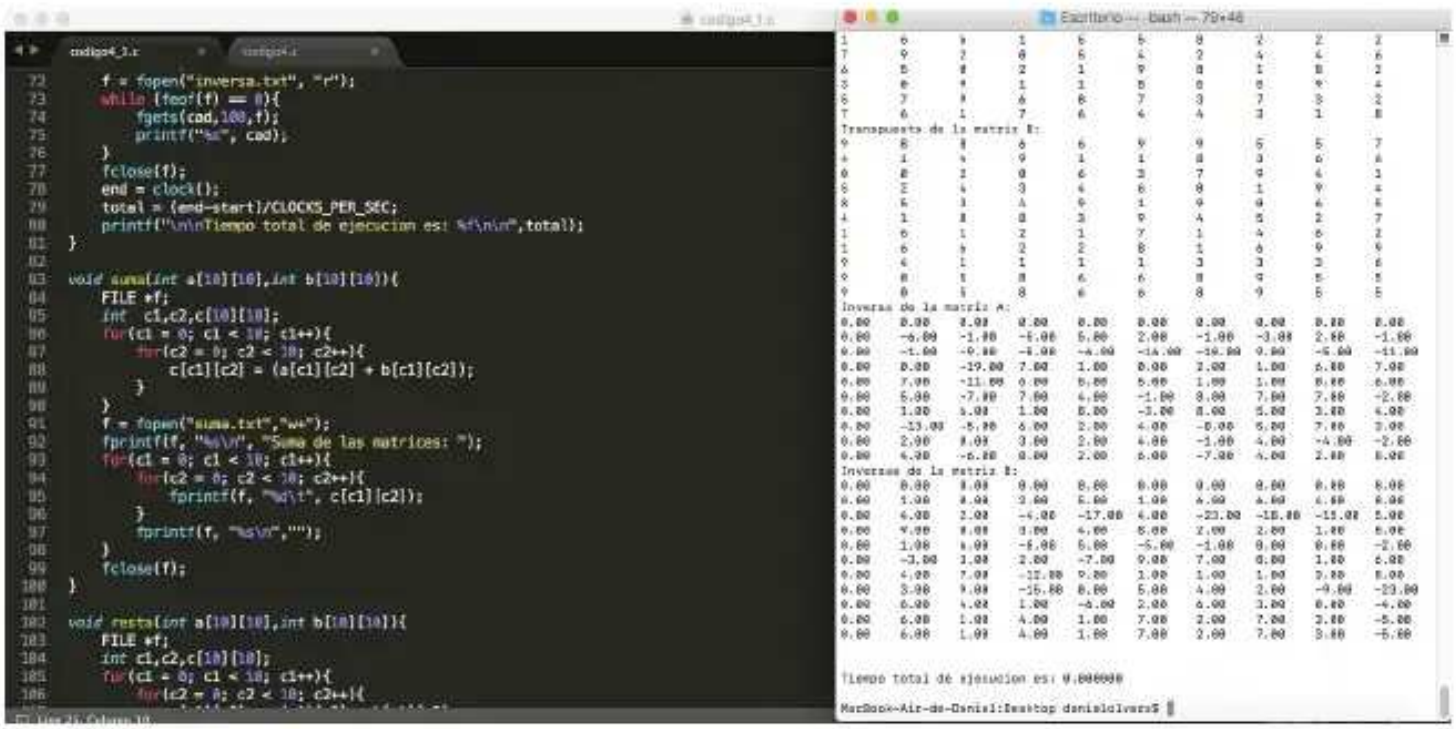
```
MacBook-Air-de-Daniel:Desktop danielolvera$ ./codigo4_1.exe

Matriz A:

|4     9     8     9     1     7     6     3     5     7  |
|4     3     7     4     6     9     5     8     7     6  |
|6     8     5     3     6     2     8     9     9     1  |
|4     9     2     7     1     8     2     1     8     7  |
|3     7     3     6     5     5     1     1     8     6  |
|3     5     8     7     5     6     9     8     7     3  |
|3     1     4     4     8     2     8     6     3     4  |
|8     5     2     6     2     6     1     8     7     3  |
|8     2     8     3     2     4     8     9     3     1  |
|1     6     1     8     2     6     2     4     2     8  |

Matriz B:

|9     6     8     5     8     6     1     1     9     9  |
|8     1     8     2     5     1     6     6     6     8  |
|9     4     3     4     3     8     1     4     1     4  |
|6     9     8     3     4     8     2     2     1     8  |
|6     1     5     4     9     5     1     2     1     6  |
|9     1     3     6     1     9     7     8     1     6  |
|9     8     7     8     9     6     1     1     3     8  |
|8     3     9     1     8     5     4     6     3     5  |
|6     6     4     9     6     2     6     9     3     5  |
|7     6     1     4     5     7     2     9     6     5  |
Suma de las matrices:
17      13      8       14      9       11      7       4       14      16
12      6       7       4       11      10      11      4       11      6
14      12      7       7       9       10      1       15      10      6
10      10      2       10      9       5       4       3       7       15
9       8       9       10      14      8       2       3       1       12
12      6       3       11      6       13      16      15      8       10
11      9       13      4       17      6       9       6       6       12
13      8       11      7       2       9       6       11      10      12
5       8       4       12      6       6       14      13      6       6
8       10      2       4       7       13      4       13      8       13
8       10      2       4       7       13      4       13      8       13
Resta de las matrices:
-6      5       8       4       -7      3       6       2       -4      -2
-4      3       7       2       1       8       -1      -6      3       6
-2      4       3       -1      3       -6      -1      3       8       -4
-2      8       2       4       -3      -8      0       -1      5       -1
-3      6       -8      2       -4      2       0       -1      -1      8
-4      4       -2      1       4       -6      2       0       4       -8
-7      -7      -1      4       -1      -2      7       4       0       -6
8       2       -7      6       2       -1      -8      -1      4       -6
-5      -4      -4      -8      -4      2       2       0       0       -4
-4      -2      8       -4      -3      -1      0       -5      -4      2
-8      -2      8       -4      -3      -1      0       -8      -4      8
Multiplicacion de las matrices:
438     275     139     219     272     328     196     314     181     331
379     216     146     231     264     283     165     278     144     298
327     174     178     285     230     282     177     262     161     282
276     194     66      184     209     174     131     213     161     199
276     142     73      134     181     202     117     186     114     197
389     251     209     177     269     242     171     246     159     336
304     200     183     148     239     222     98      185     114     184
302     188     111     188     207     207     151     219     161     269
121     169     174     68      139     168     112     192     83      120
219     103     98      115     125     164     116     199     108     164
219     103     98      115     125     164     116     199     108     164
Transpuesto de la matriz A:
4       4       8       4       3       8       2       6       6       1
9       3       8       9       7       4       1       6       2       4
8       7       5       3       3       8       6       3       8       1
9       4       3       7       6       7       4       6       3       8
1       6       6       1       6       6       8       2       2       2
7       9       2       8       5       6       2       4       4       6
6       6       8       2       1       9       8       1       6       2
8       8       8       1       1       8       5       6       9       4
8       7       8       6       8       7       3       7       3       3
7       6       1       7       6       4       6       3       1       8
Transpuesto de la matriz B:
9       8       8       6       8       9       9       6       5       7
6       1       4       9       1       1       8       3       6       6
8       8       2       8       6       3       7       9       4       1
6       7       4       3       4       6       8       1       9       4
8       5       3       6       9       1       9       8       6       5
6       1       8       8       3       9       4       8       2       7
1       6       1       2       1       7       1       4       6       2
1       6       8       2       2       8       1       6       9       9
```

