



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 dirent *) 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.

```
#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.

```
#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 pointer from integer of type 'int' is not safe
11 | pthread_exit((void*)resultado_hilo);
    |               ^
(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;
}
```

```
C:\Users\JomianTC\Desktop>gcc hola.c
C:\Users\JomianTC\Desktop>a.exe
Valores al entrar al Hilo: 10 100
Valores al salir del Hilo: 20 200
C:\Users\JomianTC\Desktop>
```

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

```
#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);
}
```



```

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

```
1  #include <windows.h>
2  #include <stdio.h>
3
4  DWORD WINAPI funcionHilo1(LPVOID lpParam);
5  DWORD WINAPI funcionHilo2(LPVOID lpParam);
6  DWORD WINAPI funcionHilo3(LPVOID lpParam);
7
8  int i, j, k;
9  DWORD idHilo1, idHilo2, idHilo3;
10 HANDLE manHilo1, manHilo2, manHilo3;
11
12 int main(){
13
14     for (i = 1; i < 16; i++){
15
16         manHilo1 = CreateThread(NULL, 0, funcionHilo1, NULL, 0, &idHilo1);
17
18         for (j = 1; j < 11; j++){
19
20             manHilo2 = CreateThread(NULL, 0, funcionHilo2, NULL, 0, &idHilo2);
21
22             for (k = 1; k < 6; k++){
23
24                 manHilo3 = CreateThread(NULL, 0, funcionHilo3, NULL, 0, &idHilo3);
25
26                 WaitForSingleObject(manHilo3, INFINITE);
27
28                 CloseHandle(manHilo3);
29             }
30
31             WaitForSingleObject(manHilo2, INFINITE);
32
33             CloseHandle(manHilo2);
34
35         }
36
37         WaitForSingleObject(manHilo1, INFINITE);
38
39         CloseHandle(manHilo1);
40     }
41
42     return 0;
43 }
44
45 DWORD WINAPI funcionHilo1(LPVOID lpParam){
46
47     printf("%d Primer nivel, Hilo %d\n", i, GetCurrentThreadId());
48
49     return 0;
50 }
51
52 DWORD WINAPI funcionHilo2(LPVOID lpParam){
53
54     printf("\t %d Segundo nivel, Hilo %d\n", j, GetCurrentThreadId());
55
56     return 0;
57 }
58
59 DWORD WINAPI funcionHilo3(LPVOID lpParam){
60
61     printf("\t\t %d Tercer nivel - practica 2\n", k);
62
63     return 0;
64 }
```

Programa padre

```
1  #include <windows.h>
2  #include <stdio.h>
3
4  int main(int argc, char *argv[]){
5
6      STARTUPINFO si;
7      PROCESS_INFORMATION pi;
8
9      int i;
10     ZeroMemory(&si, sizeof(si));
11     si.cb = sizeof(si);
12     ZeroMemory(&pi, sizeof(pi));
13
14     if(!CreateProcess(NULL, "hijo", NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)){
15
16         printf( "Fallo al invocar CreateProcess (%d)\n", GetLastError() );
17     }
18
19     WaitForSingleObject(pi.hProcess, INFINITE);
20
21     CloseHandle(pi.hProcess);
22
23     CloseHandle(pi.hThread);
24
25     exit(0);
26
27     return 0;
28 }
```

