**CSC 338 Parallel and Distributed Computing**

**Exercise No. 7, April 10, 2017**

**Introduction to Pthreads programming**

**Goal**

Learn the basics of Pthreads

**Background**

Pthreads (POSIX threads) is a library for shared memory parallel computing.

**Procedure**

1. Type and save the program (as pth\_hello.c) on the back of this page.
2. Compile the program (use Cygwin or Linux):

gcc –g –Wall –o pth\_hello pth\_hello.c -lpthread

1. Execute the program with multiple threads: ./pth\_hello 8
2. Answer the following questions:
   1. In Pthreads programs, global variables are shared among all threads; each thread has a private copy of local variables and function arguments. What variables, if any, are shared in pth\_hello?

Global: thr\_count,   
Plus - rank, thr\_handles

* 1. What do you think strtol() does?

Strtol() changes string to long

* 1. Note that the call to malloc allocates enough memory to handle thr\_count thread handles.
  2. Now look at the loop that creates the threads. The first argument is a pointer to a pthread\_t object. This is an *opaque* object that is system dependent. The second argument has to do with thread attributes and we will pass NULL. The third argument is the target function that the thread will execute. There is some trickery going on with the fourth argument. First, we are giving the thread a rank because we can't use pthread\_t to identify threads (although the operating system can). We are passing an integer but telling the function that it's a pointer; the create function wants a pointer because it expects a list of arguments. In the function, we cast the argument to a long int. This could cause a problem if the pointer type is a different size than the type of the argument—that's why we use long. On some systems an int is 32 bits but a pointer is 64 bits. A long int should be 64 bits on those systems.
  3. Note that we don't explicitly have to start the threads—the create function does that.
  4. The value of my\_rank for thread 3 is 2 .
  5. What do you think pthread\_join() does?

Combines all the threads back to the master after they are forked ad waits for each thread to complete.

* 1. What do you think free() does? We should always use free() to release any memory allocated by malloc() but it’s hard to remember that because we’ve gotten so used to languages that do garbage collection.

Free() frees up the memory that was allocated. Similar to using delete after creating a new object in C++

/\* Pthreads hello from Pacheco, Ch4 \*/

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

/\* Global variable: accessible to all threads \*/

int thr\_count;

void\* Hello(void\* rank); //Thread function

int main(int argc, char\* argv[]) {

long thr; //Use long in case of 64 bit system

pthread\_t\* thr\_handles;

//Get number of threads from cmd line

thr\_count = strtol(argv[1], NULL, 10);

thr\_handles = malloc(thr\_count \* sizeof(pthread\_t));

for (thr = 0; thr < thr\_count; thr++) {

pthread\_create(&thr\_handles[thr], NULL, Hello, (void\*) thr);

}

printf("Hello from the main thread!\n");

for (thr = 0; thr < thr\_count; thr++) {

pthread\_join(thr\_handles[thr], NULL);

}

free(thr\_handles);

return 0;

} //main

void\* Hello(void\* rank) {

long my\_rank = (long) rank; //long for 64 bit system

printf("Hello from thread %ld of %d\n", my\_rank, thr\_count);

return NULL;

} //Hello