Pantalla 6. Compilaciσn en la terminal del programa codigo4_1.c



```
    f = fopen("inversa.txt", "r");
    while (feof(f) == 0){
        fgets(cad,100,f);
        printf("%s", cad);
    }
    fclose(f);
    end = clock();
    total = (end-start)/CLOCKS_PER_SEC;
    printf("\n\nTiempo total de ejecucion es: %f\n\n",total);
}

void suma(int a[10][10],int b[10][10]){
    FILE *f;
    int  c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            c[c1][c2] = (a[c1][c2] + b[c1][c2]);
        }
    }
    f = fopen("suma.txt","w+");
    fprintf(f, "%s\n", "Suma de las matrices: ");
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
            fprintf(f, "%d\t", c[c1][c2]);
        }
        fprintf(f, "%s\n","");
    }
    fclose(f);
}

void resta(int a[10][10],int b[10][10]){
    FILE *f;
    int c1,c2,c[10][10];
    for(c1 = 0; c1 < 10; c1++){
        for(c2 = 0; c2 < 10; c2++){
```

Pantalla 7. Compilaciσn en la terminal del programa codigo4_1.c

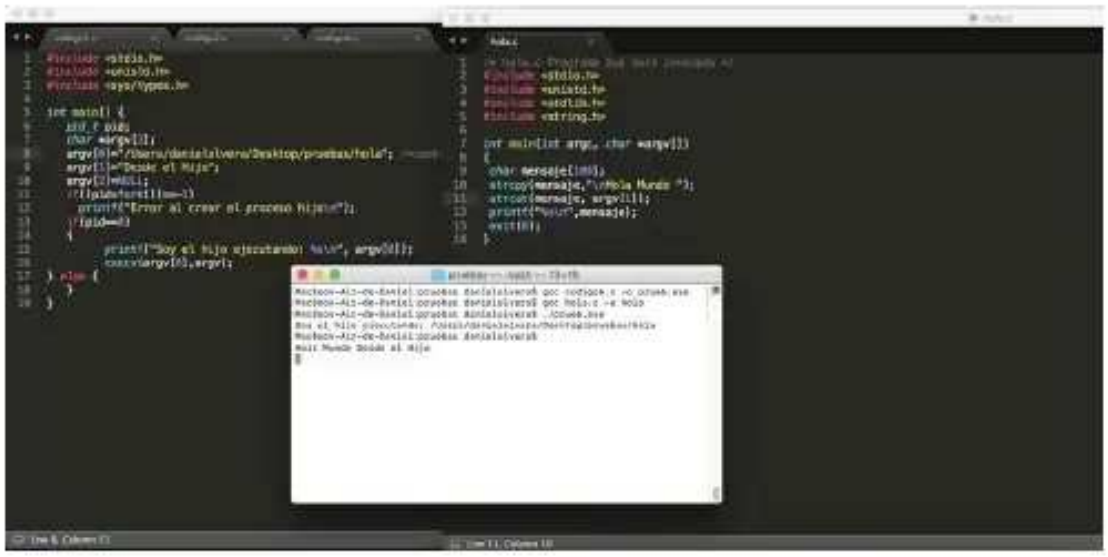


Pantalla 8. Compilaciσn en la terminal del programa codigo4_1.c

5. Capture, compile y ejecute el siguiente programa de creación de un nuevo proceso con sustitución de un nuevo código, así como el programa que será el nuevo código a ejecutar. Observe su funcionamiento y experimente con el código.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
  pid_t pid;
  char *argv[3];
  argv[0]="/Users/danielolvera/Desktop/pruebas/hola";
  argv[1]="Desde el Hijo";
  argv[2]=NULL;
  if((pid=fork())==-1)
    printf("Error al crear el proceso hijo/");
  if(pid==0)
  {
       printf("Soy el hijo ejecutando: %s/n", argv[0]);
       execv(argv[0],argv);
} else {
  }
}

/* hola.c Programa que será   invocado */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
```

```
{
 char mensaje[100];
 strcpy(mensaje,"¥nHola Mundo ");
 strcat(mensaje, argv[1]);
 printf("%s¥n",mensaje);
 exit(0);
}
```



Pantalla 9. Compilacion en la terminal del programa codigo6.c y hola.c

Sección Windows

1. Iniciar Sesión en Windows.
2. Para esta práctica se utilizará el ambiente de programación Dev C/C++.
3. Capture y compile el programa de creación de un nuevo proceso que a continuación se muestra (ver imagen 1.0).

```
#include <windows.h>
#include <stdio.h>

int main(int argc,char *argv[]){
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    int i;
    ZeroMemory(&si,sizeof(si));

    si.cb=sizeof(si);
    ZeroMemory(&pi,sizeof(pi));

    if(argc!=2){
        printf("Usar: %s Nombre_programa_hijo\n",argv[0]);
        return 0;
    }

    //Creacion de el proceso hijo
    if(!CreateProcess(NULL,argv[1],NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
        printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
        return 0;
    }

    //Proceso Padre
    printf("Soy el padre\n");
    WaitForSingleObject(pi.hProcess,INFINITE);
```

```
    //Terminacion controlada del proceso e hilo asociado de ejecucion
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

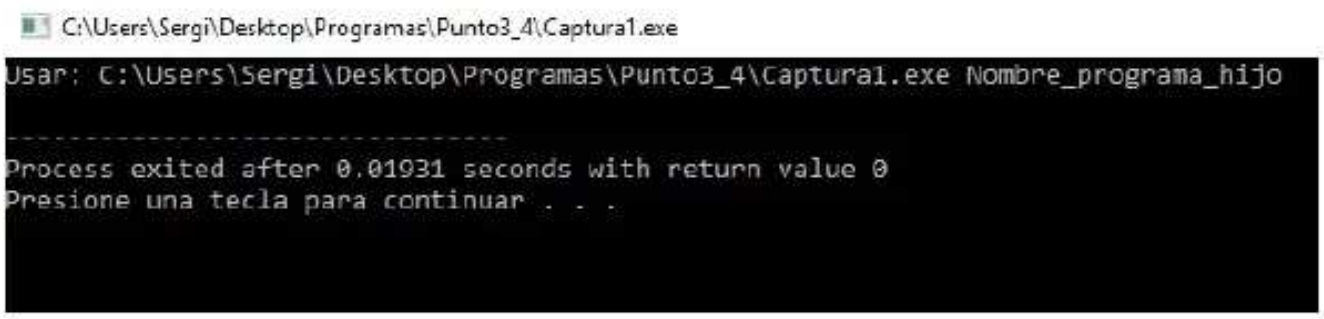


Imagen 1.0: Creacion de Proceso

4. Capture y compile el programa que contendrá al proceso hijo que a continuación se muestra (ver imagen 1.1).

```
#include <windows.h>
#include <stdio.h>

int main(void) {
    printf("Soy el proceso hijo");
    exit(0);
}
```

