Concurso 4 - Threads

Leonardo Moreira 71512

1.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

char information;
int InfoCheck = 0;

void setInformation(char set){  //sets the characters
        while(InfoCheck);
        information = set;
        InfoCheck = 1;
}

char readInformation(void){  //it will get all the chars and compare them with 'E'
        while(!InfoCheck);
        char out = information;
        InfoCheck = 0;
        return out;
}


void *Thread_Read(void *arg){ //it will be responsible for reading all the characters
        char in;
        printf("\n Thread 1: ");
        while(scanf("%c", &in) != EOF && in != 'E'){
                setInformation(in);
        }
        printf("\nReadThread Stopped\n");
        setInformation('E');
        return NULL;
}

void *Thread_Write(void *arg){  //write all chars
        char out = '\0';
        while((out = readInformation()) != 'E' ){
                printf("Thread 2: %c\n", out);
        }
        printf("WriteThread Stopped\n");
        return NULL;
}
```

```
File  Actions  Edit  View  Help

ReadThread Stopped
WriteThread Stopped

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex1

 Thread 1: ghjhglo gh 12
Thread 2: g
Thread 2: h
Thread 2: j
Thread 2: h
Thread 2: g
Thread 2: l
Thread 2: o
Thread 2:
Thread 2: g
Thread 2: h
Thread 2:
Thread 2: 1
Thread 2: 2
Thread 2:

E

ReadThread Stopped
WriteThread Stopped

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$
```

```c
}

int main(int argc, char const *argv[])
{
        pthread_t ThreadRead, ThreadWrite;

        pthread_create(&ThreadRead, NULL, Thread_Read, NULL);
        pthread_create(&ThreadWrite, NULL, Thread_Write, NULL);

        pthread_join(ThreadRead, NULL);
        pthread_join(ThreadWrite, NULL);

        return 0;
}
```

```
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex1

 Thread 1: ghjhglo gh 12
Thread 2: g
Thread 2: h
Thread 2: j
Thread 2: h
Thread 2: g
Thread 2: l
Thread 2: o
Thread 2:
Thread 2: g
Thread 2: h
Thread 2:
Thread 2: 1
Thread 2: 2
Thread 2:

E

ReadThread Stopped
WriteThread Stopped

┌──(kali㉿kali)-[~/Desktop/lab4]
```

2.

a)

Neste exercício temos que ter em conta a maneira como compilamos o programa, visto que este erro acontece geralmente quando compilamos um programa em C com G++ ou então GCC.

```
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ gcc ex2.c -o ex2c
/usr/bin/ld: /tmp/cc4mdXGS.o: in function `main':
ex2.c:(.text+0×b9): undefined reference to `pthread_create'
/usr/bin/ld: ex2.c:(.text+0×101): undefined reference to `pthread_join'
/usr/bin/ld: ex2.c:(.text+0×115): undefined reference to `pthread_join'
collect2: error: ld returned 1 exit status

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$
```

Por norma, as bibliotecas devem seguir objetos e fontes na linha de comandos, e “lpthread/pthread” não é uma “opção” é sim, uma especificação de biblioteca.
Por isso, temos que colocar -pthread depois do comando de compilação para “dizer” ao compilador para executar o programa com a biblioteca pthread.h.

```c
// ######ex2.c##########
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

pthread_t tid[2]; //creating 2 threads
int counter;

void* trythis(void* arg)
{
 unsigned long i = 0;
 counter += 1;
 printf("\n Job %d has started\n", counter);

 for (i = 0; i < (0xFFFFFFFF); i++) ;
 printf("\n Job %d has finished\n", counter);
 return NULL;
}

int main(void)
{
 int i = 0;
 int error;


 while (i < 2) {
 error = pthread_create(&(tid[i]), NULL, &trythis, NULL); //função que cria a thread
 if (error ≠ 0) //vou verificar se foi criada com sucesso
 printf("\nThread can't be created :[%s]", strerror(error));
 i++;
 }

 pthread_join(tid[0], NULL); //this function is basically the wait function, but for threads, it waits for the thread 1 to finish to execute the second
 pthread_join(tid[1], NULL);
 return 0;
}
```

