CS 5600 – Systems and Networks Prof. Peter Desnoyers
Homework 2 – User-space threads

In this homework you will write a user-space thread system (named “qthreads”, vs. the standard “pthreads” library), which creates threads, performs context switches, and includes implementations of mutexes and condition variables. You will use your code in two applications – your own test procedures, and a simple threaded webserver provided as part of the example. You will be responsible for writing a thorough test suite for your code.

Rules

I expect you to work in teams of two students; you will submit one copy of the homework and receive the same grade. Feel free to discuss the ideas in the homework with other groups; however you must write answers to written questions in your own words, and absolutely no sharing of code across groups is allowed. You do not need to explicitly submit your assignment – your grade will be based on the code in your repository at the time it is due.

Commit and push your code at the end of every day that you work on it, or (preferably) more frequently. **Include a git message which describes what you have done.** If you forget to add a message, you can add one with the ‘git commit –append’ command; if you have already pushed your commit you will need to use ‘git push –force’ to push the ammended comment.

If you do not push your code sufficiently frequently, or provide comments describing your work, you may lose points.

Note that code similarity across groups, a lack of understanding of your submitted code, or evidence of variables changed by search-and-replace may be considered evidence of academic dishonesty. (if you decide that you really hate the names that you put in your code, please leave a git comment when you change them)

Programming Assignment materials and resources

You will download the skeleton code for the assignment from the CCIS repository server, trac.ccs.neu.edu, using the '`git clone`' command:

```
git clone https://github.ccs.neu.edu/cs5600-01-f16/team-nn-hw2
```

where '*nn*' is your team number. Periodically you will commit checkpoints of your work into your local repository using 'git `commit`':

```
git commit -a -m 'message describing the checkin'
```

and push the commits to the central repository:

```
git push
```

You should be able to complete this homework on any 32-bit x86 Linux system (**NOT 64-bit)**; however, the official environment is the CS-5600 virtual machine which you used for Homework 1.

Repository Contents

The repository you clone contains the following files:

qthread.c – this is the file you will be implementing. It has some comments describing the functions and types you have to implement.

qthread.h – analogous to pthread.h, this defines the types used by qthread applications. Note that I've defined qthread_mutex_t and qthread_cond_t as structures with a single pointer to an "implementation" structure that you'll define in qthread.c.

switch.s, stack.c – these contains the context switch function (`switch_from_to`) and the stack initialization function (`init_stack`). You should not change these files. The functions are more complex than the the ones discussed in class because they contain protection code designed to abort into the debugger if you try to switch to an invalid stack.

test1.c – this is where your test code should go. (well, you can add additional files and update the makefile if you do)

server.c – a simple multi-threaded webserver using qthreads.

Makefile – this is set up to build executables for 'test1' and 'server'. Compile your code by typing 'make'.

Deliverables

The following files from your repository will be examined, tested, and graded:

qthread.c

test1.c

Additional information

More information will be posted about how to complete this assignment. Note that it's possible to implement qthreads in less than 300 lines of code – it doesn't have to be huge and complex. (my implementation has 257 lines of code, of which about 50 are the skeleton code you've been provided)

Qthreads interface

First, two function pointer typedefs – one for a function taking two arguments (with no return value) and the other taking one argument and returning 'void*'. These will make the following definitions easier to read.

```
typedef void (*f_2arg_t)(void*, void*);
typedef void * (*f_1arg_t)(void*);
```

You will need to implement the following functions – first, the basic functions to start and exit a thread, run the thread scheduler, and wait for a thread to finish ("join").

```
qthread_t qthread_start(f_2arg_t f, void *arg1, void *arg2);
qthread_t qthread_create(f_1arg_t f, void *arg1);
void qthread_run(void);
void qthread_exit(void *arg);
void *qthread_join(qthread_t thread);
```

(why did I add the qthread_start function, when pthreads only has qthread_create? Because you don't want the thread to start with the user function – you need a wrapper function to call qthread_exit with the return value if the user function exits)

Mutex and condition variable functions:

```
void qthread_mutex_init(qthread_mutex_t *m);
void qthread_mutex_lock(qthread_mutex_t *m);
void qthread_mutex_unlock(qthread_mutex_t *m);
void qthread_mutex_destroy(qthread_mutex_t *m);
void qthread_cond_init(qthread_cond_t *c);
void qthread_cond_wait(qthread_cond_t *c, qthread_mutex_t *m);
void qthread_cond_signal(qthread_cond_t *c);
qthread_cond_broadcast(qthread_cond_t *c);
qthread_cond_destroy(qthread_cond_t *c);
```

And finally functions to replace some of the blocking sleep and I/O calls in the Unix interface:

```
void qthread_usleep(long int microsecs);
ssize_t qthread_read(int fd, void *buf, size_t len);
ssize_t qthread_write(int fd, void *buf, size_t len);
int qthread_accept(int sock, struct sockaddr *addr, socklen_t *len);
```

Note that a full user-space threads package like GNU Pth (https://www.gnu.org/software/pth/) would transparently "wrap" all the blocking I/O calls with functions like this. (don't try to use Pth to implement this – it's way too big and complex)