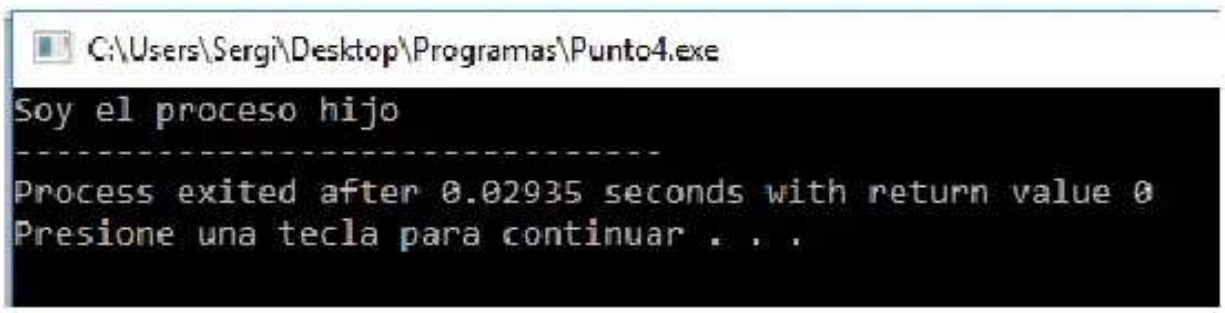


Imagen 1.1: Proceso Hijo

5. Ejecute el primer código pasando como argumento el nombre del archivo ejecutable del segundo código capturado. Observe el funcionamiento del programa, reporte sus observaciones y experimente con el código (ver imagen 1.2).

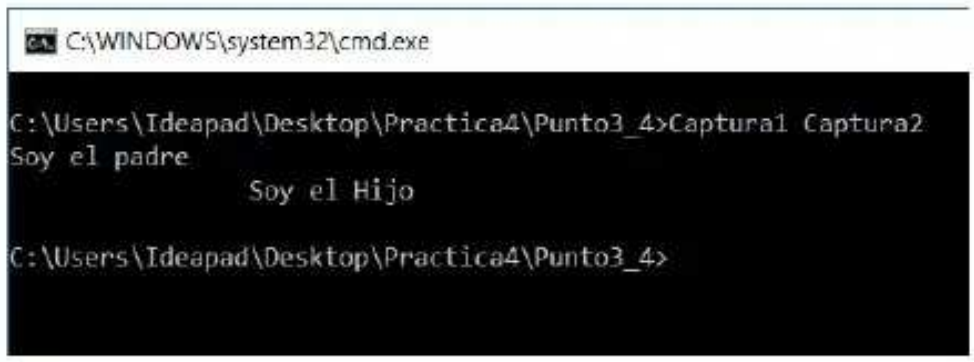


Figura 1.2: Proceso padre e hijo

6. Compare y reporte tanto las diferencias como las similitudes que encuentra con respecto a la creación de procesos por sustitución de código en Linux.

Linux	Windows
Diferencias y similitudes	
Linux combina tanto la creación de procesos por copia exacta de código, como la sustitución de código, esto para permitir que tanto el proceso creador como el nuevo proceso estén en ejecución concurrentemente.	**Windows** asocia un hilo a cada proceso que ejecuta, y es este hilo donde se ejecuta el nuevo proceso por sustitución de código, conservándose al proceso creador.
Llamadas al sistema para la creación de procesos por sustitución de código	
execv (): Función que proporciona una matriz de punteros a cadenas terminadas en nulo, que representan la lista de argumentos disponible para el nuevo programa. El primer argumento, por convención, debe apuntar al nombre de archivo asociado con el archivo que se está ejecutando. La matriz de punteros debe terminar con un puntero NULL.	**CreateProcess ():** Crea un nuevo proceso y su hilo principal. El nuevo proceso se ejecuta en el contexto de seguridad del proceso de llamada.

7. Programe una aplicación que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará 5 procesos hijos más. A su vez cada uno de los cinco procesos creará 3 procesos más. Cada uno de los procesos creados imprimirá en pantalla su identificador. Consejo investigue **GetCurrentProcessId () DEL API WIN32.**

Proceso Padre:

```
#include <windows.h>
#include <stdio.h>

void main(){
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    ZeroMemory(&si,sizeof(si));
    si.cb=sizeof(si);
    ZeroMemory(&pi,sizeof(pi));

    //Creacion de el proceso hijo
    if(!CreateProcess(NULL,"ProcesoHijo",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
        printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
        return;
    }

    //Proceso Padre
    printf("Soy el padre\n");

    WaitForSingleObject(pi.hProcess,INFINITE);

    //Terminacion controlada del proceso e hilo asociado de ejecucion
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Proceso Hijo:

```
#include <windows.h>
#include <stdio.h>

void main(){

    STARTUPINFO no;
    PROCESS_INFORMATION po;
    int j;
    ZeroMemory(&no,sizeof(no));
    no.cb=sizeof(no);
    ZeroMemory(&po,sizeof(po));

    //Creacion de los procesos hijos
    if(!CreateProcess(NULL,"ProcesoNietos",NULL,NULL,FALSE,0,NULL,NULL,&no,&po)){
        printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
        return;
    }

    printf("\tSoy el hijo\n");
    WaitForSingleObject(po.hProcess,INFINITE);