Ejecución

```
C:\> 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

```

1 #include<stdio.h>
2 #include<sys/stat.h>
3 #include<fcntl.h>
4 #include<unistd.h>
5 #include<time.h>
6 #include<stdlib.h>
7 #include<string.h>
8 #include<windows.h>
9 #define TAM 7
10
11 DWORD WINAPI sumaMatrices(LPVOID lpParam);
12 DWORD WINAPI restaMatrices(LPVOID lpParam);
13 DWORD WINAPI multiplicacionMatrices(LPVOID lpParam);
14 DWORD WINAPI transpuesta(LPVOID lpParam);
15 DWORD WINAPI hiloImprimir(LPVOID lpParam);
16 void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]);
17 void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]);
18 void imprimirMatrizT(int archivo_abierto,int matriz[TAM][TAM]);
19 void imprimir(char archivo[100]);
20 int main(int argc, char const *argv[])
21 {
22     clock_t t_ini, t_fin;
23     double secs;
24     t_ini=clock();
25     //Hilo para asignar
26     DWORD idHilo[5];
27     HANDLE manHilo[5];
28     //Hilo para la suma
29     manHilo[0]=CreateThread(NULL,0,sumaMatrices,NULL,0,&idHilo[0]);
30     //Hilo para la resta
31     manHilo[1]=CreateThread(NULL,0,restaMatrices,NULL,0,&idHilo[1]);
32     //Hilo para la multiplicacion
33     manHilo[2]=CreateThread(NULL,0,multiplicacionMatrices,NULL,0,&idHilo[2]);
34     //Crea un hilo para la transpuesta
35     manHilo[3]=CreateThread(NULL,0,transpuesta,NULL,0,&idHilo[3]);
36     //Crea un hilo para imprimir
37     manHilo[4]=CreateThread(NULL,0,hiloImprimir,NULL,0,&idHilo[4]);
38     //Espera la finalización del hilo
39     int i;
40     for(i=0;i<5;i++){
41         WaitForSingleObject(manHilo[i],INFINITE);
42     }
43     //Cierra del manejador del hilo creado
44     for(i=0;i<5;i++){
45         CloseHandle(manHilo[i]);
46     }
47     t_fin = clock();
48     secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
49     printf("%.0016g segundos\n", secs * 1000.0);
50     return 0;
51 }
52
53 DWORD WINAPI sumaMatrices(LPVOID lpParam){
54     int matriz1[TAM][TAM]={
55         {7,6,5,4,3,2,1},
56         {7,6,5,4,3,2,1},
57         {7,6,5,4,3,2,1},
58         {7,6,5,4,3,2,1},
59         {7,6,5,4,3,2,1},
60         {7,6,5,4,3,2,1},
61         {7,6,5,4,3,2,1}
62     };
63     int matriz2[TAM][TAM]={
64         {1,2,3,4,5,6,7},
65         {1,2,3,4,5,6,7},
66         {1,2,3,4,5,6,7},
67         {1,2,3,4,5,6,7},
68         {1,2,3,4,5,6,7},
69         {1,2,3,4,5,6,7},
70         {1,2,3,4,5,6,7}
71     };
72     int matrizresul[TAM][TAM];
73     int i,j=0;
74     for(i=0;i<TAM;i++){
75         for(j=0;j<TAM;j++){
76             matrizresul[i][j]=matriz1[i][j]+matriz2[i][j];
77         }
78     }
79     imprimirMatriz("Suma.txt",matrizresul);
80 }
81
82 DWORD WINAPI restaMatrices(LPVOID lpParam){
83     int matriz1[TAM][TAM]={
84         {7,6,5,4,3,2,1},
85         {7,6,5,4,3,2,1},
86         {7,6,5,4,3,2,1},
87         {7,6,5,4,3,2,1},
88         {7,6,5,4,3,2,1},
89         {7,6,5,4,3,2,1},
90         {7,6,5,4,3,2,1}
91     };
92     int matriz2[TAM][TAM]={
93         {1,2,3,4,5,6,7},
94         {1,2,3,4,5,6,7},
95         {1,2,3,4,5,6,7},
96         {1,2,3,4,5,6,7},
97         {1,2,3,4,5,6,7},
98         {1,2,3,4,5,6,7},
99         {1,2,3,4,5,6,7}
100    };
101    int matrizresul[TAM][TAM];
102    int i,j=0;
103    for(i=0;i<TAM;i++){
104        for(j=0;j<TAM;j++){
105            matrizresul[i][j]=matriz1[i][j]-matriz2[i][j];
