Instituto Politécnico Nacional

ESCOM

Practica 4

Administrador de procesos en Linux y Windows

Sistemas Operativos – 2CM17

Integrantes:

Mora Ayala José Antonio

Ramírez Cotonieto Luis Fernando

Torres Carrillo Josehf Miguel Angel

Tovar Jacuinde Rodrigo

Profesor:

Cortes Galicia Jorge

INTRODUCCIÓN

Comunicación Entre Procesos

La comunicación entre procesos, en inglés IPC (Inter-process Communication) es una función básica de los sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers, o a través de las herramientas provistas por las rutinas de IPC. El IPC provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes que ofrece la red subyacente.

La comunicación se establece siguiendo una serie de reglas (protocolos de comunicación). Los protocolos desarrollados para internet son los mayormente usados: IP (capa de red), protocolo de control de transmisión (capa de transporte) y protocolo de transferencia de archivos , protocolo de transferencia de hipertexto (capa de aplicación).

Los procesos pueden ejecutarse en una o más computadoras conectadas a una red. Las técnicas de IPC están divididas dentro de métodos para: paso de mensajes, sincronización, memoria compartida y llamadas de procedimientos remotos (RPC). El método de IPC usado puede variar dependiendo del ancho de banda y latencia (el tiempo desde el pedido de información y el comienzo del envío de la misma) de la comunicación entre procesos, y del tipo de datos que están siendo comunicados.

Tipos de Comunicación

Síncrona

Quien envía permanece bloqueado esperando a que llegue una respuesta del receptor antes de realizar cualquier otro ejercicio.

Asíncrona

Quien envía continúa con su ejecución inmediatamente después de enviar el mensaje al receptor.

Persistente

El receptor no tiene que estar operativo al mismo tiempo que se realiza la comunicación, el mensaje se almacena tanto tiempo como sea necesario para poder ser entregado (Ej.: e-Mail).

Momentánea (transient)

El mensaje se descarta si el receptor no está operativo al tiempo que se realiza la comunicación. Por lo tanto no será entregado.

Directa

Las primitivas permiten enviar y recibir explicitan el nombre del proceso con el que se comunican.
Ejemplo:
enviar (mensaje, A) envía un mensaje al proceso A

Es decir se debe especificar cuál va a ser el proceso fuente y cual va a ser el proceso Destino.
Las operaciones básicas Send y Receive se definen de la siguiente manera: Send (P, mensaje); envía un mensaje al proceso P (P es el proceso destino). Receive (Q, mensaje); espera la recepción de un mensaje por parte del proceso Q (Q es el proceso fuente).

Nota: Receive puede esperar de un proceso cualquiera, un mensaje, pero el Send sí debe especificar a quién va dirigido y cuál es el mensaje.

Indirecta

La comunicación Indirecta: Es aquella donde la comunicación está basada en una herramienta o instrumento ya que el emisor y el receptor están a distancia.

Simétrica

Todos los procesos pueden enviar o recibir. También llamada bidireccional para el caso de dos procesos.

Asimétrica

Un proceso puede enviar, los demás procesos solo reciben. También llamada unidireccional. Suele usarse para hospedar servidores en Internet.

Uso de buffers automático

El transmisor se bloquea hasta que el receptor recibe el mensaje (capacidad cero).

Estos son solo algunos de los conceptos básicos que más adelante en la prácticas e irán describiendo de forma más detallada de igual manera mediante la realización de la misma se podrá comprender de mucho mejor manera gracias a los ejemplos prácticos mediante el uso de código en C

Competencia.

El alumno aprende a familiarizarse con el administrador de procesos del sistema operativo Linux y Windows a través de la creación de
procesos ligeros (hilos) para el desarrollo de aplicaciones concurrentes sencillas.
Desarrollo.

1. A través de la ayuda en línea que proporciona Linux, investigue el funcionamiento de las funciones:
    a. **pthread_create( )** : La función pthread_create() se utiliza para crear un nuevo hilo, con los atributos especificados por attr, dentro de un proceso. Si attr es NULL, se utilizan los atributos por defecto. Si los atributos especificados por attr son modificados posteriormente, los atributos del hilo no se verán afectados. Una vez completada con éxito, pthread_create() almacena el ID del hilo creado en la ubicación referenciada por thread. RETORNO: Si tiene éxito, pthread_create () devuelve 0; en caso de error, devuelve un número de error, y el contenido de * hilo no está definido.
    b. **pthread_join( )** : La función pthread_join () espera el hilo especificado por hilo para terminar. Si ese hilo ya ha terminado, entonces pthread_join () regresa inmediatamente. El hilo especificado por el hilo debe poder unirse. RETORNO: Si tiene éxito, pthread_join () devuelve 0; en caso de error, devuelve un numero erroneo.
    c. **pthread_self( )** : La función pthread_self () devuelve el ID del hilo en el que se invoca. RETORNO: Esta función siempre tiene éxito, devolviendo el ID del hilo de llamada.
    d. **pthread_exit( )** : La función pthread_exit () termina el hilo de llamada y devuelve un valor a través de retval que (si el hilo se puede unir) es disponible para otro hilo en el mismo proceso que llama. RETORNO: Esta función no devuelve a la llamada.
    e. **scandir( )** : La función scandir() lee el directorio dirname y construye una matriz de punteros a las entradas del directorio, utilizando malloc() para asignar el espacio. La función scandir() devuelve el número de entradas en la matriz, y almacena un puntero a la matriz en la ubicación referenciada por namelist.
    Argumentos: 1. Dirname: el nombre del directorio a escanear
    2. *Namelist*: Un puntero a una ubicación donde scandir () puede almacenar un puntero a la matriz de entradas de directorio (de tipo struct direct *) que crea.
    3. *Select*: Un puntero a una subrutina proporcionada por el usuario que scandir () llama para seleccionar qué entradas incluir en la matriz. A la rutina de selección se le pasa un puntero a una entrada de directorio (una estructura dirent) y debe devolver un valor distinto de cero si la entrada de directorio se va a incluir en la matriz.
    4. *Compar*: Un puntero a una subrutina proporcionada por el usuario que se pasa a qsort () para ordenar la matriz completa. Si este puntero es NULL, la matriz no está ordenada.