    //Terminacion controlada del proceso e hilo asociado de ejecucion
    CloseHandle(po.hProcess);
    CloseHandle(po.hThread);
}
```

Procesos Nietos:

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>

#define MAX_THREADS 5
#define BUF_SIZE 255
#define MAX_S_THREADS 3


DWORD WINAPI MyThreadFunction( LPVOID lpParam );
DWORD WINAPI SMyThreadFunction( LPVOID lpParam );
void ErrorHandler(LPTSTR lpszFunction);
void Pruebas();

// Sample custom data structure for threads to use.
// This is passed by void pointer so it can be any data type
// that can be passed using a single void pointer (LPVOID).

typedef struct MyData {
    int val1;
    int val2;
} MYDATA, *PMYDATA;


int _tmain(){
    PMYDATA pDataArray[MAX_THREADS];

    DWORD   dwThreadIdArray[MAX_THREADS];
    HANDLE  hThreadArray[MAX_THREADS];

    // Create MAX_THREADS worker threads.
    int i;
    for( i=0; i<MAX_THREADS; i++ ){
// Allocate memory for thread data.

pDataArray[i] = (PMYDATA) HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
                    sizeof(MYDATA));

        if( pDataArray[i] == NULL )
        {
           // If the array allocation fails, the system is out of memory
           // so there is no point in trying to print an error message.
           // Just terminate execution.
            ExitProcess(2);
        }

        // Generate unique data for each thread to work with.

        pDataArray[i]->val1 = i;
        pDataArray[i]->val2 = i+100;

        // Create the thread to begin execution on its own.

        hThreadArray[i] = CreateThread(
            NULL,                   // default security attributes
            0,                      // use default stack size
            MyThreadFunction,       // thread function name
            pDataArray[i],          // argument to thread function
            0,                      // use default creation flags
            &dwThreadIdArray[i]);   // returns the thread identifier


        // Check the return value for success.
        // If CreateThread fails, terminate execution.
```

```
        // This will automatically clean up threads and memory.

        if (hThreadArray[i] == NULL)
        {
           ErrorHandler(TEXT("CreateThread"));
           ExitProcess(3);
        }

        Pruebas();

    } // End of main thread creation loop.
    // Wait until all threads have terminated.

    WaitForMultipleObjects(MAX_THREADS, hThreadArray, TRUE, INFINITE);

    // Close all thread handles and free memory allocations.

    for(i=0; i<MAX_THREADS; i++)
    {
        CloseHandle(hThreadArray[i]);
        if(pDataArray[i] != NULL)
        {
            HeapFree(GetProcessHeap(), 0, pDataArray[i]);
            pDataArray[i] = NULL;    // Ensure address is not reused.
        }
    }

    return 0;
}


DWORD WINAPI MyThreadFunction( LPVOID lpParam )
{
    HANDLE hStdout;
    PMYDATA pDataArray;

    TCHAR msgBuf[BUF_SIZE];
    size_t cchStringSize;
    DWORD dwChars;

    // Make sure there is a console to receive output results.

    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
    if( hStdout == INVALID_HANDLE_VALUE )
        return 1;

    // Cast the parameter to the correct data type.
    // The pointer is known to be valid because
    // it was checked for NULL before the thread was created.

    pDataArray = (PMYDATA)lpParam;

    // Print the parameter values using thread-safe functions.

    StringCchPrintf(msgBuf, BUF_SIZE, TEXT("\t\tSoy el nieto = %d, %d\n"),
        pDataArray->val1, pDataArray->val2);
    StringCchLength(msgBuf, BUF_SIZE, &cchStringSize);
    WriteConsole(hStdout, msgBuf, (DWORD)cchStringSize, &dwChars, NULL);

    return 0;
}

DWORD WINAPI SMyThreadFunction( LPVOID lpParam )
```

```
{
    HANDLE hStdout;
    PMYDATA pDataArray;

    TCHAR msgBuf[BUF_SIZE];
    size_t cchStringSize;
    DWORD dwChars;

    // Make sure there is a console to receive output results.

    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
    if( hStdout == INVALID_HANDLE_VALUE )
        return 1;

    // Cast the parameter to the correct data type.
    // The pointer is known to be valid because
    // it was checked for NULL before the thread was created.

    pDataArray = (PMYDATA)lpParam;

    // Print the parameter values using thread-safe functions.

    StringCchPrintf(msgBuf, BUF_SIZE, TEXT("\t\t\tSoy el bisnieto = %d, %d\n"),
        pDataArray->val1, pDataArray->val2);
    StringCchLength(msgBuf, BUF_SIZE, &cchStringSize);
    WriteConsole(hStdout, msgBuf, (DWORD)cchStringSize, &dwChars, NULL);

    return 0;
}

void ErrorHandler(LPTSTR lpszFunction)
{
    // Retrieve the system error message for the last-error code.

    LPVOID lpMsgBuf;
    LPVOID lpDisplayBuf;
    DWORD dw = GetLastError();

    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dw,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR) &lpMsgBuf,
        0, NULL );

    // Display the error message.

    lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
        (lstrlen((LPCTSTR) lpMsgBuf) + lstrlen((LPCTSTR) lpszFunction) + 40) *
sizeof(TCHAR));
    StringCchPrintf((LPTSTR)lpDisplayBuf,
        LocalSize(lpDisplayBuf) / sizeof(TCHAR),
        TEXT("%s failed with error %d: %s"),
        lpszFunction, dw, lpMsgBuf);
    MessageBox(NULL, (LPCTSTR) lpDisplayBuf, TEXT("Error"), MB_OK);

    // Free error-handling buffer allocations.

    LocalFree(lpMsgBuf);
    LocalFree(lpDisplayBuf);
```

```
}

void Pruebas(){
    PMYDATA SpDataArray[MAX_S_THREADS];
    DWORD   SdwThreadIdArray[MAX_S_THREADS];
    HANDLE  ShThreadArray[MAX_S_THREADS];

    // Create MAX_THREADS worker threads.
    int j;

    for( j=0; j<MAX_S_THREADS; j++ )
    {
        // Allocate memory for thread data.

        SpDataArray[j] = (PMYDATA) HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
                sizeof(MYDATA));

        if( SpDataArray[j] == NULL )
        {
           // If the array allocation fails, the system is out of memory
           // so there is no point in trying to print an error message.
           // Just terminate execution.
            ExitProcess(2);
        }

        // Generate unique data for each thread to work with.

        SpDataArray[j]->val1 = j;
        SpDataArray[j]->val2 = j+200;

        // Create the thread to begin execution on its own.

        ShThreadArray[j] = CreateThread(
            NULL,                   // default security attributes
            0,                      //  Es el tama□e pila predeterminada.
            SMyThreadFunction,       // Nombre de la funci□e hilo
            SpDataArray[j],          // Argumento para pasar por el hilo.
            0,                      // Se usa banderas para la
creaci□redeterminada.
            &SdwThreadIdArray[j]);   // Devuelve el identificador de hilo


        // Si CreateThread falla se finaliza la ejecuci□
        // Esto automaticamente limpiar^os hilos y la memoria.

        if (ShThreadArray[j] == NULL)
        {
           ErrorHandler(TEXT("CreateThread"));
           ExitProcess(3);
        }
    }

    // Espera hasta que todos los hilos hayan terminado.

    WaitForMultipleObjects(MAX_S_THREADS, ShThreadArray, TRUE, INFINITE);

    // Cierra todos los identificadores de subprocesos y asignaciones de memoria.

    for(j=0; j<MAX_S_THREADS; j++)
    {
        CloseHandle(ShThreadArray[j]);
        if(SpDataArray[j] != NULL)
        {
            HeapFree(GetProcessHeap(), 0, SpDataArray[j]);
```

```
            SpDataArray[j] = NULL;     // Nos aseguramos que la direcci□o se
reutiliza.
        }
    }
}
```