106        }
107    }
108    imprimirMatriz("Resta.txt",matrizresul);
109 }
110
111 DWORD WINAPI multiplicacionMatrices(LPVOID lpParam){
112     int matriz1[TAM][TAM]={
113         {7,6,5,4,3,2,1},
114         {7,6,5,4,3,2,1},
115         {7,6,5,4,3,2,1},
116         {7,6,5,4,3,2,1},
117         {7,6,5,4,3,2,1},
118         {7,6,5,4,3,2,1},
119         {7,6,5,4,3,2,1}
120    };
121    int matriz2[TAM][TAM]={
122        {1,2,3,4,5,6,7},
123        {1,2,3,4,5,6,7},
124        {1,2,3,4,5,6,7},
125        {1,2,3,4,5,6,7},
126        {1,2,3,4,5,6,7},
127        {1,2,3,4,5,6,7},
128        {1,2,3,4,5,6,7}
129    };
130    int matrizresul[TAM][TAM];
131    int i,j,k=0;
132    for(i=0;i<TAM;i++){
133        for(j=0;j<TAM;j++){
134            for(k=0;k<TAM;k++){
135                matrizresul[i][j]=matriz1[i][k]*matriz2[k][j];
136            }
137        }
138    }
139    imprimirMatriz("Multiplicacion.txt",matrizresul);
140 }
141
142 DWORD WINAPI transpuesta(LPVOID lpParam){
143     int matriz1[TAM][TAM]={
144         {7,6,5,4,3,2,1},
145         {7,6,5,4,3,2,1},
146         {7,6,5,4,3,2,1},
147         {7,6,5,4,3,2,1},
148         {7,6,5,4,3,2,1},
149         {7,6,5,4,3,2,1},
150         {7,6,5,4,3,2,1}
151    };
152    int matriz2[TAM][TAM]={
153        {1,2,3,4,5,6,7},
154        {1,2,3,4,5,6,7},
155        {1,2,3,4,5,6,7},
156        {1,2,3,4,5,6,7},
157        {1,2,3,4,5,6,7},
158        {1,2,3,4,5,6,7},
159        {1,2,3,4,5,6,7}
160    };
161    int matrizresul[TAM][TAM];
162    transponerMatriz(matriz1,matrizresul);
163    struct stat estatus;
164    int archivo_abierto=0;
165    char archivo[100]="Transpuesta.txt";
166    archivo_abierto=open(archivo,O_RDWR|O_CREAT);
167    imprimirMatriz(archivo_abierto,matrizresul);
168    transponerMatriz(matriz2,matrizresul);
169    imprimirMatrizT(archivo_abierto,matrizresul);
170    close(archivo_abierto);
171 }
172
173 DWORD WINAPI hiloImprimir(LPVOID lpParam){
174     imprimir("Suma.txt");
175     imprimir("Resta.txt");
176     imprimir("Multiplicacion.txt");
177     imprimir("Transpuesta.txt");
178 }
179
180 void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]){
181     int i,j=0;
182     for(i=0;i<TAM;i++){
183         for(j=0;j<TAM;j++){
184             matrizresul[i][j]=matriz[j][i];
185         }
186     }
187 }
188
189 void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]){
190     struct stat estatus;
191     int archivo_abierto=0;
192     char contenido[2];
193     archivo_abierto=open(archivo,O_RDWR|O_CREAT);
194     write(archivo_abierto,("\n", strlen("\n"));
195     //puts("");
196     int i,j=0;
197     for(i=0;i<TAM;i++){
198         for(j=0;j<TAM;j++){
199             printf("%i", matriz[i][j]);
200             write(archivo_abierto, contenido, strlen(contenido));
201             //printf("%i", matriz[i][j]);
202             if(i==9 && j==0){
203                 //printf("\n");
204                 write(archivo_abierto, "\n", strlen("\n"));
205             }
206         }
207         write(archivo_abierto, "\n", strlen("\n"));
208         close(archivo_abierto);
209     }
210 }
211
212 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         write(archivo_abierto, "\\n", strlen("\\n"));
228         //puts("\\n");
229     }
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