Puede usar alphasort () como parámetro de comparación para ordenar la matriz alfabéticamente.
RETORNO: El número de entradas en la matriz, o -1 si el directorio no se puede abrir para lectura, o malloc () no puede asignar suficiente memoria para contener todas las estructuras de datos.

f. **stat()** : La función stat () obtendrá información sobre el archivo nombrado y la escribirá en el área señalada por el argumento buf. El argumento de ruta apunta a un nombre de ruta que nombra un archivo. No se requiere permiso de lectura, escritura o ejecución del archivo nombrado. Una implementación que proporciona mecanismos de control de acceso a archivos adicionales o alternativos puede, en condiciones definidas por la implementación, hacer que stat () falle. En particular, el sistema puede negar la existencia del archivo especificado por ruta.
RETORNO: Una vez completado con éxito, se devolverá 0. De lo contrario, se devolverá -1 y errno se establecerá para indicar el error.

Explique los argumentos y retorno de cada función.

2. Capture, compile y ejecute el programa de creación de un nuevo hilo en Linux. Observe su funcionamiento.

```c
#include <stdio.h>
#include <pthread.h>
void *hilo(void *arg);
int main(void)
{
 pthread_t id_hilo;
 pthread_create(&id_hilo, NULL, (void*)hilo, NULL);
 pthread_join(id_hilo, NULL);
 return 0;
}
void *hilo(void *arg)
{
 printf("Hola mundo desde un hilo en UNIX\n");
 return NULL;
}
```

```
┌──(novachrono㉿kali)-[~/practica4]
└─$ gcc hilo1.c -o hilo1 -pthread

┌──(novachrono㉿kali)-[~/practica4]
└─$ ./hilo1
Hola mundo desde un hilo en UNIX
```

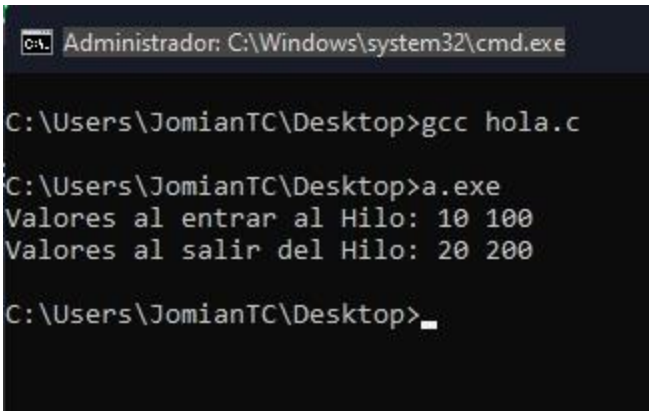
3. Capture, compile y ejecute el siguiente programa de creación de hilos en Linux. Observe su funcionamiento.

```c
#include <stdio.h>
#include <pthread.h>
void *hilo(void *arg)
int main(void)
{
 pthread_t id_hilo;
 char* mensaje="Hola a todos desde el hilo";
 int devolucion_hilo;
 pthread_create(&id_hilo,NULL,hilo,(void*)mensaje);
 pthread_join(id_hilo,(void*)&devolucion_hilo);
 printf("valor = %i\n",devolucion_hilo)
 return 0;
}
void *hilo(void *arg)
{
 char* men;
 int resultado_hilo=0;
 men=(char*)arg;
 printf("%s\n",men);
 resultado_hilo=100;
 pthread_exit((void*)resultado_hilo);
}
```

```
┌──(novachrono㉿kali)-[~/practica4]
└─$ gcc hilo2.c -o hilo2 -pthread
hilo2.c: In function ‘hilo’:
hilo2.c:11:15: warning: cast to pointe
   11 |  pthread_exit((void*)resultado
      |               ^

┌──(novachrono㉿kali)-[~/practica4]
└─$ ./hilo2
Hola a todos desde el hilo
valor = 100
```

4. Capture, compile y ejecute el siguiente programa de creación de hilos en Windows. Observe su funcionamiento.

```
#include <windows.h>
#include <stdio.h>
DWORD WINAPI funcionHilo(LPVOID lpParam);
typedef struct Informacion info;
struct Informacion
{
 int val_1;
 int val_2;
};
int main(void)
{
 DWORD idHilo; /* Identificador del Hilo */
 HANDLE manHilo; /* Manejador del Hilo */
 info argumentos;
 argumentos.val_1=10;
 argumentos.val_2=100;
 // Creacion del hilo
 manHilo=CreateThread(NULL, 0, funcionHilo, &argumentos, 0, &idHilo);

 // Espera la finalización del hilo
 WaitForSingleObject(manHilo, INFINITE);
 printf("Valores al salir del Hilo: %i %i\n", argumentos.val_1,
argumentos.val_2);

 // Cierre del manejador del hilo creado
 CloseHandle(manHilo);
 return 0;
}
DWORD WINAPI funcionHilo(LPVOID lpParam)
{
 info *datos=(info *)lpParam;
 printf("Valores al entrar al Hilo: %i %i\n", datos->val_1,
datos->val_2);
 datos->val_1*=2;
 datos->val_2*=2;
 return 0;
}
```