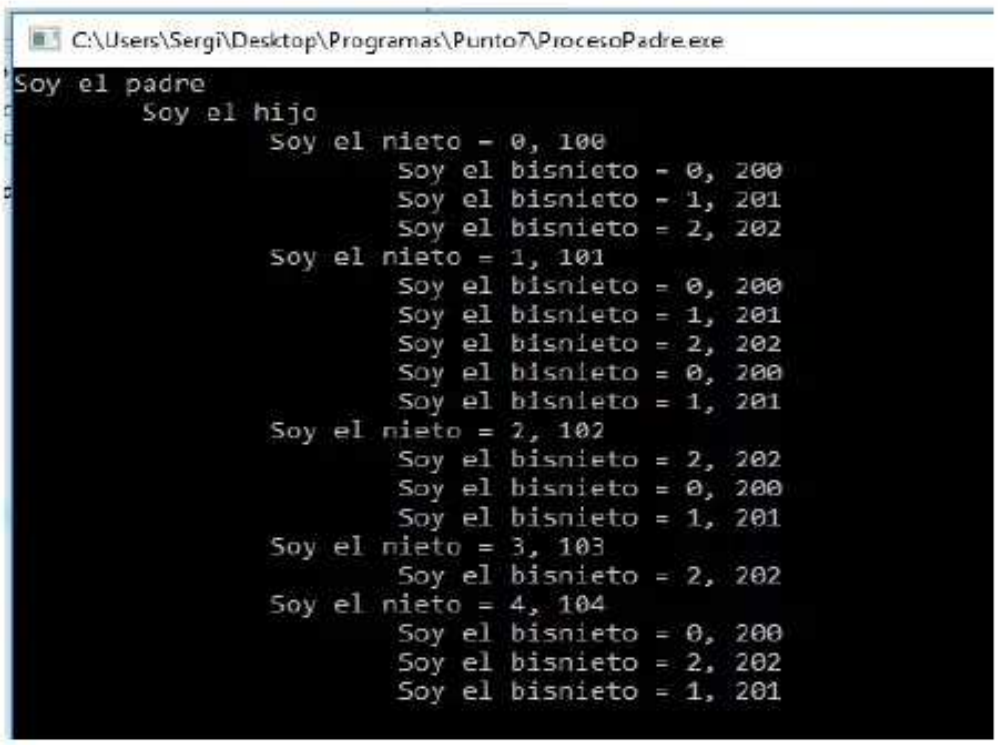
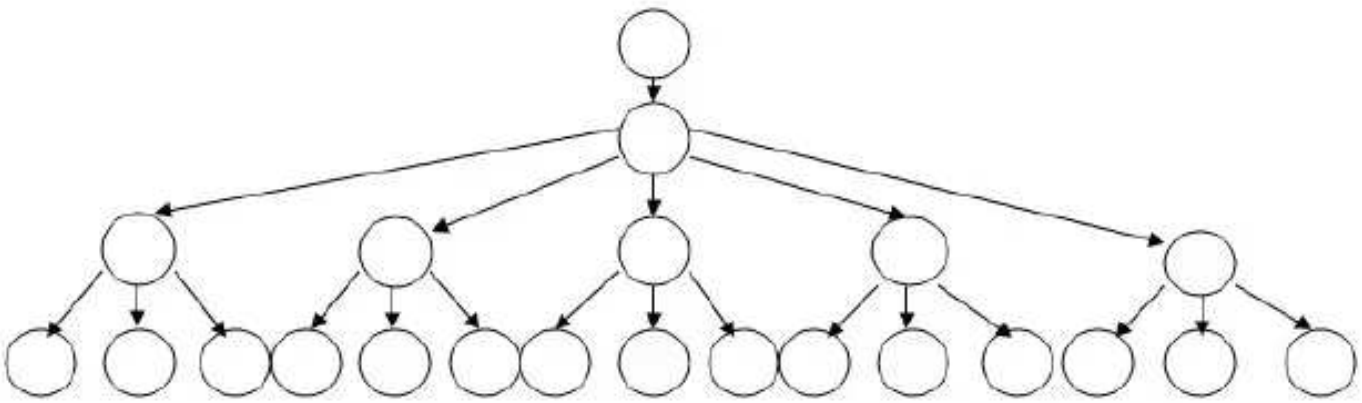


Imagen 1.3: Procesos

Árbol de Procesos



Función GetCurrentProcessId () DEL API WIN32.

Características	Descripción de sus funciones
Función Principal	Recupera el identificador de proceso del proceso de llamada.
Parámetros	Esta función de tiene parámetros.
Valor de Retorno	El valor de retorno es el proceso identificador de la llamada al proceso.
Observaciones	Hasta que el proceso termina, el identificador de proceso identifica de manera única el proceso en todo el sistema.
Sintaxis	DWORD WINAPI GetCurrentProcessId(void);

8. Programe una aplicación que cree seis procesos (por copia exacta de código). El primer proceso se encargará de realizar la suma de dos matrices de 10x10 elementos tipo entero, el segundo proceso realizará la resta sobre las mismas matrices, el tercer proceso realizará la multiplicación de las matrices, el cuarto proceso obtendrá las transpuestas de cada matriz y el quinto proceso obtendrá las matrices inversas. Cada uno de estos procesos escribirá un archivo con los resultados de la operación que realizó. El sexto proceso leerá los archivos de resultados y los mostrará en pantalla cada uno de ellos.

Creación de los 6 procesos

```
#include <windows.h>
#include <stdio.h>

void main(){
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    int i;
    ZeroMemory(&si,sizeof(si));
    si.cb=sizeof(si);
    ZeroMemory(&pi,sizeof(pi));
    do{
    printf("Elija que proceso quiere realizar:\n
            1.SumaMatriz\n
            2.RestaMatriz\n
            3.MultiplicaMatriz\n
```

```
            4.TranspuestaMatriz\n
            5.InversaMatriz\n
            6.VerResultados\n");
    scanf("%d",&i);
    switch(i){
        case 1:
            //Creacion de el proceso hijo


if(!CreateProcess(NULL,"SumaMatriz",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
            printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
            return;
        }
        break;
        case 2:
            //Creacion de el proceso hijo

if(!CreateProcess(NULL,"RestaMatriz",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
            printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
            return;
        }
        break;
        case 3:
                //Creacion de el proceso hijo

if(!CreateProcess(NULL,"MultiplicaMatriz",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
            printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
            return;
         }
         break;
         case 4:
                //Creacion de el proceso hijo

if(!CreateProcess(NULL,"TranspuestaMatriz",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
            printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
            return;
         }
         break;
         case 5:
                //Creacion de el proceso hijo

if(!CreateProcess(NULL,"InversaMatriz",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
            printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
            return;
          }
          break;
          case 6:
                //Creacion de el proceso hijo

if(!CreateProcess(NULL,"Leer",NULL,NULL,FALSE,0,NULL,NULL,&si,&pi)){
            printf("Fallo al invocar CreateProcess(%d)\n",GetLastError());
            return;
          }
          break;
        default:
                break;
    }
    }while(i!=6);

    //Proceso Padre
    printf("Soy el padre\n");
    WaitForSingleObject(pi.hProcess,INFINITE);
```

```
    //Terminacion controlada del proceso e hilo asociado de ejecucion
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Suma de Matriz

```
#include<stdio.h>
int A[10][10]={
            {8,5,6,4,1,8,2,7,3,3},
            {7,5,5,3,3,2,3,1,2,9},
            {3,5,4,2,2,1,4,7,6,7},
            {2,4,5,8,7,1,2,6,5,4},
            {8,8,1,2,7,5,1,5,5,1},
            {5,6,7,5,6,4,2,4,9,5},
            {6,4,9,8,6,2,5,6,5,3},
            {3,5,6,2,8,4,3,4,8,1},
            {1,7,2,4,8,4,9,5,6,3},
            {6,1,4,5,8,8,8,5,7,4},
            };
int B[10][10]={
            {6,4,2,5,6,3,2,6,5,7},
            {2,3,6,1,4,2,1,4,2,5},
            {4,8,7,4,3,7,2,6,3,3},
            {8,6,7,6,4,7,8,4,1,2},
            {1,3,9,4,6,8,8,8,6,9},
            {7,4,8,5,3,8,6,9,4,5},
            {8,7,5,7,4,8,2,2,9,8},
            {1,2,6,8,8,2,4,6,2,8},
            {9,2,6,1,3,7,2,3,4,1},
            {5,3,5,3,4,2,4,8,2,5},
            };

void main()
{
    int i,j;
    FILE *fichero;
    fichero = fopen("ResultadoSuma.txt","w");
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            fprintf(fichero,"%d\n",A[i][j]+B[i][j]);

