Operating System

Lab14 Tasks

Name: Adeesha Sherani

SapID:47379

Batch: BSCS-5thsem

SUBMITTED TO: Kausar Nasreen Khattak

Task1:

Typethefollowingandexecuteit.

Solution:

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

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);
}
```

```
"task1.c" [New] 22L, 404B written
[root@localhost ~]# gcc task1.c -o task1 -pthread
[root@localhost ~]# ./task1
Before Thread
Printing GeeksQuiz from Thread
After Thread
[root@localhost ~]#
```

CodeExplanation:

pthread_create:

This function creates a new thread that runs the function myThreadFun. The thread willsleepfor1secondand then print"PrintingGeeksQuizfromThread".

pthread_join:

Thiswaitsforthenewlycreated threadtofinish itsexecutionbeforemovingonwiththemainfunction.

Output:

The program first prints "Before Thread", then the thread runs and prints "PrintingGeeksQuizfromThread",andfinally,themainfunctionprints"After Thread".

Task2:

Trytoexecutefollowingcode:

Solution:

```
void *myThreadFun(void *vargp)
{
    int myid = (int *)vargp;  // Store the value argument passed to this thread
    static int s = 0;         // Static variable to observe its changes

    // 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);
    return NULL;
}

int main()
{
    int i;
    pthread_t tid;

    // Create three threads
    for (i = 0; i < 3; i++)
    {
        pthread_create(&tid, NULL, myThreadFun, (void *)&tid);
    }

    pthread_exit(NULL);  // Wait for all threads to complete
    return 0;
}
```

Thiscodecreatesthreethreadsthateachmodifystaticandglobalvariablesandprinttheirvalues:

- Globalvariable(g) ismodifiedbyall threads.
- Static variable (s) is local to the thread but retains its value between function callswithin thatthread.
- Eachthread prints itsuniqueIDalongwiththeupdated valuesoftheglobalandstaticvariables.

Task3:

Trytoexecutefollowingcode:
Modifypthread_createandconvertintoforloop
andcreatethreethreadsandshowoutput.Alsoremoves pthread_exitandseewhathappens.

Solution:

Codeexplanation:

1. Header Files: The necessary libraries (stdio.h, stdlib.h, unistd.h, pthread.h) areincluded to handle input/output, memory allocation, threading, and otherfunctions.
2. WorkerFunction:TheworkerThreadFunc printsamessageidentifyingthethread'sID.
3. Main Function: A single thread is created using pthread_create and runstheworkerThreadFunc.

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

// Worker thread function
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 store thread IDs
    for (long i = 0; i < 3; i++) {
        pthread_create(&tids[i], NULL, workerThreadFunc, (void *)&i);
    }
    // Removed pthread_exit
    return 0;
}
```

ObservationWithoutpthread_exit:

- Without pthread_exit, the main program may terminate before all threadscomplete,dependingon how theOSschedulesthreads.
- With pthread_exit, the main thread ensures that all threads complete before theprogramexits.

Task4:

Define POSIX thread and its working in your own words.

Solution:

POSIX Threads (commonly called Pthreads) are a standardized way to create and manage threads in operating systems that comply with the POSIX (Portable Operating System Interface) standard. A thread is a lightweight unit of execution within a process, and Pthreads provide a set of APIs (functions) to perform thread-related operations like creation, synchronization, and termination.

How Pthreads Work:

1. **Creation**:
A thread is created using the `pthread_create` function. Each thread runs a specific function provided by the user.

2. **Parallelism**:
Multiple threads can execute concurrently, sharing the same memory space of the parent process. This enables efficient multitasking and parallel processing.

3. **Synchronization**:
To prevent issues like race conditions, Pthreads offer synchronization tools such as mutexes (mutual exclusion locks) and condition variables.

4. **Termination**:
A thread can finish execution by returning from its function or by explicitly being terminated using functions like `pthread_exit`.

5. **Communication**:
Since threads share the same memory space, they can communicate directly by accessing shared variables. However, proper synchronization is required to avoid data inconsistency.