5. Programe una aplicación (tanto en Linux como en Windows), que cree un proceso hijo a partir de un proceso padre, el hijo creado a su vez creará 15 hilos. A su vez cada uno de los 15 hilos creará 10 hilos más. A su vez cada uno de los 10 hilos creará 5 hilos más. Cada uno de los hilos creados imprimirá en pantalla "Práctica 4" si se trata de un hilo terminal o los identificadores de los hilos creados si se trata de un proceso o hilo padre.

Seccion Linux

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h>

int i,j,k;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* hilo1(){
    pthread_mutex_lock(&mutex);
    printf("%d primer nivel, hilo %d\n",j,pthread_self());
    pthread_mutex_unlock(&mutex);
}
void* hilo2(){
    pthread_mutex_lock(&mutex);
    printf("----%dsegundo nivel, hilo %d\n",i,pthread_self());
    pthread_mutex_unlock(&mutex);
}
void* hilo3(){
    pthread_mutex_lock(&mutex);
    printf(" -----------%dtercer nivel Practica 2\n",k,pthread_self());
    pthread_mutex_unlock(&mutex);
}
```

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

        pid_t pid1;
        pthread_t th[1000];
        pthread_mutex_init(&mutex, NULL);

    if((pid1 = fork()) == 0){
        printf("soy el proceso hijo\n");
        for (j = 0; j < 15; j++)
        {
            pthread_create(th + j, NULL, &hilo1, NULL);
            for (i = 0; i < 10; i++)
            {
                pthread_create(th + i, NULL, &hilo2, NULL);
                for (k = 1; k < 6; k++)
                {
                    pthread_create(th + k, NULL, &hilo3, NULL);
                    pthread_join(th[k],NULL);
                }
                pthread_join(th[i],NULL);
            }
            pthread_join(th[j],NULL);
        }
    }
    return 0;
}
```

```
┌──(novachrono㉿kali)-[~/practica4]
└─$ gcc arbol.c -o arbol -pthread

┌──(novachrono㉿kali)-[~/practica4]
└─$ ./arbol
```

```
14 primer nivel, hilo 1059489536
├──0segundo nivel, hilo 883242752
│  ├─────────1tercer nivel Practica 2
│  ├─────────2tercer nivel Practica 2
│  ├─────────3tercer nivel Practica 2
│  ├─────────4tercer nivel Practica 2
│  └─────────5tercer nivel Practica 2
├──1segundo nivel, hilo 883242752
│  ├─────────1tercer nivel Practica 2
│  ├─────────2tercer nivel Practica 2
│  ├─────────3tercer nivel Practica 2
│  ├─────────4tercer nivel Practica 2
│  └─────────5tercer nivel Practica 2
├──2segundo nivel, hilo 874850048
│  ├─────────1tercer nivel Practica 2
│  ├─────────2tercer nivel Practica 2
│  ├─────────3tercer nivel Practica 2
│  ├─────────4tercer nivel Practica 2
│  └─────────5tercer nivel Practica 2
├──3segundo nivel, hilo 866457344
│  ├─────────1tercer nivel Practica 2
│  ├─────────2tercer nivel Practica 2
│  ├─────────3tercer nivel Practica 2
│  ├─────────4tercer nivel Practica 2
│  └─────────5tercer nivel Practica 2
├──4segundo nivel, hilo 858064640
│  ├─────────1tercer nivel Practica 2
│  ├─────────2tercer nivel Practica 2
│  ├─────────3tercer nivel Practica 2
│  ├─────────4tercer nivel Practica 2
│  └─────────5tercer nivel Practica 2
└──5segundo nivel, hilo 849671936
   ├─────────1tercer nivel Practica 2
   ├─────────2tercer nivel Practica 2
```