    }   }
    fclose(fichero);
    printf("El resultado ha sido almacenado en el archivo ResultadoSuma\n");
    return;
}
```

Resta de Matrices

```
#include<stdio.h>
int A[10][10]={
            {8,5,6,4,1,8,2,7,3,3},
            {7,5,5,3,3,2,3,1,2,9},
            {3,5,4,2,2,1,4,7,6,7},
            {2,4,5,8,7,1,2,6,5,4},
            {8,8,1,2,7,5,1,5,5,1},
            {5,6,7,5,6,4,2,4,9,5},
            {6,4,9,8,6,2,5,6,5,3},
            {3,5,6,2,8,4,3,4,8,1},
            {1,7,2,4,8,4,9,5,6,3},
```

```
            {6,1,4,5,8,8,8,5,7,4},
            };
int B[10][10]={
            {6,4,2,5,6,3,2,6,5,7},
            {2,3,6,1,4,2,1,4,2,5},
            {4,8,7,4,3,7,2,6,3,3},
            {8,6,7,6,4,7,8,4,1,2},
            {1,3,9,4,6,8,8,8,6,9},
            {7,4,8,5,3,8,6,9,4,5},
            {8,7,5,7,4,8,2,2,9,8},
            {1,2,6,8,8,2,4,6,2,8},
            {9,2,6,1,3,7,2,3,4,1},
            {5,3,5,3,4,2,4,8,2,5},
            };

void main()
{
    int i,j;
    FILE *fichero;
    fichero = fopen("ResultadoResta.txt","w");
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            fprintf(fichero,"%d\n",A[i][j]-B[i][j]);
        }
    }
    fclose(fichero);
    printf("El resultado ha sido almacenado en el archivo ResultadoResta\n");
     return;
}
```

Multiplicación de matrices

```
#include<stdio.h>
int A[10][10]={
            {8,5,6,4,1,8,2,7,3,3},
            {7,5,5,3,3,2,3,1,2,9},
            {3,5,4,2,2,1,4,7,6,7},
            {2,4,5,8,7,1,2,6,5,4},
            {8,8,1,2,7,5,1,5,5,1},
            {5,6,7,5,6,4,2,4,9,5},
            {6,4,9,8,6,2,5,6,5,3},
            {3,5,6,2,8,4,3,4,8,1},
            {1,7,2,4,8,4,9,5,6,3},
            {6,1,4,5,8,8,8,5,7,4},
            };
int B[10][10]={
            {6,4,2,5,6,3,2,6,5,7},
            {2,3,6,1,4,2,1,4,2,5},
            {4,8,7,4,3,7,2,6,3,3},
            {8,6,7,6,4,7,8,4,1,2},
            {1,3,9,4,6,8,8,8,6,9},
            {7,4,8,5,3,8,6,9,4,5},
            {8,7,5,7,4,8,2,2,9,8},
            {1,2,6,8,8,2,4,6,2,8},
            {9,2,6,1,3,7,2,3,4,1},
            {5,3,5,3,4,2,4,8,2,5},
            };

void main()

{   int i,j,k,C[10][10];
    FILE *fichero;
```

```
    //OPERACION DE MULTIPLICACION
    for (i=0;i<10;i++)
    {
        for (j=0;j<10;j++)
       {
            C[i][j]=0;
            for (k=0;k<10;k++)
            {
                C[i][j]=C[i][j]+A[i][k]*B[k][j];

            }
       }
    }
    fichero = fopen("ResultadoMultiplica.txt","w");
    for(i=0;i<10;i++)
    {
        for(j=0;j<10;j++)
        {
            fprintf(fichero,"%d\n",C[i][j]);
        }
    }
    fclose(fichero);
    printf("El resultado ha sido almacenado en el archivo Resultado
Multiplica\n");
    return;
}
```

Transpuesta de matrices

```
#include<stdio.h>
int A[10][10]={
            {8,5,6,4,1,8,2,7,3,3},
            {7,5,5,3,3,2,3,1,2,9},
            {3,5,4,2,2,1,4,7,6,7},
            {2,4,5,8,7,1,2,6,5,4},
            {8,8,1,2,7,5,1,5,5,1},
            {5,6,7,5,6,4,2,4,9,5},
            {6,4,9,8,6,2,5,6,5,3},
            {3,5,6,2,8,4,3,4,8,1},
            {1,7,2,4,8,4,9,5,6,3},
            {6,1,4,5,8,8,8,5,7,4},
            };
int B[10][10]={
            {6,4,2,5,6,3,2,6,5,7},

            {2,3,6,1,4,2,1,4,2,5},
            {4,8,7,4,3,7,2,6,3,3},
            {8,6,7,6,4,7,8,4,1,2},
            {1,3,9,4,6,8,8,8,6,9},
            {7,4,8,5,3,8,6,9,4,5},
            {8,7,5,7,4,8,2,2,9,8},
            {1,2,6,8,8,2,4,6,2,8},
            {9,2,6,1,3,7,2,3,4,1},
            {5,3,5,3,4,2,4,8,2,5},
            };
void Matriz(int A[10][10],int k,int C[20][10]);

