**AAiT ITSE 4931: Operating Systems Lab 1 – Threads**

# PART 0. INTRODUCTION

**Submission**

Include the following in your lab report.

* Your full names in the upper right corner of the first page in large print.
* A cohesive summary of what you learned through the 11 exercises in this lab. Try to get at the main idea of the exercises, and write your answers to the questions. You have 2 hours to finish.

**Note:** Those student who did not submit last week’s laboratory report must submit today along with this lab report.

**Resources for this lab**

Use the class lecture slides (Get them from your instructor or other students).

**Objectives**

This lab examines aspects of threads and multiprocessing (and multithreading). The primary objective of this lab is to implement the *Thread Management* Functions:

* Creating Threads
* Terminating Thread Execution
* Joining Threads

**Introduction**

A **thread** is a semi-process, that has its own *stack*, and executes a given piece of code. Unlike a real process, the thread normally shares its memory with other threads (where as for processes we usually have a different memory area for each one of them).

A *Thread Group* is a set of threads all executing inside the same process. They all share the same memory, and thus can access the same global variables, same heap memory, same set of file descriptors, etc. All these threads execute in *parallel* (i.e. using time slices, or if the system has several processors, then really in parallel).

**Pthreads** areLinux implementation of thread management mechanism. **Pthreads** are defined as a set of C language programming types and procedure calls. Vendors usually provide a Pthreads implementation in the form of a header/include file and a library, which you link with your program. Here is the list of the two names used in pthreads very often.

|  |  |
| --- | --- |
| pthread\_t | Thread objects |
| pthread\_attr | Thread attributes objects |

**PART I: Thread Creation and Waiting in Pthreads**

The function pthread\_create is used to create a new thread, and a thread to terminate itself uses the function pthread\_exit. A thread to wait for termination of another thread uses the function pthread\_join. Here is a description of pthread\_create and pthread\_join. Later we will see pthread\_exit function.

|  |  |
| --- | --- |
| int pthread\_create (  threadhandle, */\* Thread variable (of type pthread\_t) returned by reference\*/*  pthread\_attr\_t \*attribute,*/\*Special Attribute for starting thread,usually* *NULL\*/*  start\_routine, */\* Function which thread executes; \*/*  arg */\*A single argument that may be passed to start\_routine. It must be passed by reference as a pointer cast of type void (see lab3.c in page4. NULL may be used if no argument is to be passed.\*/*  ); | |
| ***Description:*** Request the PThread library for creation of a new thread. The return value is 0 on success. The return value is negative on failure. The *pthread\_t* is an abstract *datatype* that is used as a handle to reference the thread. | |
|  | |
| int pthread\_join(  pthread\_t threadhandle, /\* Pass threadhandle \*/  void \*\*returnvalue /\* Return value is returned by ref. \*/  ); |
| ***Description:*** Return 0 on success, and negative on failure. The returned value is a pointer, usually its value is NULL. Since we don’t care about the returned value. |

Initially, threads are created from within a process. Once created, threads are peers, and may create other threads. Note that an "initial thread" exists by default and is the thread, which runs main.

**/\*Lab1.c - Creating Threads \*/**

#include <stdio.h>

#include <pthread.h>

/\* All thread functions and datatypes are defined in pthread.h \*/

void \*kidfunc(void \*p)

printf ("Kid ID is ---> %d\n", getpid( ));

}

void main(){

/\* Declare a variable of type pthread\_t :\*/

pthread\_t kid;

pthread\_create (&kid, NULL, kidfunc, NULL);

printf ("Parent ID is ---> %d\n", getpid());

pthread\_join (kid, NULL);

printf ("No more kid!\n") ;

}

**NOTE:** When you compile the program you need to tell the the compiler (gcc) that it is going to use a library called pthreads and gcc needs to link our program with this library to produce an executable program. This can be done by adding -lpthread to the linker flags as:

gcc lab1.c –o threads -lpthread

Then you can run the program as:

./threads

1. Compile and run the above program called lab1.c. What is displayed (printed) by the program?
2. How many threads are there in the above program? Which tread executes which statement? Write down the sequence of instructions executed by the processor which led to producing the output you wrote down in question #1.  
   Were there any context switches between the threads?
3. Are the process id numbers of the threads the same or different? why?
4. Which thread finishes execution first? Why?

**/\*Lab2.c\*/**

#include<stdio.h>

#include <pthread.h>

int glob\_data = 5;

void \*kidfunc(void \*p){

printf ("Kid here. Global data was %d.\n", glob\_data);

glob\_data = 15;

printf ("Kid Again. Global data was now %d.\n", glob\_data);

}

void main (){

pthread\_t kid;

pthread\_create (&kid, NULL, kidfunc, NULL);

printf ("Parent here. Global data = %d\n", glob\_data);

glob\_data = 10;

pthread\_join (kid, NULL);

printf ("End of program. Global data = %d\n", glob\_data);

}

1. Compile and run the above program called lab2.c. What is displayed (printed) by the program?
2. How many threads are there in the above program? Which tread executes which statement? Write down the sequence of instructions executed by the processor which led to producing the output you wrote down in question #5.  
   Were there any context switches between the threads?
3. Do the threads have separate copies of glob\_data?
4. modify the program lab2.c by add the following statements after pthread\_create statement.  
   **printf ("Just stalling to get context switched\n");  
   printf ("I should get context switched by now\n");**Do exercise 5 and 6 for the modified program too.

**PART II: Thread Termination and attributes**

There are several ways in which a Pthread may be terminated. The main reasons are:

* The thread returns from its starting routine (the main routine for the initial thread). By default, the Pthreads library will reclaim any system resources used by the thread. This is similar to a process terminating when it reaches the end of main.
* The thread makes a call to the **pthread\_exit** function (covered below).
* The thread is canceled by another thread via the **pthread\_cancel** function (not covered here; please refer on the Internet).

|  |
| --- |
| void pthread\_exit ( ***/\*is similar to exit() you saw in lab 1\*/***  void \*retval */\* return value passed as a pointer \*/*  ); |
| ***Description:*** This Function is used by a thread to terminate. The return value is passed as a pointer. This pointer value can be anything including NULL. For details refer to the Internet. |

The simple example code below creates 5 threads with the pthread\_create( ) routine. Each thread prints a "Hello World!" message, and then terminates with a call to pthread\_exit( ).

**/\*Lab3.c\*/**

#include <pthread.h>

#include <stdio.h>

The variable t, will be passed as an argument to i.e PrintHello as a void pointer. It basically means PrintHello has to worry about the datatype.

PrintHello accepts t using the argument threadid

#define NUM\_THREADS 5

void \*PrintHello(**void \*threadid**)

{

printf("\n**%d**: Hello World!\n", **threadid**);

pthread\_exit(NULL);

}

int main( )

{

pthread\_t threads [NUM\_THREADS];

int rc, t;

for(t=0; t < NUM\_THREADS; t++){

printf ("Creating thread %d\n", t);

rc=pthread\_create(&threads[t], NULL, PrintHello,**(void\*) t** );

if (rc) {

printf("ERROR;return code from pthread\_create()=%d\n",rc);

exit(-1);

}

}

pthread\_exit(NULL);

}

1. Compile and run the above program called lab3.c. What is displayed (printed) by the program?
2. How many threads are there in the above program? Which tread executes which statement? Write down the sequence of instructions executed by the processor which led to producing the output you wrote down in question #9.  
   Were there any context switches between the threads?
3. Did all the threads finish execution before the program terminated? Why didn’t they finish? Fix the program so that it waits for all the threads to finish execution before it terminates. and write the modified version of the program in your report.