Programa Windows

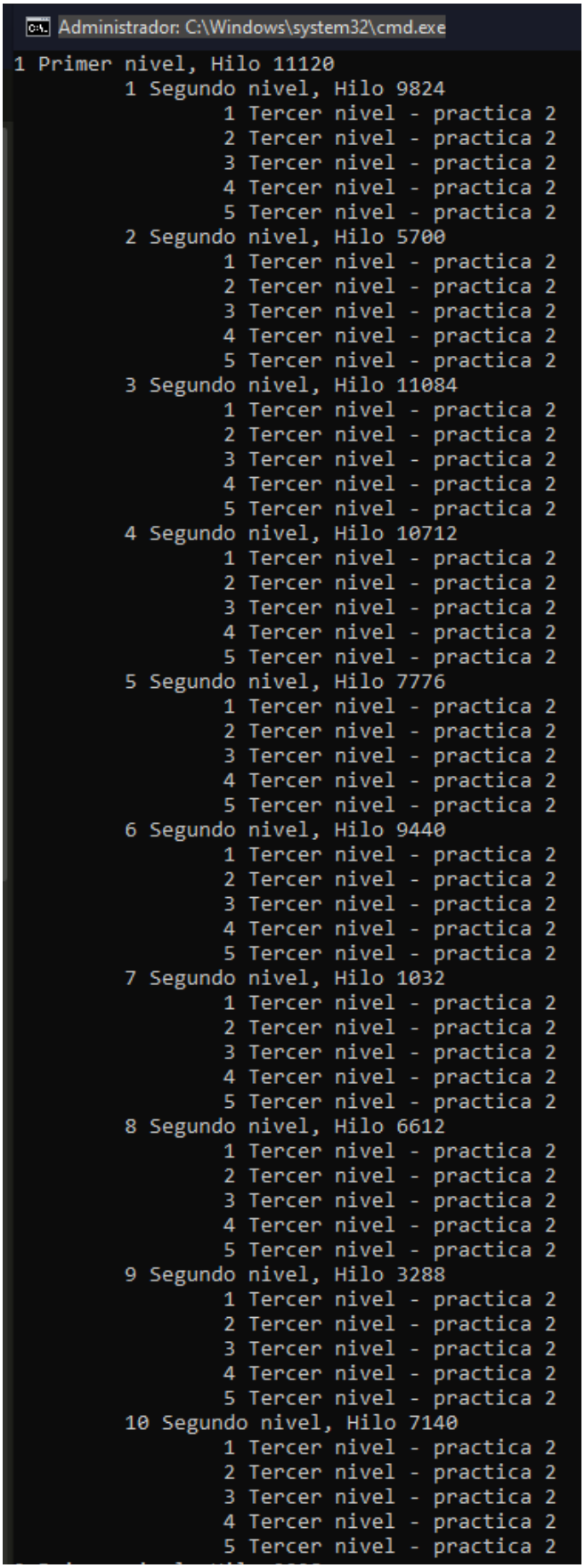
Programa hijo

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

DWORD WINAPI funcionHilo1(LPVOID lpParam);
DWORD WINAPI funcionHilo2(LPVOID lpParam);
DWORD WINAPI funcionHilo3(LPVOID lpParam);

int i, j, k;
DWORD idHilo1, idHilo2, idHilo3;
HANDLE manHilo1, manHilo2, manHilo3;

int main(){

    for (i = 1; i < 16; i++){

        manHilo1 = CreateThread(NULL, 0, funcionHilo1, NULL, 0, &idHilo1);

        for (j = 1; j < 11; j++){

            manHilo2 = CreateThread(NULL, 0, funcionHilo2, NULL, 0, &idHilo2);

            for (k = 1; k < 6; k++){

                manHilo3 = CreateThread(NULL, 0, funcionHilo3, NULL, 0, &idHilo3);

                WaitForSingleObject(manHilo3, INFINITE);

                CloseHandle(manHilo3);
            }

            WaitForSingleObject(manHilo2, INFINITE);

            CloseHandle(manHilo2);

        }

        WaitForSingleObject(manHilo1, INFINITE);

        CloseHandle(manHilo1);
    }

    return 0;
}

DWORD WINAPI funcionHilo1(LPVOID lpParam){

    printf("%d Primer nivel, Hilo %d\n", i, GetCurrentThreadId());

    return 0;
}

DWORD WINAPI funcionHilo2(LPVOID lpParam){

    printf("\t %d Segundo nivel, Hilo %d\n", j, GetCurrentThreadId());

    return 0;
}

DWORD WINAPI funcionHilo3(LPVOID lpParam){

    printf("\t\t %d Tercer nivel - practica 2\n", k);

    return 0;
}
```

Programa padre

```c
#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(!CreateProcess(NULL, "hijo", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){

        printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );
    }

    WaitForSingleObject(pi.hProcess, INFINITE);

    CloseHandle(pi.hProcess);

    CloseHandle(pi.hThread);

    exit(0);

    return 0;
}
```

Ejecución


Administrador: C:\Windows\system32\cmd.exe
1 Primer nivel, Hilo 11120
1 Segundo nivel, Hilo 9824
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
2 Segundo nivel, Hilo 5700
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
3 Segundo nivel, Hilo 11084
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
4 Segundo nivel, Hilo 10712
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
5 Segundo nivel, Hilo 7776
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
6 Segundo nivel, Hilo 9440
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
7 Segundo nivel, Hilo 1032
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
8 Segundo nivel, Hilo 6612
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
9 Segundo nivel, Hilo 3288
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2
10 Segundo nivel, Hilo 7140
1 Tercer nivel - practica 2
2 Tercer nivel - practica 2
3 Tercer nivel - practica 2
4 Tercer nivel - practica 2
5 Tercer nivel - practica 2

5. Programe la misma aplicación del punto 5 de la práctica 3 pero utilizando hilos (tanto en Linux como en Windows) en vez de procesos. Compare ambos programas (el creado en la práctica 3 y el creado en esta práctica) y dé sus observaciones tanto de funcionamiento como de los tiempos de ejecución resultantes.

Windows

```c
#include<stdio.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<time.h>
#include<stdlib.h>
#include<string.h>
#include<windows.h>
#define TAM 7

