Instituto Politécnico Nacional

Escuela superior de Cómputo

Práctica Seis

Threads

Alumno:

Rojas Zepeda Luis Eduardo

Profesor:
Velez Saldaña Ulises

2CM6, 09/05/19

Índice

1. Teoría

Threads Un thread es un hilo de ejecucion paralelo a nuestro programa. Si queremos que un UNICO programa realice varias tareas a la vez, de forma paralela y no secuencialmente hemos de hacer uso de Threads.

Diferencias entre threads e hilos:

1. A diferencia del fork, el cual padre tiene un PID e hijo tiene otro diferente por tanto son dos procesos diferentes, en un programa con Threads el PID de todos los threads es el mismo (se diferencian por un Thread ID interno).
2. Los threads, al ser un mismo proceso, comparten memoria con el padre, por lo que no hace falta realizar el uso de pipes para enviar informacion de unos a otros.
3. Al compartir memoria, hay que tener mas cuidado al cambiar el valor de las variables (si son compartidas) porque puede darse el caso de que dos threads vayan a cambiar el valor a la vez, haciendo que desconozcamos el valor que puede tomar la variable. (Esto se evitará mas adelante con los semáforos)
4. Programar con threads consume (por lo normal) menos memoria ram que hacerlo con forks, ya que el fork hacia una copia de todas las variables para el hijo. Recordemos que los threads comparten memoria.
5. Cuando el proceso padre muere, los threads también lo hacen.

Para compilar un programa con uso de threads se hace de la siguiente forma:

```
gcc sesion.c -Wall -Wextra -o s5.exe -lpthread
```

Y se usa la siguiente libreria:

```
#include <pthread.h>
```

Dentro de nuestro código, debemos crear una fonción que se ejecutará especialmente para un hilo, debe tener la siguiente forma:

```
void *funcionThread (void *arg) {

//Casteamos lo que nos pasan por argumento... puede ser char,
    struct, etc.
char *tmpStr = (char *) arg;

// Codigo de la funcion


pthread_exit(NULL);
}
```

El hilo se crea de la siguiente forma

```
stat = pthread_create(&thLecturaplato, NULL, thLectura, &fileName )
    ;
if ( 0 != stat ) {
```

```
// Error al crear el thread, salimos del padre lanzando signal o
    exit
exit(-1);
}
```

donde el primer parámetro es un hilo previamente declarado (pthreadt thread), posteriormente se colocan las propiedades pero para nuestro caso lo dejaremos en NULL, el tercer parámetro es el método a ejecutar y el último parámetro es alguna variable que deseamos enviar.

Finalmente esperamos a que termine el hilo para poder continuar con la siguiente linea:

```
stat = pthread_join(thLecturaplato, NULL);
if ( 0 != stat ) {
// Error al cerrar el thread, salimos del padre lanzando signal o
    exit
exit(-1);
}
```

Makefile Make es una herramienta de gestión de dependencias, típicamente, las que existen entre los archivos que componen el código fuente de un programa, para dirigir su recompilación o "generación.ªutomáticamente. Si bien es cierto que su función básica consiste en determinar automáticamente qué partes de un programa requieren ser recompiladas y ejecutar los comandos necesarios para hacerlo, también lo es que Make puede usarse en cualquier escenario en el que se requiera, de alguna forma, actualizar automáticamente un conjunto de archivos a partir de otro, cada vez que éste cambie

La estructura básica es la siguiente:

```
objetivo: dependencias
        comandos
```

En "objetivo" definimos el módulo o programa que queremos crear, después de los dos puntos y en la misma línea podemos definir qué otros módulos o programas son necesarios para conseguir el "objetivo". Por último, en la línea siguiente y sucesivas indicamos los comandos necesarios para llevar esto a cabo. Es muy importante que los comandos estén separados por un tabulador del comienzo de línea.