```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ gcc -o ex2c ex2c.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex2c

 Job 1 has started

 Job 1 has finished

 Job 2 has started

 Job 2 has finished

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ gcc -o ex2 ex2.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex2

 Job 1 has started

 Job 2 has started

 Job 2 has finished

 Job 2 has finished

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$
```

Nota: É sempre melhor usar -pthread em vez de -lpthread, pois -lpthread apenas adiciona a biblioteca pthread e não definirá as macros predefinidas.

b)
Neste caso, penso que a ideia do programa seria ter uma thread a ser executada de cada vez, ou seja, um processo só seria executado quando o anterior acabasse. Isto é, o job 1 seria iniciado e o job 2 só seria iniciado quando o 1 acabasse, daí o objetivo seria obter um output deste tipo:

```
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex2c

Job 1 has started

Job 1 has finished

Job 2 has started

Job 2 has finished
```

O problema é que temos várias threads a serem executadas ao mesmo tempo, visto que quando executamos o programa, reparamos que ao mesmo tempo, o job 1 e o job 2 foram iniciados e só o job 2 é terminado, 2 vezes, o que não era suposto.

```
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex2

Job 1 has started

Job 2 has started

Job 2 has finished

Job 2 has finished
```

Isto acontece pelo facto das threads partilharem memória, por isso, quando o valor do counter foi incrementado o valor manteve-se igual nas 2 threads visto que estavam as 2 a serem executadas ao mesmo tempo.
Por este motivo, é importante usar mutex, pois permite que um processo seja executado e os restantes aguardem que esse processo termine para serem executados de seguida.

c)

ex2.c

```c
// ######ex2.c##########
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

//exercício 2, alinea c)S

pthread_t tid[2];  //2 threads
int counter;

pthread_mutex_t mutex;

void* trythis_1(void* arg)
{
  if (0 != (errno = pthread_mutex_lock(&mutex)))
  {
    perror("pthread_mutex_lock failed");
    exit(EXIT_FAILURE);
  }

  unsigned long i = 0;
  counter += 1;
  printf("\n Job %d has started\n", counter);
  for (i = 0; i < (0xFFFFFFFF); i++) ;
  printf("\n Job %d has finished\n", counter);
  sleep(1);

  if (0 != (errno = pthread_mutex_unlock(&mutex)))
  {
    perror("pthread_mutex_unlock failed");
    exit(EXIT_FAILURE);
  }
  return NULL;
}

int main(void)
{
  int i = 0;
  int error;
  while (i < 2)
  {
    error = pthread_create(&(tid[i]), NULL, &trythis_1, NULL);
    if (error != 0)
      printf("\nThread can't be created :[%s]", strerror(error));
    i++;
  }
  pthread_join(tid[0], NULL);
  pthread_join(tid[1], NULL);
  return 0;
}
```

File Actions Edit View Help

```
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ gcc -o ex2c ex2c.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex2c

 Job 1 has started

 Job 1 has finished

 Job 2 has started

 Job 2 has finished

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$
```

```
┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ gcc -o ex2c ex2c.c -pthread

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$ ./ex2c

 Job 1 has started

 Job 1 has finished

 Job 2 has started

 Job 2 has finished

┌──(kali㉿kali)-[~/Desktop/lab4]
└─$
```

Bibliografia:

https://stackoverflow.com/questions/1662909/undefined-reference-to-pthread-create-in-linux
https://myeasytuts.com/solved-undefined-reference-to-pthread_create-linux/
https://tutoria.ualg.pt/2021/pluginfile.php/236191/mod_resource/content/1/Threads.pdf
https://www.youtube.com/watch?v=d9s_d28yJq0
https://www.youtube.com/watch?v=HDohXvS6UIk
https://www.youtube.com/watch?v=xoXzp4B8aQk
https://www.youtube.com/watch?v=ln3el6PR__Q
https://www.youtube.com/watch?v=oq29KUy29iQ
https://www.youtube.com/watch?v=FY9livorrJI&list=PLfqABt5AS4FmuQf70psXrsMLEDQXNkLq2&index=3