DWORD WINAPI sumaMatrices(LPVOID lpParam);
DWORD WINAPI restaMatrices(LPVOID lpParam);
DWORD WINAPI multiplicacionMatrices(LPVOID lpParam);
DWORD WINAPI transpuesta(LPVOID lpParam);
DWORD WINAPI hiloImprimir(LPVOID lpParam);
void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]);
void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]);
void imprimirMatrizT(int archivo_abierto,int matriz[TAM][TAM]);
void imprimir(char archivo[100]);
int main(int argc, char const *argv[])
{
    clock_t t_ini, t_fin;
    double secs;
    t_ini=clock();
    //Hilo long unsigned
    DWORD idHilo[5];
    HANDLE manHilo[5];
    //Hilo para la suma
    manHilo[0]=CreateThread(NULL,0,sumaMatrices,NULL,0,&idHilo[0]);
    //Hilo para la resta
    manHilo[1]=CreateThread(NULL,0,restaMatrices,NULL,0,&idHilo[1]);
    //Hilo para la multiplicacion
    manHilo[2]=CreateThread(NULL,0,multiplicacionMatrices,NULL,0,&idHilo[2]);
    //Creo un hilo para la transpuesta
    manHilo[3]=CreateThread(NULL,0,transpuesta,NULL,0,&idHilo[3]);
    //Creo un hilo para imprimir
    manHilo[4]=CreateThread(NULL,0,hiloImprimir,NULL,0,&idHilo[4]);
    //Espera la finalización del hilo
    int i;
    for(i=0;i<5;i++){
       WaitForSingleObject(manHilo[i],INFINITE);
    }
    //Cierre del manejador del hilo creado
    for(i=0;i<5;i++){
        CloseHandle(manHilo[i]);
    }
    t_fin = clock();
    secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
    printf("%.0016g segundos\n", secs * 1000.0);
    return 0;
}
```

```c
DWORD WINAPI sumaMatrices(LPVOID lpParam){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            matrizresul[i][j]=matriz1[i][j]+matriz2[i][j];
        }
    }
    imprimirMatriz("Suma.txt",matrizresul);
}
DWORD WINAPI restaMatrices(LPVOID lpParam){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            matrizresul[i][j]=matriz1[i][j]-matriz2[i][j];
        }
    }
```

```c
    imprimirMatriz("Resta.txt",matrizresul);
}
DWORD WINAPI multiplicacionMatrices(LPVOID lpParam){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    int i,j,k=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            //Inicializa valor en cero
            matrizresul[i][j]=0;
            for(k=0;k<10;k++){
                matrizresul[i][j]+=matriz1[i][k]*matriz2[k][j];
            }
        }
    }
    imprimirMatriz("Multiplicacion.txt",matrizresul);
}
DWORD WINAPI transpuesta(LPVOID lpParam){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
```

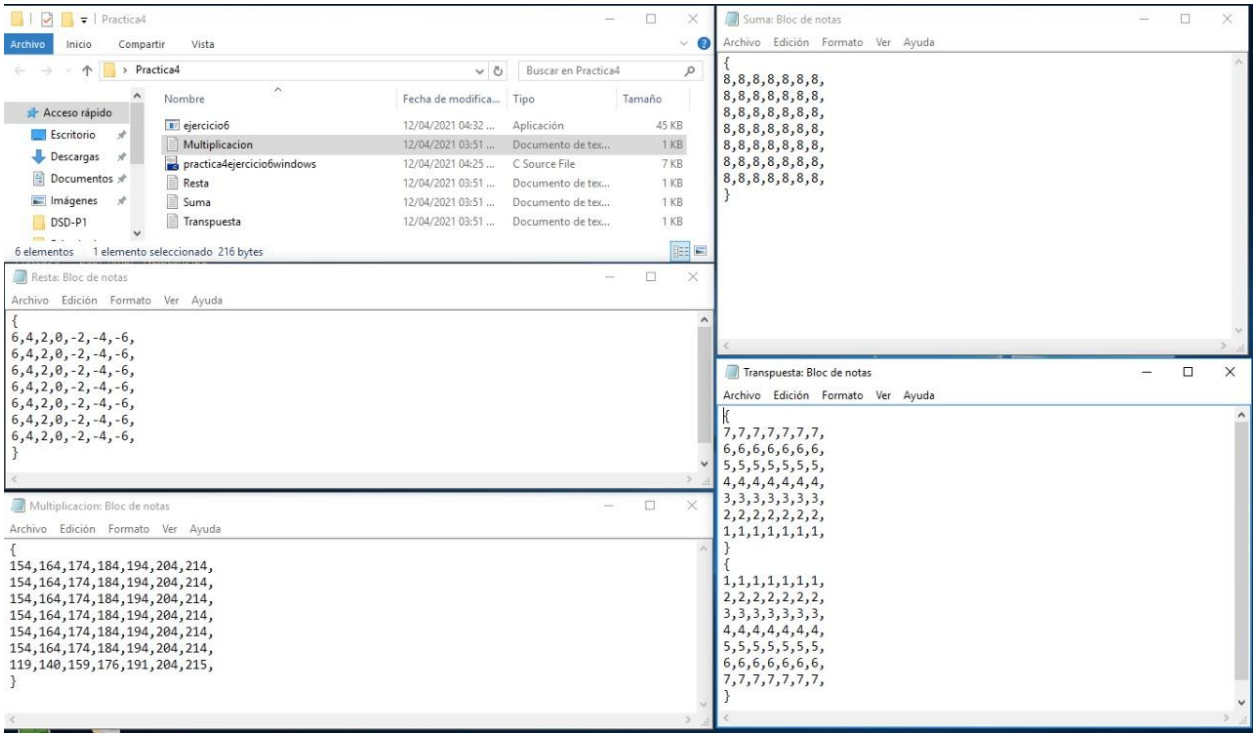
```c
    int matrizresul[TAM][TAM];
    transponerMatriz(matriz1,matrizresul);
    struct stat estatus;
    int archivo_abierto=0;
    char archivo[100]="Transpuesta.txt";
    archivo_abierto=open(archivo,O_RDWR|O_CREAT);
    imprimirMatrizT(archivo_abierto,matrizresul);
    transponerMatriz(matriz2,matrizresul);
    imprimirMatrizT(archivo_abierto,matrizresul);
    close(archivo_abierto);
}
DWORD WINAPI hiloImprimir(LPVOID lpParam){
    imprimir("Suma.txt");
    imprimir("Resta.txt");
    imprimir("Multiplicacion.txt");
    imprimir("Transpuesta.txt");
}

