**Trent University: Operating Systems (COIS3320)**

**Dr. Bin Guo**

**Lab 5: Threads**

**Outline**

In this lab you will learn how to create threads using the Pthreads (POSIX standard) API.

1. The C program shown on the next page demonstrates the basic Pthreads API for

constructing a multithreaded program that calculates the summation of a non-

negative integer in a separate thread.

2. In a Pthreads program, separate threads begin execution in a specified function. In

the code below, this is the runner() function.

3. When this program begins, a single thread of control begins in main().

4. After some initialization, main() creates a second thread that begins control in the

runner() function. ***Both threads share the global data sum.***

5. All Pthreads programs must include the pthread.h header file.

6. The statement pthread\_t tid declares the identifier for the thread we will create.

Each thread has a set of attributes, including stack size and scheduling information.

The pthread\_attr\_t attr declaration represents the attributes for the thread.

We set the attributes in the function call pthread\_attr\_init(&attr). Because

we did not explicitly set any attributes, we use the default attributes provided.

7. A separate thread is created with the pthread\_create() function call. In addition

to passing the thread identifier and the attributes for the thread, we also pass the

name of the function where the new thread will begin execution—in this case, the

runner() function. Last, we pass the integer parameter that was provided on the

command line, argv[1].

8. At this point, the program has two threads: the initial (or parent) thread in main()

and the summation (or child) thread performing the summation operation in the

runner() function. This program follows the thread *create/join* strategy, whereby

after creating the summation thread, the parent thread will wait for it to terminate by

calling the pthread\_join() function.

9. The summation thread will terminate when it calls the function pthread\_exit().

10. Once the summation thread has returned, the parent thread will output the value of

the shared data sum.

A white screen with black text

Description automatically generated

In your programs you may need to create more than one thread.A simple method for

waiting on several threads using the pthread\_join() function is to enclose the operation

within a simple for loop.

A white background with black text

Description automatically generated

**Lab question**

Given a list of size 20 consisting of natural numbers, write a multithreaded C program for

adding all the numbers in the list as follows: The list of numbers is divided into two smaller

lists of equal size. Two separate threads (which we will term as ***summing threads***) add

numbers in each sublist.

Because global data are shared across all threads, the easiest way to set up the data is to

create a global array. Each ***summing thread*** will work on one half of this array. This lab will

require passing parameters to each of the **summing threads**. In particular, it will be

necessary to identify the starting index and ending index of the sublist in which each thread

is to begin adding numbers. The parent thread will output the sum once all summing

threads have exited.

You are to write a C program using Pthreads that contains the entire solution for this

question. In particular your program needs to do the following:

1. To be able to create threads in your C program you need to include the pthread.h

header file.

2. Each thread has a unique thread ID. To create thread IDs for your threads in your

program you should use the pthread\_t data type.

3. Thread attributes should be created/modified using pthread\_attr\_t structure.

4. Declare and code the function in which the thread begins control. For an example

see the runner() function on the previous page.

5. To be able to identify the starting index and ending index of the sublist in which each

thread begins adding numbers you can do the following:

a. Create a structure to store the starting index and ending index of the sublist.

For example:

typedef struct { int from\_index;

int to\_index; } parameters;

6. Since threads can share heap, you can simply create a variable of type

parameters, allocate memory for it on the heap, and assign values to its members

as follows:

parameters \*data =

(parameters \*) malloc (sizeof(parameters));

data->from\_index = 0; data->to\_index = (SIZE/2) - 1;

7. To create threads use pthread\_create() function and pass in the necessary

parameters.

8. For the parent thread to output the sum after all summing threads have exited, it is

important that you use the pthread\_join() function.

9. Whenever you dynamically allocate memory on the heap, it is important that you

deallocate/free this memory when it is not required by the program using the

free()function.

10. To compile your program, you need to use the –pthread option as follows:

gcc -pthread -o Lab5 Lab5.c

**Sample Output for the following list:**

List={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}

./Lab4

------------------------------------------------------------------------------------------------------------

Sum of numbers in the list is: 210