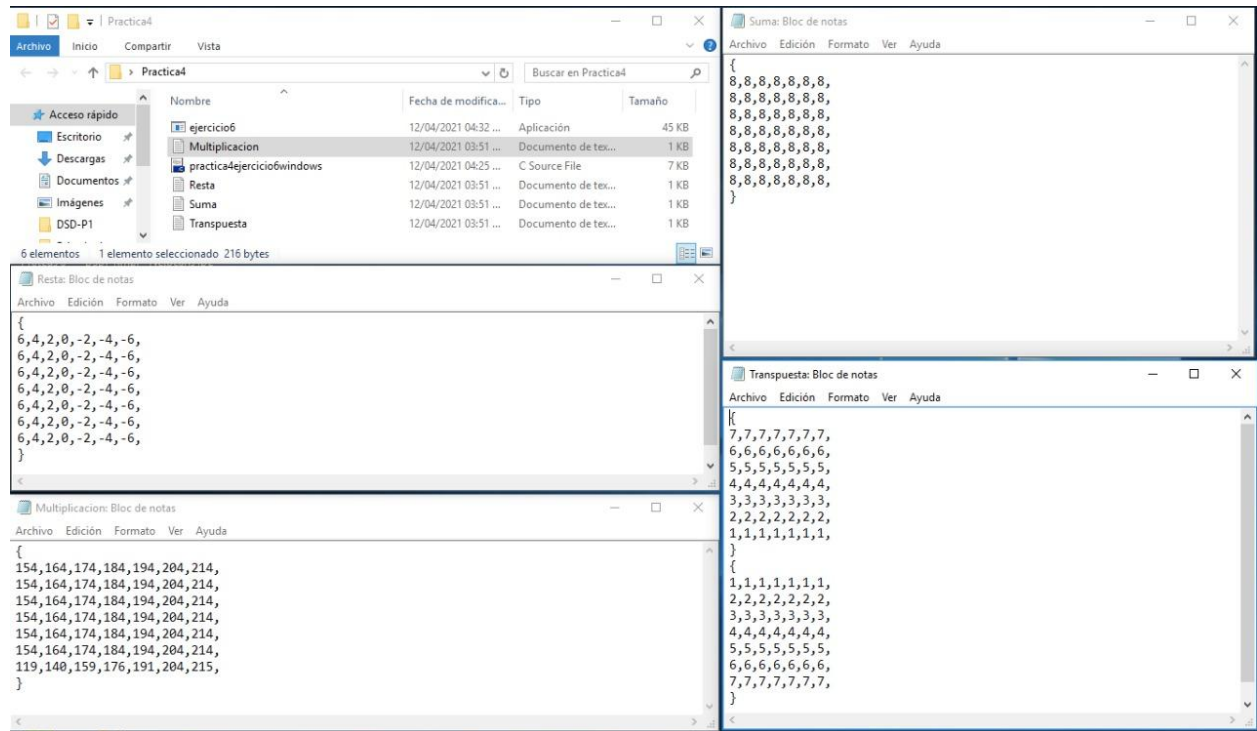
C:\Users\Luis Coto\Desktop\Practica4>gcc practica4ejercicio6windows.c -o ejercicio6