void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]){
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            matrizresul[i][j]=matriz[j][i];
        }
    }
}
void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]){
    struct stat estatus;
    int archivo_abierto=0;
    char contenido[2];
    archivo_abierto=open(archivo,O_RDWR|O_CREAT);
    write(archivo_abierto, "{\n", strlen("{\n"));
    //puts("{");
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            sprintf(contenido, "%i", matriz[i][j]);
            write(archivo_abierto, contenido, strlen(contenido));
            //printf("%i", matriz[i][j]);
            if(i==9 && j==9){
            }else{
                //printf(",");
                write(archivo_abierto, ",", strlen(","));
            }
        }
        write(archivo_abierto, "\n", strlen("\n"));
        //puts("\n");
    }
    //puts("}");
    write(archivo_abierto, "}\n", strlen("}\n"));
    close(archivo_abierto);
}
void imprimirMatrizT(int archivo_abierto,int matriz[TAM][TAM]){
```

```
212     //printf("%i\n", archivo_abierto);
213     char contenido[2];
214     write(archivo_abierto, "{\n", strlen("{\n"));
215     //puts("{");
216     int i,j=0;
217     for(i=0;i<TAM;i++){
218         for(j=0;j<TAM;j++){
219             sprintf(contenido, "%i", matriz[i][j]);
220             write(archivo_abierto, contenido, strlen(contenido));
221             //printf("%i", matriz[i][j]);
222             if(i==9 && j==9){
223             }else{
224                 //printf(",");
225                 write(archivo_abierto, ",", strlen(","));
226             }
227         }
228         write(archivo_abierto, "\n", strlen("\n"));
229         //puts("\n");
230     }
231     //puts("}");
232     write(archivo_abierto, "}\n", strlen("}\n"));
233 }
234 void imprimir(char archivo[100]){
235     struct stat estatus;
236     int archivo_abierto=0;
237     archivo_abierto=open(archivo,O_RDONLY);
238     stat(archivo,&estatus);
239     char contenido[estatus.st_size];
240     read(archivo_abierto, contenido, estatus.st_size);
241     int i=0;
242     printf("Contenido de %s:\n", archivo);
243     while ((*(contenido + i) != '\0') && i+1 <=estatus.st_size) {
244         if(*(contenido +i) > 0){
245             printf("%c", *(contenido+i));
246             if(*(contenido+i)=='}'){
247                 puts("");
248                 break;
249             }
250         }
251         i++;
252     }
253     close(archivo_abierto);
254 }
255
```

Compilación

```
Símbolo del sistema
(c) 2016 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Luis Coto>cd Desktop

C:\Users\Luis Coto\Desktop>cd Practica4

C:\Users\Luis Coto\Desktop\Practica4>gcc practica4ejercicio6windows.c -o ejercicio6

C:\Users\Luis Coto\Desktop\Practica4>"C:\Users\Luis Coto\Desktop\Practica4\ejercicio6.exe"
Contenido de Suma.txt:
{
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
}
Contenido de Resta.txt:
{
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
}
Contenido de Multiplicacion.txt:
{
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
119,140,159,176,191,204,215,
}
Contenido de Transpuesta.txt:
{
7,7,7,7,7,7,7,
6,6,6,6,6,6,6,
5,5,5,5,5,5,5,
4,4,4,4,4,4,4,
3,3,3,3,3,3,3,
2,2,2,2,2,2,2,
1,1,1,1,1,1,1,
}
106 segundos

C:\Users\Luis Coto\Desktop\Practica4>_
Escribe aquí para buscar
04:32 p. m.
12/04/2021
```

Archivos creados



Linux