2. Material

Editor de Texto Nano
Solo se instala con la siguiente linea:

```
sudo apt-get install nano
```

Copilador gcc
Se instala con la siguiente linea:

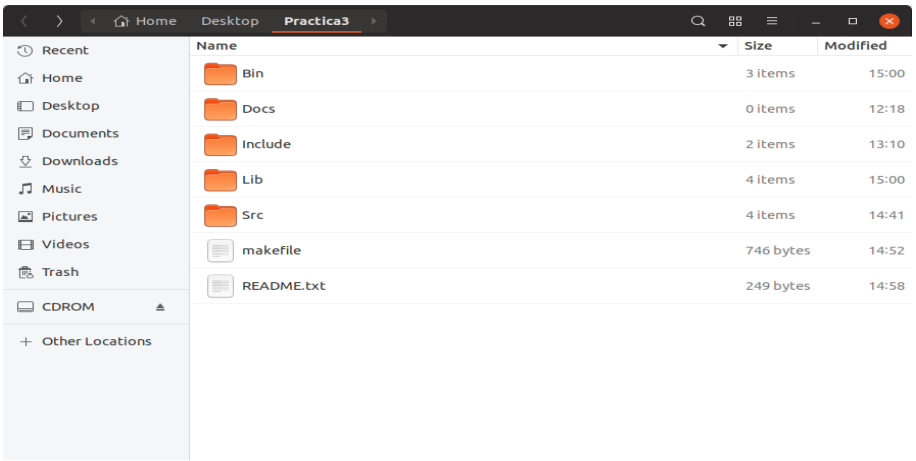
```
sudo apt-get install gcc
```

Makefile
Se instala con la siguiente linea:

```
sudo make install
```

3. Desarrollo

Crearemos las siguientes carpetas:



Después se debe generar el siguiente código:

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

struct  filas
{
        int* filaA;
        int* filaB;
        int columnas;
};

char* getDimension(char name[30]){
        FILE *fp;
        char file[80];
        char pal[80];
        char *str;
        int filas;

        strcpy(file,name);
        strcat(file, ".txt");
        fp = fopen ( file, "r" );
        if (fp==NULL) {fputs ("File error",stderr); exit
            (1);}

        if(fscanf(fp, "%s", pal)!=EOF){
                str = (char *) malloc(sizeof(char) * sizeof
                    (pal));
                strcpy(str, pal);
        }
```

```
        return str;
}

int** leerArchivos(char name[30]){
        FILE *fp;
        char file[80];
        char pal[80];
        int i = 0, j = 0, k = 0, l=0, filas, columnas;
        int **matriz;

        strcpy(file,name);
        strcat(file, ".txt");
        fp = fopen ( file, "r" );
        if (fp==NULL) {fputs ("File error",stderr); exit
            (1);}

        while(fscanf(fp, "%s ", pal)!=EOF){
                if(j==0){
                        filas = pal[0] - '0';
                        columnas = pal[2] -'0';
                        matriz = (int **)malloc(filas*
                            sizeof(int));
                        for(i=0; i<filas; i++)
                        matriz[i] = (int *)malloc(columnas*
                            sizeof(int));

                }else{
                        for(k=0; k<columnas; k++){
                                matriz[j-1][k] = pal[l] - '
                                    0';
                                        l=l+2;
                                }
                        }
                        l=0;
                        j++;
                }

        printf("**** Matriz %s ****\n", file);
                for (i = 0; i < filas; ++i){
                        for (j = 0; j < columnas; ++j)
                                printf("%d\t", matriz[i][j
                                    ]);
                        printf("\n");
                }
        printf("\n\n");

        fclose (fp);
        return matriz;
}

void *thread_routine(struct filas *f){
        int *filaSum = (int*)malloc(sizeof(int) * f->
            columnas), i;
        for(i = 0; i<f->columnas; i++)
        filaSum[i] = f->filaA[i] + f->filaB[i];

        pthread_exit(filaSum);
}

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

```
        int filas = 0, columnas, i, j, status;
        int **mA = leerArchivos(argv[1]);
        int **mB = leerArchivos(argv[2]);
        int **mS;
        char *dimA = getDimension(argv[1]), *dimB =
            getDimension(argv[2]);

        if(argc < 2){
                printf("Argumentos incompletos \n");
                return -1;
        }
                if(*dimA!=*dimB){
                printf("dimensiones diferentes de las
                    matrices\n");
                return -1;
        }

        filas = dimA[0] - '0';
        columnas = dimA[2] - '0';
        pthread_t thread[filas];
        mS = (int **)malloc(filas*sizeof(int));
        for(i=0; i<filas; i++)
        mS[i] = (int *)malloc(columnas*sizeof(int));

        for(i = 0; i < filas; i++){
                printf("creando hilo para fila %d\n", i);
                struct filas f = {mA[i], mB[i], columnas};
                int *filaSum = (int*)malloc(sizeof(int) * f
                    .columnas);
                status = pthread_create(&thread[i], NULL, (
                    void*)thread_routine, (void*)&f);
                pthread_join(thread[i], (void **) &filaSum)
                    ;
                for(j = 0; j<columnas; j++)
                mS[i][j] = filaSum[j];

        }

        printf("\n");
        for (i = 0; i < filas; ++i){
        for (j = 0; j < columnas; ++j)
        printf("%d\t", mS[i][j]);
        printf("\n");
        }

        return 0;
}
```