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,
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,
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,
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>

```


Archivos creados



Linux

```

1 #include<stdio.h>
2 #include<sys/stat.h>
3 #include<fcntl.h>
4 #include<unistd.h>
5 #include<time.h>
6 #include<stdlib.h>
7 #include<string.h>
8 #include<sys/wait.h>
9 #include<pthread.h>
10 #define TAM 7
11 void *sumaMatrices(void *arg);
12 void *restaMatrices(void *arg);
13 void *multiplicacionMatrices(void *arg);
14 void *transpuesta(void *arg);
15 void *hiloImprimir(void *arg);
16 void transponeMatrix(int matriz[TAM][TAM],int matrizresul[TAM][TAM]);
17 void imprimirMatrix(char archivo[100],int matriz[TAM][TAM]);
18 void imprimirMatrix(int archive_abierto,int matriz[TAM][TAM]);
19 void imprimirMatrix(char archivo[100]);
20 int main(int argc, char const *argv[])
21 {
22     clock_t t_ini, t_fin;
23     double secs;
24     t_ini=clock();
25     pthread_t id_hilo[5];
26     //Hilo para la suma
27     pthread_create(&id_hilo[0],NULL,sumaMatrices,NULL);
28     pthread_join(id_hilo[0],NULL);
29     //Hilo para la resta
30     pthread_create(&id_hilo[1],NULL,restaMatrices,NULL);
31     pthread_join(id_hilo[1],NULL);
32     //Hilo para la multiplicacion
33     pthread_create(&id_hilo[2],NULL,multiplicacionMatrices,NULL);
34     pthread_join(id_hilo[2],NULL);
35     //Hilo para la transpuesta
36     pthread_create(&id_hilo[3],NULL,transpuesta,NULL);
37     pthread_join(id_hilo[3],NULL);
38     //Hilo para la transpuesta
39     pthread_create(&id_hilo[4],NULL,hiloImprimir,NULL);
40     pthread_join(id_hilo[4],NULL);
41     t_fin = clock();
42     secs = (double)(t_fin - t_ini) / CLOCKS_PER_SEC;
43     printf("%.16g segundos\n", secs * 1000.0);
44     return 0;
45 }
46 void *sumaMatrices(void *arg){
47     int matriz1[TAM][TAM]={
48         {7,6,5,4,3,2,1},
49         {7,6,5,4,3,2,1},
50         {7,6,5,4,3,2,1},
51         {7,6,5,4,3,2,1},
52         {7,6,5,4,3,2,1},
53         {7,6,5,4,3,2,1},
54         {7,6,5,4,3,2,1}
55     };
56     int matriz2[TAM][TAM]={
57         {1,2,3,4,5,6,7},
58         {1,2,3,4,5,6,7},
59         {1,2,3,4,5,6,7},
60         {1,2,3,4,5,6,7},
61         {1,2,3,4,5,6,7},
62         {1,2,3,4,5,6,7},
63         {1,2,3,4,5,6,7}
64     };
65     int matrizresul[TAM][TAM];
66     int i,j=0;
67     for (i=0;i<TAM;i++){
68         for (j=0;j<TAM;j++){
69             matrizresul[i][j]=matriz1[i][j]+matriz2[i][j];
70         }
71     }
72     imprimirMatrix("Suma.txt",matrizresul);
73 }
74 void *restaMatrices(void *arg){
75     int matriz1[TAM][TAM]={
76         {7,6,5,4,3,2,1},
77         {7,6,5,4,3,2,1},
78         {7,6,5,4,3,2,1},
79         {7,6,5,4,3,2,1},
80         {7,6,5,4,3,2,1},
81         {7,6,5,4,3,2,1},
82         {7,6,5,4,3,2,1}
83     };
84     int matriz2[TAM][TAM]={
85         {1,2,3,4,5,6,7},
86         {1,2,3,4,5,6,7},
87         {1,2,3,4,5,6,7},
88         {1,2,3,4,5,6,7},
89         {1,2,3,4,5,6,7},
90         {1,2,3,4,5,6,7},
91         {1,2,3,4,5,6,7}
92     };
93     int matrizresul[TAM][TAM];
94     int i,j=0;
95     for (i=0;i<TAM;i++){
96         for (j=0;j<TAM;j++){
97             matrizresul[i][j]=matriz1[i][j]-matriz2[i][j];
98         }
99     }
100     imprimirMatrix("Resta.txt",matrizresul);
101 }
102 void *multiplicacionMatrices(void *arg){
103     int matriz1[TAM][TAM]={
104         {7,6,5,4,3,2,1},
105         {7,6,5,4,3,2,1},
106         {7,6,5,4,3,2,1},
107         {7,6,5,4,3,2,1},
108         {7,6,5,4,3,2,1},
109         {7,6,5,4,3,2,1},
110         {7,6,5,4,3,2,1}
111     };
112     int matriz2[TAM][TAM]={
113         {1,2,3,4,5,6,7},
114         {1,2,3,4,5,6,7},
115         {1,2,3,4,5,6,7},
116         {1,2,3,4,5,6,7},
117         {1,2,3,4,5,6,7},
118         {1,2,3,4,5,6,7},
119         {1,2,3,4,5,6,7}
120     };
111     int matrizresul[TAM][TAM];
112     int i,j=0;
113     for (i=0;i<TAM;i++){
114         for (j=0;j<TAM;j++){
115             matrizresul[i][j]=0;
116             for (k=0;k<TAM;k++){
117                 matrizresul[i][j]+=matriz1[i][k]*matriz2[k][j];
118             }
119         }
120     }
121     imprimirMatrix("Multiplicacion.txt",matrizresul);
122 }
123 void *transpuesta(void *arg){
124     int matriz[TAM][TAM]={
125         {7,6,5,4,3,2,1},
126         {7,6,5,4,3,2,1},
127         {7,6,5,4,3,2,1},
128         {7,6,5,4,3,2,1},
129         {7,6,5,4,3,2,1},
130         {7,6,5,4,3,2,1},
131         {7,6,5,4,3,2,1}
132     };
133     int matrizresul[TAM][TAM];
134     int i,j=0;
135     for (i=0;i<TAM;i++){
136         for (j=0;j<TAM;j++){
137             matrizresul[i][j]=matriz[j][i];
138         }
139     }
140     imprimirMatrix("Transpuesta.txt",matrizresul);
141 }
142 void *hiloImprimir(void *arg){
143     char archivo[100]="";
144     int fd=open(archivo,O_CREAT|O_WRONLY|O_TRUNC,0666);
145     if (fd<0){
146         printf("Error al crear el archivo\n");
147         return;
148     }
149     int matriz[TAM][TAM]={
150         {7,6,5,4,3,2,1},
151         {7,6,5,4,3,2,1},
152         {7,6,5,4,3,2,1},
153         {7,6,5,4,3,2,1},
154         {7,6,5,4,3,2,1},
155         {7,6,5,4,3,2,1},
156         {7,6,5,4,3,2,1}
157     };
158     int i,j=0;
159     for (i=0;i<TAM;i++){
160         for (j=0;j<TAM;j++){
161             fprintf(fd,"%d\t",matriz[i][j]);
162             if (j==TAM-1){
163                 fprintf(fd,"\n");
164             }
165         }
166     }
167     close(fd);
168 }

```