```c
#include<stdio.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<time.h>
#include<stdlib.h>
#include<string.h>
#include<sys/wait.h>
#include<pthread.h>
#define TAM 7
void *sumaMatrices(void *arg);
void *restaMatrices(void *arg);
void *multiplicacionMatrices(void *arg);
void *transpuesta(void *arg);
void *hiloImprimir(void *arg);
void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]);
void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]);
void imprimirMatrizT(int archivo_abierto,int matriz[TAM][TAM]);
void imprimir(char archivo[100]);
int main(int argc, char const *argv[])
{
    clock_t t_ini, t_fin;
    double secs;
    t_ini=clock();
    pthread_t id_hilo[5];
    //Hilo para la suma
    pthread_create(&id_hilo[0],NULL,sumaMatrices,NULL);
    pthread_join(id_hilo[0],NULL);
    //Hilo para la resta
    pthread_create(&id_hilo[1],NULL,restaMatrices,NULL);
    pthread_join(id_hilo[1],NULL);
    //Hilo para la multiplicacion
    pthread_create(&id_hilo[2],NULL,multiplicacionMatrices,NULL);
    pthread_join(id_hilo[2],NULL);
    //Hilo para la transpuesta
    pthread_create(&id_hilo[3],NULL,transpuesta,NULL);
    pthread_join(id_hilo[3],NULL);
    //Hilo para la transpuesta
    pthread_create(&id_hilo[4],NULL,hiloImprimir,NULL);
    pthread_join(id_hilo[4],NULL);
    t_fin = clock();
    secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
    printf("%.16g segundos\n", secs * 1000.0);
    return 0;
}
void *sumaMatrices(void *arg){
}
void *sumaMatrices(void *arg){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            matrizresul[i][j]=matriz1[i][j]+matriz2[i][j];
        }
    }
    imprimirMatriz("Suma.txt",matrizresul);
}
void *restaMatrices(void *arg){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
```

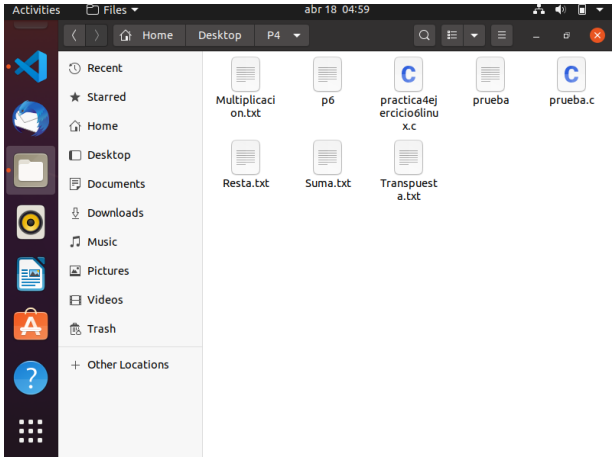
```
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            matrizresul[i][j]=matriz1[i][j]-matriz2[i][j];
        }
    }
    imprimirMatriz("Resta.txt",matrizresul);
}
void *multiplicacionMatrices(void *arg){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    int i,j,k=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            //Inicia array en cero
            matrizresul[i][j]=0;
            for(k=0;k<10;k++){
                matrizresul[i][j]+=matriz1[i][k]*matriz2[k][j];
            }
        }
    }
    imprimirMatriz("Multiplicacion.txt",matrizresul);
}
void *transpuesta(void *arg){
        int matriz1[TAM][TAM]={
        {7,6,5,4,3,2,1},
```

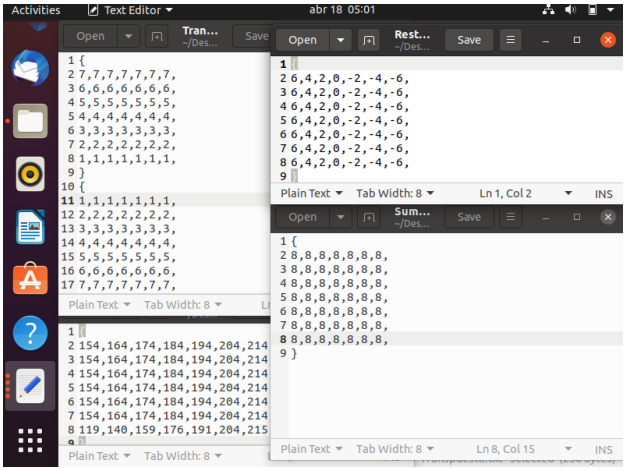
```
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1},
        {7,6,5,4,3,2,1}
    };
    int matriz2[TAM][TAM]={
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7},
        {1,2,3,4,5,6,7}
    };
    int matrizresul[TAM][TAM];
    transponerMatriz(matriz1,matrizresul);
    struct stat estatus;
    int archivo_abierto=0;
    char archivo[100]="Transpuesta.txt";
    archivo_abierto=open(archivo,O_RDWR|O_CREAT);
    imprimirMatrizT(archivo_abierto,matrizresul);
    transponerMatriz(matriz2,matrizresul);
    imprimirMatrizT(archivo_abierto,matrizresul);
    close(archivo_abierto);
}
void *hiloImprimir(void *arg){
    imprimir("Suma.txt");
    imprimir("Resta.txt");
    imprimir("Multiplicacion.txt");
    imprimir("Transpuesta.txt");
}

void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]){
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            matrizresul[i][j]=matriz[j][i];
        }
    }
}
void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]){
    struct stat estatus;
    int archivo_abierto=0;
    char contenido[2];
```

```
    char contenido[2];
    archivo_abierto=open(archivo,O_RDWR|O_CREAT);
    write(archivo_abierto, "{\n", strlen("{\n"));
    //puts("{");
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            sprintf(contenido, "%i", matriz[i][j]);
            write(archivo_abierto, contenido, strlen(contenido));
            //printf("%i", matriz[i][j]);
            if(i==9 && j==9){
            }else{
                //printf(",");
                write(archivo_abierto, ",", strlen(","));
            }
        }
        write(archivo_abierto, "\n", strlen("\n"));
        //puts("\n");
    }
    //puts("}");
    write(archivo_abierto, "}\n", strlen("}\n"));
    close(archivo_abierto);
}
void imprimirMatrizT(int archivo_abierto,int matriz[TAM][TAM]){
    //printf("%i\n", archivo_abierto);
    char contenido[2];
    write(archivo_abierto, "{\n", strlen("{\n"));
    //puts("{");
    int i,j=0;
    for(i=0;i<TAM;i++){
        for(j=0;j<TAM;j++){
            sprintf(contenido, "%i", matriz[i][j]);
            write(archivo_abierto, contenido, strlen(contenido));
            //printf("%i", matriz[i][j]);
            if(i==9 && j==9){
            }else{
                //printf(",");
                write(archivo_abierto, ",", strlen(","));
            }
        }
        write(archivo_abierto, "\n", strlen("\n"));
        //puts("\n");
    }
    //puts("}");
    write(archivo_abierto, "}\n", strlen("}\n"));
}
```