void main()
{
    int i,j,k=1;
    int C[20][10];
    FILE *fichero;
    Matriz(A,k,C);
    k++;
    Matriz(B,k,C);
```

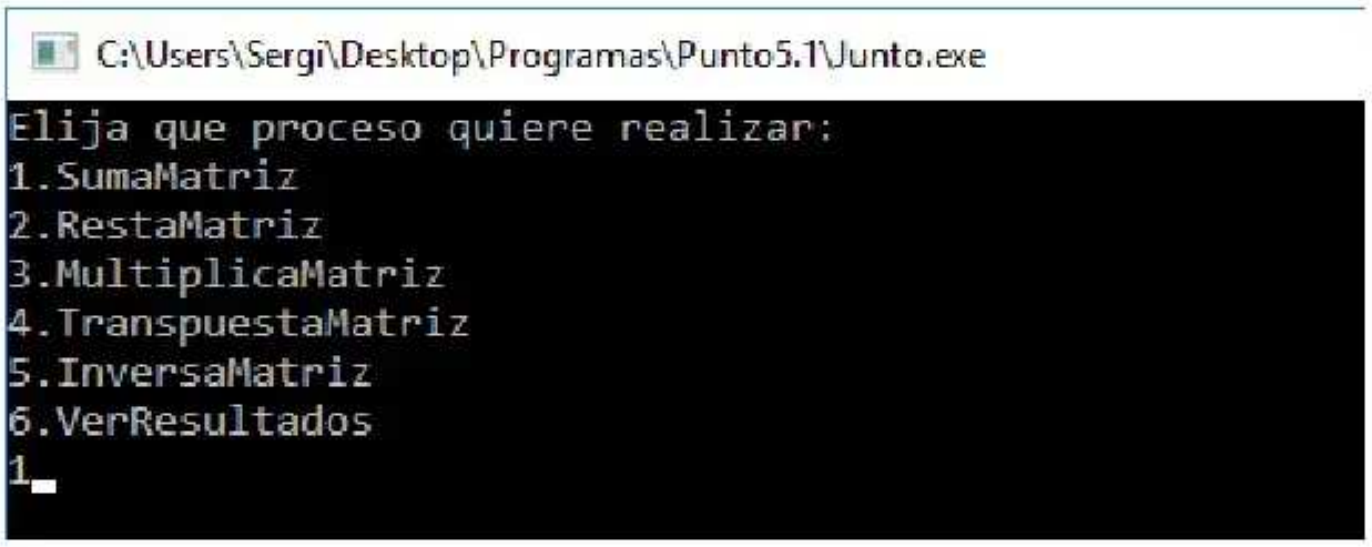
```
    fichero = fopen("ResultadoTranspuesta.txt","w");
    for(i=0;i<20;i++)
    {
        for(j=0;j<10;j++)
        {
            fprintf(fichero,"%d\n",C[i][j]);
        }
    }
    fclose(fichero);
    printf("El resultado ha sido almacenado en el archivo Resultado
Transpuesta\n");
    return;
}

void Matriz(int A[10][10],int k,int C[20][10])
{
    int i,j;
    if(k==1)
    {
        for(i=0;i<10;i++)
        {
            for(j=0;j<10;j++)
            {
                C[i][j] = A[j][i];
            }
        }
    }
    else
    {
        for(i=10;i<20;i++)
        {
            for(j=0;j<10;j++)
            {
                C[i][j] = A[j][i-10];
            }
        }
    }
}
```

Leer los archivos de resultados y mostrarlos en pantalla

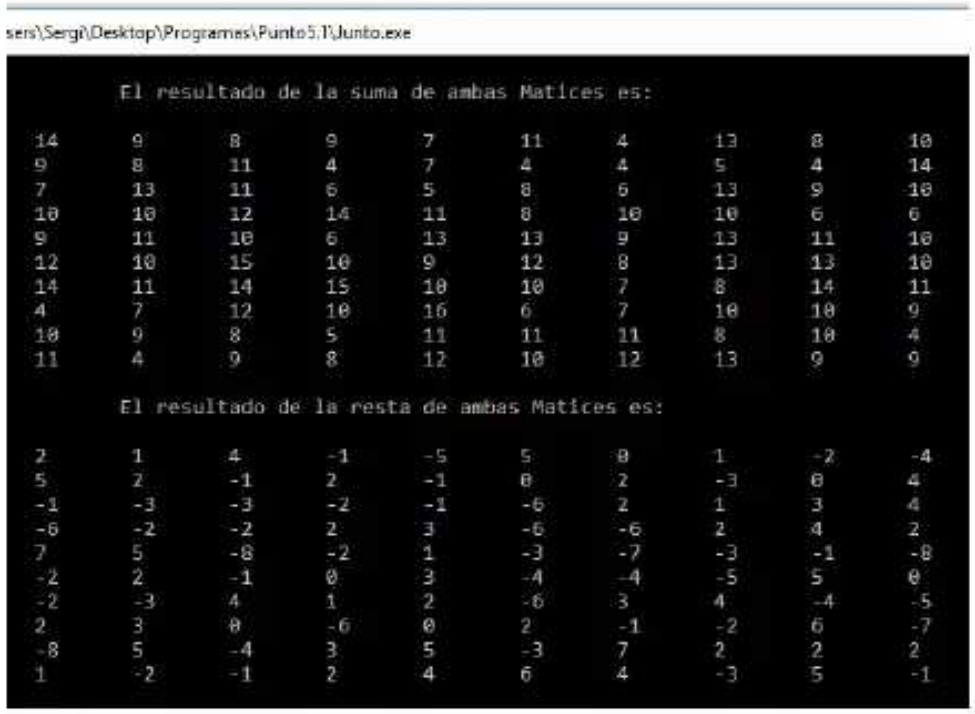
```
#include <stdio.h>
void Imprimir(char nombre[],char operacion[]);
void Imprimir2(char nombre[],char operacion[]);

void main()
{
    Imprimir("ResultadoSuma.txt","suma");
    Imprimir("ResultadoResta.txt","resta");
    Imprimir("ResultadoMultiplica.txt","multiplicacion");
    Imprimir2("ResultadoTranspuesta.txt","transpuesta");
    return;
}

void Imprimir(char nombre[],char operacion[])
{
    FILE *fichero;
    int i,j,numero;
    fichero = fopen(nombre,"r");
    if(fichero==NULL)
    {
        printf("Aun no se ha generado el archivo de resultado correspondiente\n");
    }
    else
```

```
    {
        printf("\n\t\tEl resultado de la %s de ambas Matices es:\n\n",operacion);
        for(i=0;i<10;i++)
        {
            for(j=0;j<10;j++)
            {
                fscanf(fichero,"%d",&numero);
                printf("\t %d",numero);
            }

            printf("\n");
        }
    }
    fclose(fichero);
}

void Imprimir2(char nombre[],char operacion[])
{
    FILE *fichero;
    int i,j,numero;
    fichero = fopen(nombre,"r");
    if(fichero==NULL)
    {
       printf("Aun no se ha generado el archivo de resultado correspondiente\n");
    }
    else
    {
            (                                                   ,         );
        printf("\n\t\tEl resultado de la %s de cada Matriz es:\n\n" operacion
        for(i=0;i<200;i++)
        {

