RIPHAH INTERNATIONAL UNIVERSITY, ISLAMABAD

Lab#14

Bachelors of Computer Science – 5th Semester

Subject: Operating System

Submitted to: Ms. Kausar Nasreen Khattak

Submitted by: Tabinda Hassan

SAP-46374

Date of Submission: 26-11-2024

Exercise 1:

Type the following and execute it.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //Header file for sleep(). man 3 sleep for details.
#include <pthread.h>

// A normal C function that is executed as a thread
// when its name is specified in pthread_create()
void *myThreadFun(void *vargp)
{
    sleep(1);
    printf("Printing GeeksQuiz from Thread \n");
    return NULL;
}

int main()
{
    pthread_t thread_id;
    printf("Before Thread\n");
    pthread_create(&thread_id, NULL, myThreadFun, NULL);
    pthread_join(thread_id, NULL);
    printf("After Thread\n");
    exit(0);
}
```

Explanation:

This program demonstrates how to create and manage threads in C using **pthreads**.

• **Thread Creation:** The program defines a function `myThreadFun()` that will be run by the thread.

• **Thread Execution:** The thread sleeps for 1 second (`sleep(1)`), then prints the message: "Printing GeeksQuiz from Thread".

• **Synchronization:** `pthread_join(thread_id, NULL)` ensures the main program waits for the thread to finish before continuing. Without this, the main program might finish and exit before the thread has a chance to print its message.

• **Output:** The program first prints "Before Thread", then the thread prints "Printing GeeksQuiz from Thread", and finally, the main program prints "After Thread"

Output:

```
Before Thread
Printing GeeksQuiz from Thread
After Thread
```

Exercise # 2:

Try to execute following code:

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

// Let us create a global variable to change it in threads
int g = 0;

// The function to be executed by all threads
void *myThreadFun(void *vargp)
{
    // Store the value argument passed to this thread
    int *myid = (int *)vargp;

    // Let us create a static variable to observe its changes
    static int s = 0;

    // Change static and global variables
    ++s; ++g;

    // Print the argument, static and global variables
    printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}

int main()
{
    int i;
    pthread_t tid;

    // Let us create three threads
    for (i = 0; i < 3; i++)
        pthread_create(&tid, NULL, myThreadFun, (void *)&tid);

    pthread_exit(NULL);
    return 0;
}
```

Show output with explanation

Explanation:

This program demonstrates the use of global and static variables in multithreading:

1. **Global variable `g`** is shared by all threads and modified by each one.
2. **Static variable `s`** is unique to each thread but persists across calls within the same thread.
3. The `main()` function creates three threads, and each thread executes `myThreadFun()`, which prints the thread ID, static, and global variable values.
4. `pthread_exit()` ensures the main thread waits for all threads to complete before exiting.

Note: The program has a potential issue with passing the same `tid` to all threads, which can cause race conditions.

Output:

```
Thread ID: 140129406736576, Static: 1, Global: 1
Thread ID: 140129406736576, Static: 2, Global: 2
Thread ID: 140129406736576, Static: 3, Global: 3
```

Exercise # 3:

Try to execute following code:

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

void * workerThreadFunc(void * tid){
  long * myID = (long *) tid;
  printf("HELLO WORLD! THIS IS THREAD %ld\n",*myID);
}

int main(){

  pthread_t tid0;
  pthread_create(&tid0,NULL,workerThreadFunc,(void *)&tid0);

  pthread_exit(NULL);
  return 0;
}
```

Modify pthread_create and convert into for loop and create three threads and show output. Also removes pthread_exit and see what happens.

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


void *workerThreadFunc(void *tid) {
   long *myID = (long *)tid;
   printf("HELLO WORLD! THIS IS THREAD %ld\n", *myID);
   return NULL;
}


int main() {
   pthread_t tids[3];  // Array to hold thread IDs
   for (long i = 0; i < 3; i++) {
      pthread_create(&tids[i], NULL, workerThreadFunc, (void *)&i);
   }
   // No pthread_exit, the program will terminate once main finishes
   return 0;
}
```

Explanation:

- This code creates three threads using a for loop.
- Each thread executes the workerThreadFunc function, which prints a message with the thread's ID.
- The main thread creates the threads and then exits, but without pthread_exit(), the main thread may finish before the others, causing unpredictable behavior.
- The value of i (used as the thread ID) is shared among all threads, potentially leading to race conditions.

Without `pthread_exit()`, the program may terminate before the threads finish executing. The output could be unpredictable due to race conditions with the loop variable `i`. If the threads do get executed before the main function exits, the output may be:

Output:

```
HELLO WORLD! THIS IS THREAD 2
HELLO WORLD! THIS IS THREAD 2
HELLO WORLD! THIS IS THREAD 2
```

Or, if the threads execute in the expected order:

Output:

```
HELLO WORLD! THIS IS THREAD 0
HELLO WORLD! THIS IS THREAD 1
HELLO WORLD! THIS IS THREAD 2
```

Exercise 4:

Define posix thread and its working in your own words. 03

- A **POSIX thread** (pthread) is a standardized method for managing threads in Unix-like operating systems.
- It allows concurrent execution of multiple tasks within the same process, sharing resources like memory.
- Threads are created using `pthread_create()`, and synchronization is achieved using mechanisms like **mutexes**. Threads can be joined with `pthread_join()` to ensure they complete before the program exits. POSIX threads enable efficient multitasking and parallel processing in applications.