LAB NO.: 6 Date:

PROGRAMS ON THREADS

Objectives:

In this lab, the student will be able to:

- Understand the concepts of multithreading
- Grasp the execution of the different processes concerning multithreading

A process will start with a single thread which is called the main thread or master thread. Calling pthread_create() creates a new thread. It takes the following parameters.

- A pointer to a pthread_t structure. The call will return the handle to the thread in this structure.
- A pointer to a pthread attributes structure, which can be a null pointer if the default attributes are to be used. The details of this structure will be discussed later.
- The address of the routine to be executed.
- A value or pointer to be passed into the new thread as a parameter.

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

void* thread_code( void * param )
{ printf( "In thread code\n" );
}       int
main()
{ pthread_t thread;
        pthread_create(&thread, 0, &thread_code, 0 ); printf("In
        main thread\n" );
}
```

In this example, the main thread will create a second thread to execute the routine thread_code(), which will print one message while the main thread prints another. The call to create the thread has a value of zero for the attributes, which gives the thread default attributes. The call also passes the address of a pthread_t variable for the function to store a handle to the thread. The return value from the thread_create() call is zero if the call is successful; otherwise, it returns an error condition.

Thread termination :

Child threads terminate when they complete the routine they were assigned to run. In the above example, child thread will terminate when it completes the routine thread_code().

The value returned by the routine executed by the child thread can be made available to the main thread when the main thread calls the routine pthread_join().

The pthread_join() call takes two parameters. The first parameter is the handle of the thread that is to be waited for. The second parameter is either zero or the address of a pointer to a void, which will hold the value returned by the child thread.

The resources consumed by the thread will be recycled when the main thread calls pthread_join(). If the thread has not yet terminated, this call will wait until the thread terminates and then free the assigned resources.

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

void* thread_code( void * param )
{ printf( "In thread code\n" );
}       int
main()
{ pthread_t thread;
        pthread_create( &thread, 0, &thread_code, 0 );
        printf( "In main thread\n" ); pthread_join(
        thread, 0 );
}
```

Another way a thread can terminate is to call the routine pthread_exit(), which takes a single parameter either zero or a pointer to void. This routine does not return and instead terminates the thread. The parameter passed into the pthread_exit() call is returned to the main thread through the pthread_join(). The child threads do not need to explicitly call pthread_exit() because it is implicitly called when the thread exits.

Passing Data to and from Child Threads

In many cases, it is important to pass data into the child thread and have the child thread return status information when it completes. To pass data into a child thread, it should be cast as a pointer to void and then passed as a parameter to pthread_create().

```
for ( int i=0; i<10; i++ ) pthread_create( &thread, 0,
&thread_code, (void *)i );
```

Following is a program where the main thread passes a value to the Pthread and the thread returns a value to the main thread.

```
#include <pthread.h> #include
<stdio.h>
void* child_thread( void * param )
{ int id = (int)param; printf( "Start
        thread %i\n", id ); return (void
        *)id;
}

int main()
{ pthread_t thread[10]; int
        return_value[10]; for (
        int i=0; i<10; i++ )
        { pthread_create( &thread[i], 0, &child_thread, (void*)i );
        }
        for ( int i=0; i<10; i++ )
        { pthread_join( thread[i], (void**)&return_value[i] ); printf(
        "End thread %i\n", return_value[i] ); }
}
```

Setting the Attributes for Pthreads

The attributes for a thread are set when the thread is created. To set the initial thread attributes, first, create a thread attributes structure, and then set the appropriate attributes in that structure, before passing the structure into the pthread_create() call.

```
#include <pthread.h>
...
int main()
{ pthread_t thread; pthread_attr_t attributes; pthread_attr_init(
        &attributes ); pthread_create( &thread, &attributes,
        child_routine, 0 );
}
```

LAB NO.: 6

Lab Exercises:

1. Write a multithreaded program that generates the Fibonacci series. The program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program then will create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that is shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution the parent will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish.
2. Write a multithreaded program that calculates the summation of non-negative integers in a separate thread and passes the result to the main thread.
3. Write a multithreaded program for generating prime numbers from a given starting number to the given ending number.
4. Write a multithreaded program that performs the sum of even numbers and odd numbers in an input array. Create a separate thread to perform the sum of even numbers and odd numbers. The parent thread has to wait until both the threads are done.

Additional Exercises:

1. Write a multithreaded program for matrix multiplication.
2. Write a multithreaded program for finding row sum and column sum