```
void imprimir(char archivo[100]){
    struct stat estatus;
    int archivo_abierto=0;
    archivo_abierto=open(archivo,O_RDONLY);
    stat(archivo,&estatus);
    char contenido[estatus.st_size];
    read(archivo_abierto, contenido, estatus.st_size);
    int i=0;
    printf("Contenido de %s:\n", archivo);
    while ((*(contenido + i) != '\0') && i+1 <=estatus.st_size) {
        if(*(contenido +i) > 0){
            printf("%c", *(contenido+i));
            if(*(contenido+i)=='}'){
                puts("");
                break;
            }
        }
        i++;
    }
    close(archivo_abierto);
}
```

Archivos

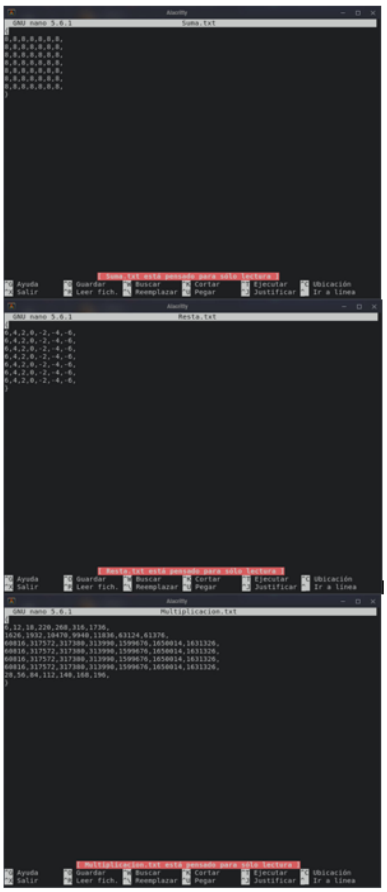




Compilación

```
Contenido de Suma.txt:
{
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
8,8,8,8,8,8,8,
}
Contenido de Resta.txt:
{
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
6,4,2,0,-2,-4,-6,
}
Contenido de Multiplicacion.txt:
{
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
154,164,174,184,194,204,214,
119,140,159,176,191,204,215,
}
Contenido de Transpuesta.txt:
{
7,7,7,7,7,7,7,
6,6,6,6,6,6,6,
5,5,5,5,5,5,5,
4,4,4,4,4,4,4,
3,3,3,3,3,3,3,
2,2,2,2,2,2,2,
1,1,1,1,1,1,1,
}
106 segundos
```

Contenido de archivos



Comparación:

Los hilos permiten que varios procesos se puedan llevar a cabo al mismo tiempo, por lo que podríamos decir que estamos "optimizando" el código realizado en la práctica anterior por algunos segundos.

CONCLUSIONES

Esta práctica, como el resto de las mismas, me dio la oportunidad de poder conocer más acerca de la utilización que tienen los hilos, permitiendo adquirir una idea más clara de cómo estos pueden ser implementados.
La práctica proporcionó bastantes conocimientos tanto prácticos como teóricos, pues los documentos proporcionados por el profesor así como la actividad designada para cada uno de ellos permite esclarecer mucho más el concepto de lo que estamos tratando y conocer de raíz como estos funcionan.
Personalmente me gustó más la forma de trabajar con el entorno de Windows

-Mora Ayala José Antonio

Esta práctica me permitió comprender el uso de los hilos para permitirnos la optimización de tiempos durante el diseño de un algoritmo y la ejecución de un programa. Su análisis me resulta interesante, pues con respecto al caso temporal podemos encontrar ciertas diferencias debido a que los diversos procesos se realizan a la par unos con otros, y esto mejora el rendimiento. Encuentro una información valiosa que sin duda estaré poniendo en práctica para códigos a futuro. Para su implementación, curiosamente me agrado más utilizar windows, se me hizo un entorno más amigable.

-Ramírez Cotonieto Luis Fernando

Esta práctica me ayudó a entender un poco más sobre los hilos al menos en el lenguaje C, en Java usamos los hilos para crear temporizadores pero jamas crei que los podíamos usar como un tipo de subproceso sin serlo, es decir crear un rutina dentro de un proceso pero sin crearlo, es bastante curioso como funcionan y ver lo distinta que es la forma de programarlo en Windows y Linux, sin lugara duda una función y forma de programar interesante que tiene muchas aplicaciones cuando estamos creando los programas para el sistema

-Torres Carrillo Josehf Miguel Angel

Después de haber realizado esta práctica me doy cuenta de la utilidad que tienen los hilos, porque estos me permiten tener una ejecución simultánea, además también es muy sencillo implementarlos, aunque en esta práctica me atore un poco viendo cómo darle un orden jerárquico, pero investigando y leyendo encontre que estos no son jerárquicos sino que son planos, pero gracias a la función pthread_join me pude apoyar para darle el enfoque que necesitaba. En lo personal me agrado mucho el cómo es posible trabajar con ellos y estoy seguro que buscaré el modo de implementarlos en más lenguajes de programación y también ocuparlos en proyectos futuros.

-Rodrigo Tovar Jacuinde