if(i==10||i==20||i==30||i==40||i==50||i==60||i==70||i==80||i==90||i==110||i==120|
|i==130||i==140||i==150||i==160||i==170||i==180||i==190)
            {
                printf("\n");
            }
            if(i==100)
            {
                printf("\n\n");
            }
            fscanf(fichero,"%d",&numero);
            printf("\t %d",numero);
        }
    }
    fclose(fichero);
}
```

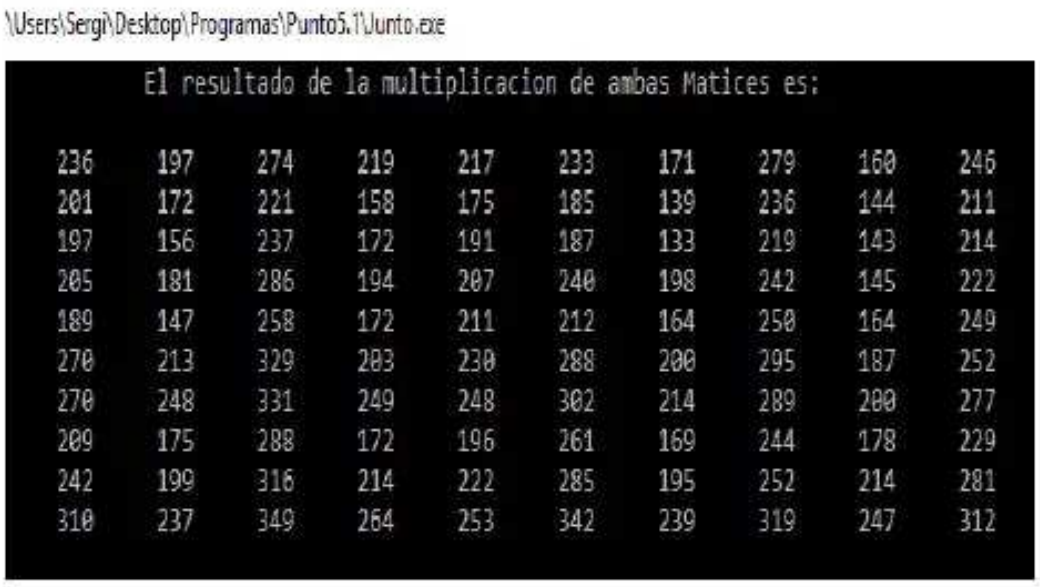
Menú de procesos



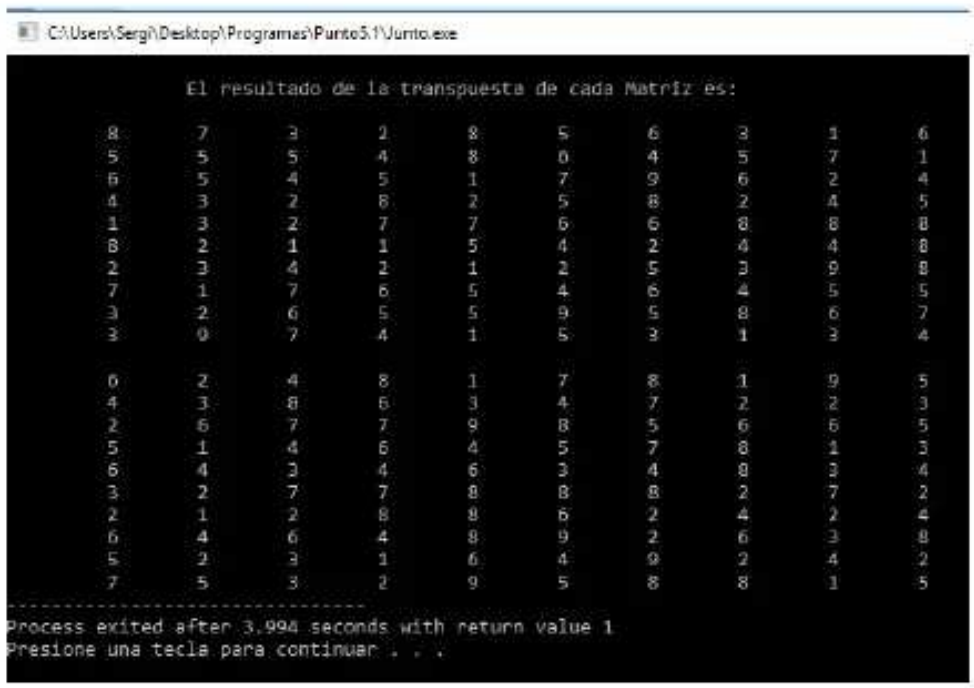
En este tenemos que seleccionar la opcion o el proceso que se va a ejecutar, una vez damos el numero automaticamente hace el proceso y para ver los resultados ingresamos el numero 6 que es la opcion para ver todos aquellos procesos que se ejecutaron, de lo contrario nos saldrá un mensaje de que no se han creado los procesos.



Como se puede apreciar en la imagen y como se explicó anteriormente cuando ingresamos los procesos que vamos a ejecutar estos son los resultados de los procesos de: Suma de matrices y Resta de matrices.



Este es el proceso de la multiplicacion de las matrices A y B (ver figura).



Por último tenemos las matrices transpuestas de la matriz A y de la matriz B, que es el proceso numero 6(ver figura)

Análisis critico

Leonardo Daniel Olvera Aguila

Para realizar esta práctica se tuvo que investigar los procesos en los dos sistemas operativos. Los ejemplos que contenia la práctica y los revisados en el aula de clases fueron de gran utilidad para llevarla a cabo. Pudimos observar también que en el sistema Windows no se pueden realizar las llamadas al proceso por copia exacta de código, debido a la misma configuración del mismo sistema, pero nos ayuda de una manera optima la creacion de procesos por sustitución de código, en cambio en el sistema operativo Linux, se pueden realizar ambas formas de creacion de procesos. Y mediante un control de ejecución de los procesos podemos mantener salidas mas homogeneas y así no tener diferentes salidas en cada ejecución.

Mendoza Parra Sergio

Para la implementación del punto 5 de la práctica fue necesario conocer el manejo de procesos, pues ya que este se llevó a cabo durante toda la práctica, además nos permitió ampliar nuestro conocimiento, reafirmando aún más lo que ya se había obtenido en clase con respecto a las llamadas al sistema, un claro ejemplo es fork (), la cual se utilizó durante toda la práctica tanto para la creación de árboles de procesos como para el desarrollo de los diferentes puntos de la práctica

Paz Sánchez Brandon

El desarrollo de esta práctica permitio la creación de un árbol de procesos por sustitución de código el cual permite brindarle un programa distinto a cada proceso hijo, con ello se observo que la aplicación ejecute cada programa de manera concurrente y rápida. Por otra parte, los procesos que se crearon durante la práctica fueron parte fundamental en la planificación de procesos de un sistema operativo, esto permitio visualizar más detalladamente su funcionalidad.

Conclusiones

Leonardo Daniel Olvera Aguila

Durante el desarrollo de esta práctica se conocieron diversos comportamientos de los procesos, estos dependían de la prioridad que el sistema operativo le otorgara a cada uno, esta observación fue muy importante cuando al momento de compilar se podía verificar que, durante la ejecución de los procesos, estos se ejecutaban alternadamente y no siempre en el mismo orden. Considero que la elaboración de esta práctica fue un gran aporte que nos permitió, en general, familiarizarnos con la funcionalidad de los procesos que ejecuta internamente el sistema operativo, sin embargo, la práctica resulto ser más faci, ya que algunos puntos no lograron ser concluidos como en las anteriores prácticas.

Mendoza Parra Sergio

En la elaboración de esta práctica se emplearon conceptos previamente revisados, además de explorar un nivel más profundo lo que es un sistema operativo. En lo particular los procesos se concluyo que la importancia de los procesos en un sistema operativo, un proceso es "Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistema asociados". Sin la existencia de estos, no se podría tener orden en los procesos de un sistema y probablemente este colapsaría al no saber que acciones ejecutar en determinado tiempo

Paz Sánchez Brandon

La creación de procesos es algo que no tomamos en cuenta cuando programamos aplicaciones. Ahora sabemos cómo podemos crearlos dependiendo del sistema operativo que estemos utilizando. De ahora en adelante es importante que los consideremos, pues resultan bastante útiles para la eficiencia y rapidez de nuestros programas, ya que no se estarían realizando de manera secuencial, sino que daría una apariencia de que están ejecutándose concurrentemente.