CENG328 Operating Systems
Laboratory Chapter 4

1 Introduction to Threads

Threads, also known as “lightweight processes”, helps achieving parallelization by executing different or the same instruction sequences in the scope of the same process concurrently. The following figure demonstrates the difference between multiple processes and multiple threads:

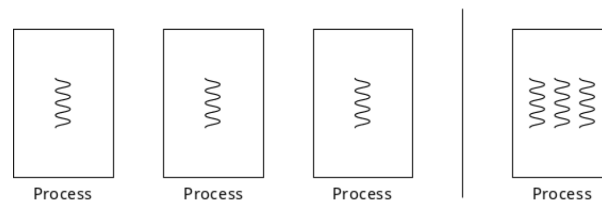

Figure 1: Multiple Processes vs Multiple Threads

The following code (th1.c) demonstrates how to create 2 new threads and assign them different tasks to perform:

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

void* threadFun1(void* arg) {
  while(1)
    printf("1");
}

void* threadFun2(void* arg) {
  while(1)
    printf("0");
}

int main() {
  pthread_t thread1, thread2;
  pthread_create(&thread1, NULL, threadFun1, NULL);
  pthread_create(&thread2, NULL, threadFun2, NULL);

  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);
  return 0;
}
```

In addition to including pthread header in the code, all pthread programs must be compiled against pthread library with gcc as well:

```
gcc th1.c -o th1 -lpthread
```

When executed, the main thread (which has already been created and started by the hosting process) creates two additional threads and tells them to execute the functions threadFun1 and threadFun2 respectively. You may stop the program using Ctrl + C.

pthread_create() function creates a new thread. It takes 4 parameters:

1. **thread variable** - the newly created thread will be attached to this variable.
2. **thread attributes** - thread attributes such as scope, scheduling, etc can be defined here. If default attributes are to be used, this variable can be NULL.
3. **thread function** - the created thread will execute the instructions stored in this function.
4. **thread arguments** - Function arguments can be defined here. If no arguments will be used, this variable can be NULL.

pthread_join() function blocks until the related thread is terminated, before continuing execution. If not used, all running threads will be terminated altogether when the main thread has finiseed. Its parameters are:

1. **thread variable** - which thread to be waited for.
2. **thread return code** - to trace how the thread has terminated. If return code is unimportant for your code, it can be NULL.

Since all threads reside in the same process, all of them will have the same process id but they still will have different thread id's (th2.c):

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

void* threadFun(void* arg) {
  printf("Thread %u - pid = %d\n", pthread_self(), getpid());
}

int main() {
  pthread_t threads[5];
  int i;
  for (i = 0; i < 5; ++i)
    pthread_create(&threads[i], NULL, threadFun, NULL);
  for (i = 0; i < 5; ++i)
    pthread_join(threads[i], 0);
  return 0;
}
```

The 4th parameter of pthread_create function takes address of a variable, therefore instead of copying seperate values to each thread, all threads will access the same value stored at that address. This may sometimes cause unwanted errors. Take a look at the code below (th3.c). When you run it, you will see some of the threads report the same or different values. This is because the value stored at the address may be accessed, updated and printed by more than one threads simultaneously.

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

void* threadFun(void* arg) {
  int a = *((int*) arg);
  printf("i = %d\n", a);
}

int main() {
  pthread_t threads[50];
  int i;
  for (i = 0; i < 50; ++i) {
    pthread_create(&threads[i], NULL, threadFun, &i);
  }
  for (i = 0; i < 50; ++i) {
    pthread_join(threads[i], NULL);
  }
  return 0;
}
```

For this reason, seperate variables should be passed as function parameters as required (th4.c):

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

void* threadFun(void* arg) {
  int a = *((int*) arg);
  printf("id = %d\n", a);
}

int main() {
  pthread_t threads[50];
  int i, ids[50];
  for (i = 0; i < 50; ++i) {
    ids[i] = i;
    pthread_create(&threads[i], NULL, threadFun, &ids[i]);
  }
  for (i = 0; i < 50; ++i) {
    pthread_join(threads[i], NULL);
  }
  return 0;
}
```

If more than one variable is needed, they must be encapsulated in a struct and that struct must be used as the parameter instead (th5.c):

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

typedef struct _course {
  char* code;
  char* name;
} COURSE;

void* threadFun(void* arg) {
  COURSE c = *((COURSE*) arg);
  printf("Welcome to %s - %s\n", c.code, c.name);
}

int main() {
  pthread_t threads[2];
  COURSE courses[2];
  int i;

  courses[0].code = "CENG328";
  courses[0].name = "Operating Systems";
  courses[1].code = "CENG501";
  courses[1].name = "Advanced Operating Systems";

  for (i = 0; i < 2; ++i)
    pthread_create(&threads[i], NULL, threadFun, &courses[i]);

  for (i = 0; i < 2; ++i)
    pthread_join(threads[i], NULL);

  return 0;
}
```

2 Exercises

1. Read man pages for the following library functions: pthread_create, pthread_self, pthread_join.
2. Modify th1.c such that both threads stop after printing 50 integers.
3. Remove pthread_join lines in th1.c to observe the importance of this function.