NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
ISLAMABAD

OPERATING SYSTEMS LAB SPRING 2019

Lab Manual 06
Introduction to Threads

1 WHAT IS A THREAD ?

"A thread is a sequence of control within a process"

To the software developer, the concept of a "procedure" that runs independently from its main program may best describe a thread.

2 PROPERTIES OF THREADS

Threads use and exist within process resources, yet are able to be scheduled by the operating system and run as independent entities largely because they duplicate only the bare essential resources that enable them to exist as executable code.
This independent flow of control is accomplished because a thread maintains its own:

1. Stack pointer
2. Registers
3. Scheduling properties (such as policy or priority)
4. Set of pending and blocked signals
5. Thread specific data

3 Summarize Threads

1. Exists within a process and uses the process resources.
2. Has its own independent flow of control as long as its parent process exists and the OS supports it.
3. Duplicates only the essential resources it needs to be independently schedulable.
4. May share the process resources with other threads that act equally independently (and dependently).
5. Dies if the parent process dies.
6. Is "lightweight" because most of the overhead has already been accomplished through the creation of its process.
7. Threads do Not Share:
   (a) Thread ID
   (b) Set of registers, including program counter and stack pointer
   (c) Stack (for local variables and return addresses)
   (d) Signal mask
   (e) Priority

4 Thread Implementation

In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995).

Implementations adhering to this standard are referred to as POSIX threads, or **Pthreads**. Pthreads are defined as a set of C language programming types and procedure calls, implemented with a **pthread.h** header

To link this library to your program use

```
gcc -pthread -o a.out myprogram.c
```

5 Thread Creation

Initially, your main() program comprises a single, default thread. All other threads must be explicitly created by the programmer.

pthread_create creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine)(void*), void *arg);
```

PTHREAD_CREATE ARGUMENTS:

1. **thread:** unique identifier for the new thread returned by the subroutine.
2. **attr:** An attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
3. **start_routine:** The C routine that the thread will execute once it is created.
4. **arg:** A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

6 PTHREAD FUNCTIONS

PTHREAD_SELF()

Return the thread ID of currently executing thread.

```
pthread_t pthread_self (void);
```

Returns the unique thread ID of the calling thread.

PTHREAD_EXIT()

Used to terminate threads

```
pthread_exit(*status)
```

This is also the mechanism used to get a return value from a thread. Several ways of termination:

1. The thread returns from its starting routine (the main routine for the initial thread).
2. The thread makes a call to the pthread_exit subroutine.
3. The thread is canceled by another thread via the pthread_cancel routine.
4. The entire process (main) is terminated due to a call to the exit subroutines.

EXAMPLE 01

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
pthread_t thread_id;      //creating object of type "pthread_t"
                          //for storing thread id
void printids(const char *s){
        pid_t pid;
        pthread_t tid;
        pid = getpid();
        tid = pthread_self();
        printf("%s pid %lu tid %lu\n", s, (unsigned long)pid,
                                        (unsigned long)tid);
 }

void *thr_fn(void *arg){  // New thread will call this function/routine.
        printids("New thread: ");
         return(NULL);
 }

int main(void){
        int ret;
        /*create a thread,store its id in thread_id with
                                        default attrib by passing NULL*/
        /* Thread will immediately execute "thr_fn"
                                        function with NULL parameters */
        ret = pthread_create(&thread_id, NULL, thr_fn, NULL);
        if (ret != 0)
                printf("can't create thread\n");
        printids("Main thread:");
       /* Exit from main() will cause new thread to destroy,
                                        lets give it some time */
        sleep(1);
        exit(0);
}
```

7 Passing Arguments to Threads

The pthread_create() routine permits the programmer to pass one argument to the thread start routine.

For cases where multiple arguments must be passed, this limitation is easily overcome by creating a structure which contains all of the arguments, and then passing a pointer to that structure in the pthread_create() routine.

All arguments must be passed by reference and cast to (void *).

Example 02

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
/* this function is run by the second thread */
void *inc_x(void *arg){
        /* increment x to 100 */
        printf("Thread: Process Id is %d and Thread Id is %ld\n",getpid()
                                                        ,pthread_self());
        int *input;
        input=(int *)arg;
        while(++(*input) < 100);
        printf("x increment finished\n");
        /* the function must return something - NULL will do */
        pthread_exit(NULL);
        }
void main(){     // main Thread;
        int x = 0, y = 0;
        /* show the initial values of x and y */
        printf("x: %d, y: %d\n", x, y);
        printf("main: Process Id is %d and Thread Id is %ld\n",getpid()
                                                       ,pthread_self());
        /* this variable is our reference to the second thread */
        pthread_t thread_2;

        /* create a second thread which executes inc_x(&x) */
        (pthread_create(&thread_2, NULL, inc_x, &x));

         printf("x: %d, y: %d\n", x, y);

         pthread_exit(NULL);

}
```

EXAMPLE 03

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
struct student_database{
        int rollno;
        float cgpa;
};

struct student_database obj;

void *print_fn(void *threadarg){
        struct student_database *temp_data;
        temp_data = (struct student_database *) threadarg;
        int rollno= temp_data->rollno;
        float cgpa = temp_data->cgpa;
        printf("Inside Thread\n");
        printf("Roll no %d has cgpa :%f\n",rollno,cgpa);
        pthread_exit(NULL);
}

int main(int argc, char *argv[]){
        pthread_t thread_id;
        obj.rollno=90010;
        obj.cgpa=3.88;
        printf("Creating thread\n");
        pthread_create(&thread_id, NULL, print_fn, &obj);
        pthread_exit(NULL);
}
```

8 Lab Tasks

Task 1

Write a c++ code, in which you have to create 10 threads using for loop, pass the value of for loop iterator as an arguments to the function of thread. In the function of thread print the value of Process ID, Thread ID and also the value of iterator passed to it. you also have to comment on the output of your code specially on the value of iterator that you passed to it.
Hint: Concept of independent stack and parallel processing/context switching helps.

Task 2

Write a c++ code, in which you have to create two threads other then your main thread. Create separate functions for each thread, pass the length and width of a square to both threads. First thread will calculate the area of the square and print it, while second thread will calculate perimeter and print it.