Com S 352 Fall 2016 Project 1: A User-level Thread Library

Due: Wednesday, October 26, 11:59pm

In this project, you are required to develop a user-level thread library called ***uthread***. The thread library should create user-level threads (upon request from some user application code), map the user-level threads to multiple kernel threads following the many-to-many mapping model, schedule the mapping based on the amount of time each user-level thread has been mapped to a kernel thread (i.e., when a kernel thread becomes available, the user-level thread that has been mapped to a kernel thread for the least amount of time should be mapped to the kernel thread; when there are multiple user-level threads that have been mapped to kernel threads for the same amount of time, one of these threads that arrives at the ready queue the earliest should be selected for mapping).

Thread Library Interface

The user-level thread library that you develop should provide the following functions:

void uthread_init(int numKernelThreads)

This function has to be called before any other functions of the uthread library can be called. It initializes the uthread system and specifies the maximum number of kernel threads to be argument ***numKernelThreads***. For example, it may establish and initialize a priority ready queue of user-level threads (with the amount of time each user-level thread has been mapped to kernel threads as priority number) and other data structures.

int uthread_create(void (* func)( ))

The calling thread requests the thread library to create a new user-level thread that runs the function func(), which is specified as the argument of this function. At the time when this function is called, if less than numKernelThreads kernel threads have been active, a new kernel thread is created to execute function func(); otherwise, a context of a new user-level thread should be properly created and stored on the priority ready queue. This function returns 0 if succeeds, or -1 otherwise.

void uthread_yield()

The calling thread requests to yield the kernel thread to another user-level thread with the same or higher priority (note: the priority is based on the time a thread has been mapped to kernel threads). If each ready thread has lower priority than this calling thread, the calling thread will continue its running; otherwise, the kernel thread is yielded to a ready thread with the highest priority.

void uthread_exit()

This function is called when the calling user-level thread terminates its execution. In response to this call, if no ready user-level thread in the system, the whole process terminates; otherwise, a ready user thread with the highest priority should be mapped to the kernel thread to run.

Sample Test Code 1 and Output 1

```
#include "uthread.h"

void th1()

{

        int i;

        for(i=0;i<8;i++){

                printf("Thread 1: run.\n");

                sleep(1); //note: during sleep the user-level thread is still mapped

                printf("Thread 1: yield.\n");

                uthread_yield();

        }

        printf("Thread 1: exit.\n");

        uthread_exit();

}


void th2()

{

        int i;

        for(i=0;i<4;i++){

                printf("Thread 2: run.\n");

                sleep(2);

                printf("Thread 2: yield.\n");

                uthread_yield();

        }

        printf("Thread 2: exit.\n");

        uthread_exit();
```

```
}

void th3()
{
        int i;
        for(i=0;i<2;i++){
                printf("Thread 3: run.\n");
                sleep(4);
                printf("Thread 3: yield.\n");
                uthread_yield();
        }
        printf("Thread 3: exit.\n");
        uthread_exit();
}

int main()
{
        uthread_init(1);
        uthread_create(th1);
        uthread_create(th2);
        uthread_create(th3);
        uthread_exit();
}
```

Output:

```
Thread 1: run.
```

Thread 1: yield.

Thread 2: run.

Thread 2: yield.

Thread 3: run.

Thread 3: yield.

Thread 1: run.

Thread 1: yield.

Thread 2: run.

Thread 2: yield.

Thread 1: run.

Thread 1: yield.

Thread 1: run.

Thread 1: yield.

Thread 3: run.

Thread 3: yield.

Thread 2: run.

Thread 2: yield.

Thread 1: run.

Thread 1: yield.

Thread 1: run.

Thread 1: yield.

Thread 2: run.

Thread 2: yield.

Thread 1: run.

Thread 1: yield.

Thread 1: run.

Thread 1: yield.

Thread 3: exit.

Thread 2: exit.

Thread 1: exit.

(The order could be slightly different.)

Sample Test 2

Code: only change from Test code 1: uthread_init(1) ➔ uthread_init(2)

Output:

Thread 2: run.

Thread 1: run.

Thread 1: yield.

Thread 3: run.

Thread 2: yield.

Thread 1: run.

Thread 1: yield.

Thread 2: run.

Thread 3: yield.

Thread 1: run.

Thread 2: yield.

Thread 3: run.

Thread 1: yield.

Thread 1: run.

Thread 1: yield.

Thread 2: run.

Thread 3: yield.

Thread 1: run.

Thread 2: yield.

Thread 2: run.

Thread 1: yield.

Thread 1: run.

Thread 1: yield.

Thread 1: run.

Thread 2: yield.

Thread 3: exit.

Thread 2: exit.

Thread 1: yield.

Thread 1: run.

Thread 1: yield.

Thread 1: exit.

(The order could be slightly different)

Submission

- You should use C or C++ to develop the code.
- You need to turn in your library source code (named uthread.c), a header file (named uthread.h) declaring the functions exposed by your library, and a sample test (named test.c; you develop arbitrarily) code that uses the library functions.
- You need to turn in electronically by submitting a zip file named Firstname_Lastname_Project1.zip.
- Source code must include proper documentation to receive full credit.
- All projects require the use of a make file or a certain script file (accompanying with a readme file to specify how to use the script file to compile), such that the grader will be able to compile/build your executable by simply typing "make" or some simple command you specifies in your readme file.
- Source code must compile and run correctly (No memory leakage during the running of your library is recommended but not required; but make sure the system can sustain for long time) on the department machine pyrite, which will be used by the TA for grading.
- You are responsible for thoroughly testing and debugging your code. The samples provided are for explanation/illustration purpose only; the TA may try to break your code by subjecting it to bizarre test cases.
- You can have multiple submissions, but the TA will grade only the last one.

Hints

You may need the following tools for this project (If you are not familiar, review related lecture nodes or book sections):

- clone system call;
- system calls for managing user thread contexts: getcontext, makecontext, setcontext, and swapcontext (Lecture 8);
- function gettimeofday() defined in <sys/time.h>; and
- system call syscall(SYS_gettid) defined in <unistd.h> and <sys/syscall.h> that can be used to obtain the ID of the caller thread.

You are suggested to review the many2one-mapping-v1.c code posted with Lecture 8, to understand how to implement yield and exit functions for user-level threads.

Start as early as possible!