CS 416 - Operating Systems Design

Homework 0
User Level Threads

January 22, 2016
Due February 7th, 2016

1 Introduction

In this homework, you will gain experience with multithreaded systems by implementing your own threading library.

2 User Level Thread Library

For this part you will implement a cooperative User Level Thread (ULT) library for Linux that can replace the default PThreads libary. Cooperative user level threading is conceptually similar to a concept known as coroutines, in which a programming language provides a facility for switching execution between different contexts, each of which has its own independent stack. A very simple program using coroutines might look like this:

```
void coroutine1()
{
     // Some work
     yield(coroutine2);
}

void coroutine2()
{
     // Some different work
     if (!done)
          yield(coroutine1);
}

int main()
{
     coroutine1();
}
```

Note that *cooperative* threading is fundamentally different than the *preemptive* threading done by many modern operating systems in that every thread **must** yield in order for another thread to be scheduled.

2.1 Basic User Level Thread Library

Write a **non-preemptive cooperative user-level** thread library that operates similarly to the Linux pthreads library, implementing the following functions:

- mypthread_create
- mypthread_exit
- mypthread_yield
- mypthread_join

Please note that we are prefixing the typical function names with "my" to avoid conflicts with the standard library. In this ULT model, one thread yields control of the processor when one of the following conditions is true:

- thread exits by calling **mypthread_exit**
- thread explicitly yields by calling **mypthread_yield**
- thread waits for another thread to terminate by calling **mypthread_join**

You should use the functionality provided by the Linux functions **setcontext()**, **getcontext()**, and **swapcontext()** in order to implement your thread library. See the Linux manual pages for details about using these functions.

2.2 Requirements

We will provide a Makefile, test program, and simple template. You may not modify the test program, so your library must provide exactly the API that we specify. You should implement all of your code in the provided files **mypthread.h**, and **mypthread.c**.

3 Coding and Submission Instructions

3.1 Coding

Along with this PDF, you will find on Sakai a file named **homework0-template.tar.gz**. To get started, download this file to a Linux/Unix system and type **tar zxvf homework0-template.tar.gz** at your console. This will produce a new directory called **homework0**. Inside you will find a directory named **ult**. Inside this directory, you will find some template source and header files, as well as a Makefile. For this assignment, you should NOT have to create any other files, simply do your implementation in these existing files.

3.1.1 Implementation

We provide a **mypthread.h** that defines the required API. You will need to make changes to the types defined here, but do not change the function call API. You will implement your library in **mypthread.c**. Once you complete your implementation, you may compile it by typing **make ult**. You may also type **make system** to produce a version of the test program compiled against the system pthreads library (to see how it is supposed to work). The test program is implemented in **mtsort.c**, and the compiled versions will be **mtsort-ult** and **mtsort-system**, respectively.

3.2 Submission

When you have completed the project, **cd** to the top level directory (**homework0**) and type **make repack netid=YOUR_GROUPID**. This will produce a file called **homework0-YOUR_GROUPID.tar.gz**. Submit this file on Sakai along with a report in PDF or TXT format (no MS, OpenOffice, or LibreOffice).

Only one group member should submit the project, but the *names and netids* of all group members should be entered into the text field on the Sakai submission, and should also appear on the report.

4 Notes

- The code has to be written in the **C** language. You should discuss on Piazza if you think inline asm is necessary to accomplish something.
- Do not copy the solution from other students. Use Piazza to discuss ideas of how to implement it. Do not post your solution there. Use Sakai to submit it.