```

91     {1,2,3,4,5,6,7}
92 };
93 int matrizresul[TAM][TAM];
94 int i,j=0;
95 for(i=0;i<TAM;i++){
96     for(j=0;j<TAM;j++){
97         matrizresul[i][j]=matriz1[i][j]-matriz2[i][j];
98     }
99 }
100 imprimirMatriz("Resta.txt",matrizresul);
101 }
102 void *multiplicacionMatrices(void *arg){
103     int matriz1[TAM][TAM]={
104         {7,6,5,4,3,2,1},
105         {7,6,5,4,3,2,1},
106         {7,6,5,4,3,2,1},
107         {7,6,5,4,3,2,1},
108         {7,6,5,4,3,2,1},
109         {7,6,5,4,3,2,1},
110         {7,6,5,4,3,2,1}
111     };
112     int matriz2[TAM][TAM]={
113         {1,2,3,4,5,6,7},
114         {1,2,3,4,5,6,7},
115         {1,2,3,4,5,6,7},
116         {1,2,3,4,5,6,7},
117         {1,2,3,4,5,6,7},
118         {1,2,3,4,5,6,7},
119         {1,2,3,4,5,6,7}
120     };
121     int matrizresul[TAM][TAM];
122     int i,j,k=0;
123     for(i=0;i<TAM;i++){
124         for(j=0;j<TAM;j++){
125             //Inicia array en cero
126             matrizresul[i][j]=0;
127             for(k=0;k<10;k++){
128                 matrizresul[i][j]+=matriz1[i][k]*matriz2[k][j];
129             }
130         }
131     }
132     imprimirMatriz("Multiplicacion.txt",matrizresul);
133 }
134 void *transpuesta(void *arg){
135     int matriz1[TAM][TAM]={
136         {7,6,5,4,3,2,1},
137         {7,6,5,4,3,2,1},
138         {7,6,5,4,3,2,1},
139         {7,6,5,4,3,2,1},
140         {7,6,5,4,3,2,1},
141         {7,6,5,4,3,2,1},
142         {7,6,5,4,3,2,1}
143     };
144     int matriz2[TAM][TAM]={
145         {1,2,3,4,5,6,7},
146         {1,2,3,4,5,6,7},
147         {1,2,3,4,5,6,7},
148         {1,2,3,4,5,6,7},
149         {1,2,3,4,5,6,7},
150         {1,2,3,4,5,6,7},
151         {1,2,3,4,5,6,7}
152     };
153     int matrizresul[TAM][TAM];
154     transponerMatriz(matriz1,matrizresul);
155     struct stat estatus;
156     int archivo_abierto=0;
157     char archivo[100]="Transpuesta.txt";
158     archivo_abierto=open(archivo,O_RDWR|O_CREAT);
159     imprimirMatrizT(archivo_abierto,matrizresul);
160     transponerMatriz(matriz2,matrizresul);
161     imprimirMatrizT(archivo_abierto,matrizresul);
162     close(archivo_abierto);
163 }
164 void *hiloImprimir(void *arg){
165     imprimir("Suma.txt");
166     imprimir("Resta.txt");
167     imprimir("Multiplicacion.txt");
168     imprimir("Transpuesta.txt");
169 }
170
171 void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]){
172     int i,j=0;
173     for(i=0;i<TAM;i++){
174         for(j=0;j<TAM;j++){
175             matrizresul[i][j]=matriz[j][i];
176         }
177     }
178 }
179 void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]){
180     struct stat estatus;
181     int archivo_abierto=0;
182     char contenido[2];

```