Archivo Makefile

```
compilar: Src/programa.c
	$(CC) Src/programa.c -o Src/progra

correr:
	Src/progra

```

Y finalmente crearemos el archivo README.txt con el siguiente texto:

Archivo README

```
COMPILACION DEL PROGRAMA:
    make compilar

EJECUCION:
    make correr

```

Este archivo servirá como manual para el usuario para correr los programas de manera más fácil. Este archivo, junto con el makefile, irán en la raíz de la carpeta como se muestra en la primera imagen de esta sección de desarrollo.

Necesitaremos dos archivos de texto (.txt) con una matriz cada uno. El primera linea irán las dimensiones de esta y seguido estará la matriz de la siguiente forma:

Archivo file2.txt

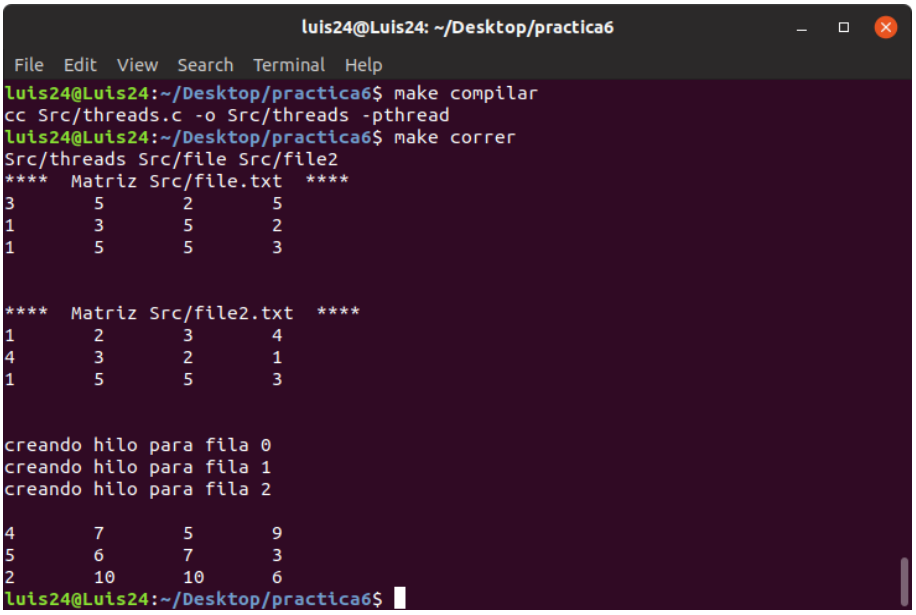
```
3x4
1,2,3,4
4,3,2,1
1,5,5,3
```

Archivo file2.txt

```
3x4
1,2,3,4
4,3,2,1
1,5,5,3
```

Finalmente para compilar el programa y correrlo se hará de la siguiente forma:

Y se verá de la siguiente forma la ejecución:


luis24@Luis24: ~/Desktop/practica6
File Edit View Search Terminal Help
luis24@Luis24:~/Desktop/practica6$ make compilar
cc Src/threads.c -o Src/threads -pthread
luis24@Luis24:~/Desktop/practica6$ make correr
Src/threads Src/file Src/file2
**** Matriz Src/file.txt ****
3 5 2 5
1 3 5 2
1 5 5 3
**** Matriz Src/file2.txt ****
1 2 3 4
4 3 2 1
1 5 5 3
creando hilo para fila 0
creando hilo para fila 1
creando hilo para fila 2
4 7 5 9
5 6 7 3
2 10 10 6
luis24@Luis24:~/Desktop/practica6$

4. Bibliografía

Referencias

[1] https://www.driverlandia.com/programacion-c-avanzada-t3-uso-de-threads-en-c/

[2] https://www.geeksforgeeks.org/multithreading-c-2/