```

182     char contenido[2];
183     archivo_abierto=open(archivo,O_RDWR|O_CREAT);
184     write(archivo_abierto, "\n", strlen("\n"));
185     //puts("");
186     int i,j=0;
187     for(i=0;i<TAM;i++){
188         for(j=0;j<TAM;j++){
189             sprintf(contenido, "%i", matriz[i][j]);
190             write(archivo_abierto, contenido, strlen(contenido));
191             //printf("%i", matriz[i][j]);
192             if(i==9 && j==9){
193                 }else{
194                     //printf(",");
195                     write(archivo_abierto, ",", strlen(","));
196                 }
197             write(archivo_abierto, "\n", strlen("\n"));
198             //puts("");
199         }
200     }
201     //puts("");
202     write(archivo_abierto, "\n", strlen("\n"));
203     close(archivo_abierto);
204 }
205 void imprimirMatrizT(int archivo_abierto,int matriz[TAM][TAM]){
206     //printf("%i\n", archivo_abierto);
207     char contenido[2];
208     write(archivo_abierto, "\n", strlen("\n"));
209     //puts("");
210     int i,j=0;
211     for(i=0;i<TAM;i++){
212         for(j=0;j<TAM;j++){
213             sprintf(contenido, "%i", matriz[i][j]);
214             write(archivo_abierto, contenido, strlen(contenido));
215             //printf("%i", matriz[i][j]);
216             if(i==9 && j==9){
217                 }else{
218                     //printf(",");
219                     write(archivo_abierto, ",", strlen(","));
220                 }
221             write(archivo_abierto, "\n", strlen("\n"));
222             //puts("");
223         }
224     }
225     //puts("");
226     write(archivo_abierto, "\n", strlen("\n"));
227 }
228 void imprimir(char archivo[100]){
229     struct stat estatus;
230     int archivo_abierto=0;
231     archivo_abierto=open(archivo,O_RDONLY);
232     stat(archivo,&estatus);
233     char contenido[estatus.st_size];
234     read(archivo_abierto, contenido, estatus.st_size);
235     int i=0;
236     printf("Contenido de %s:\n", archivo);
237     while ((*contenido + i) != '\0' && i+1 <= estatus.st_size) {
238         if((*contenido + i) > 0){
239             printf("%c", *contenido+i);
240             if((*contenido+i)==' '){
241                 puts("");
242                 break;
243             }
244         }
245         i++;
246     }
247     close(archivo_abierto);
248 }

```

```

137     {7,6,5,4,3,2,1},
138     {7,6,5,4,3,2,1},
139     {7,6,5,4,3,2,1},
140     {7,6,5,4,3,2,1},
141     {7,6,5,4,3,2,1},
142     {7,6,5,4,3,2,1},
143     {7,6,5,4,3,2,1}
144 };
145 int matriz2[TAM][TAM]={
146     {1,2,3,4,5,6,7},
147     {1,2,3,4,5,6,7},
148     {1,2,3,4,5,6,7},
149     {1,2,3,4,5,6,7},
150     {1,2,3,4,5,6,7},
151     {1,2,3,4,5,6,7},
152     {1,2,3,4,5,6,7}
153 };
154 int matrizresul[TAM][TAM];
155 transponerMatriz(matriz1,matrizresul);
156 struct stat estatus;
157 int archivo_abierto=0;
158 char archivo[100]="Transpuesta.txt";
159 archivo_abierto=open(archivo,O_RDWR|O_CREAT);
160 imprimirMatrizT(archivo_abierto,matrizresul);
161 transponerMatriz(matriz2,matrizresul);
162 imprimirMatrizT(archivo_abierto,matrizresul);
163 close(archivo_abierto);
164 }
165 void *hiloImprimir(void *arg){
166     imprimir("Suma.txt");
167     imprimir("Resta.txt");
168     imprimir("Multiplicacion.txt");
169     imprimir("Transpuesta.txt");
170 }
171
172 void transponerMatriz(int matriz[TAM][TAM],int matrizresul[TAM][TAM]){
173     int i,j=0;
174     for(i=0;i<TAM;i++){
175         for(j=0;j<TAM;j++){
176             matrizresul[i][j]=matriz[j][i];
177         }
178     }
179 }
180 void imprimirMatriz(char archivo[100],int matriz[TAM][TAM]){
181     struct stat estatus;
182     int archivo_abierto=0;
183     char contenido[2];

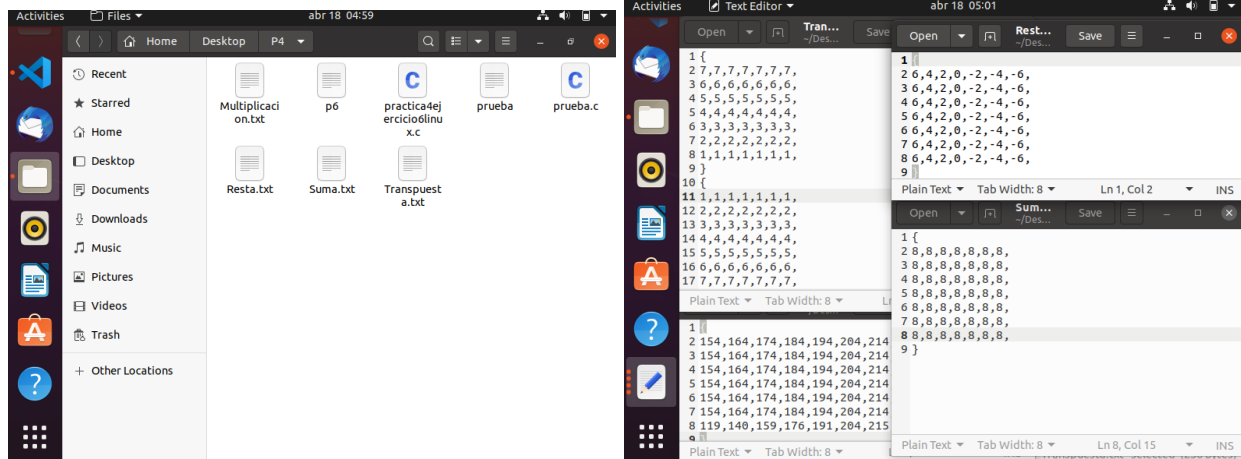
```

```

228 void imprimir(char archivo[100]){
229     struct stat estatus;
230     int archivo_abierto=0;
231     archivo_abierto=open(archivo,O_RDONLY);
232     stat(archivo,&estatus);
233     char contenido[estatus.st_size];
234     read(archivo_abierto, contenido, estatus.st_size);
235     int i=0;
236     printf("Contenido de %s:\n", archivo);
237     while ((*contenido + i) != '\0' && i+1 <= estatus.st_size) {
238         if((*contenido + i) > 0){
239             printf("%c", *contenido+i);
240             if((*contenido+i)==' '){
241                 puts("");
242                 break;
243             }
244         }
245         i++;
246     }
247     close(archivo_abierto);
248 }

```


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 creí 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 lugar a 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 encuentre 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