EC440: Project 2 - User Mode Thread Library

Project Goals

- To understand the idea of threads.
- To implement independent, parallel execution within a process.

Collaboration policy

- You are encouraged to discuss this project with your classmates/instructors but are required to turn in your own solution.
- You must be able to fully explain your solution during oral examination.

Deadline

It is due on Friday, October 20, 23:59:59 EDT (no deadline extensions or late submissions).

Project Description

The main deliverable for this project is a basic thread system for Linux. In the lectures, we learned that threads are independent units of execution that run (virtually) in parallel in the address space of a single process. As a result, they share the same heap memory, open files (file descriptors), process identifier, etc. Each thread has its own context, which consists of a set of CPU registers and a stack. The thread subsystem provides a set of library functions that applications may use to create, start and terminate threads, and manipulate them in various ways.
The most well-known and widespread standard that specifies a set of interfaces for multi-threading programming on Unix-style operating systems is called POSIX threads (or *pthreads*). Recall that pthreads merely prescribes the interface of the threading functionality. The implementation then either implement user-mode threads, take advantage of kernel-mode threads (if provided by the operating system), or mix the two approaches. In this project, you will implement a small subset of the pthread API exclusively in user-mode.

In particular, we aim to implement the following three functions from the pthread interface in user mode on Linux (prototypes and explanations partially taken from the respective man pages):

```
int pthread_create( pthread_t *thread,
      const  pthread_attr_t  *attr,
      void *(*start_routine) (void *),
      void *arg);
```

The `pthread_create()` function creates a new thread within a process. Upon successful completion, `pthread_create()` stores the ID of the created thread in the location referenced by `thread`. In our implementation, the second argument (`attr`) shall always be NULL. The thread is created (i.e., your library must create a new Thread context, cf. slides ec440-4) and executes `start_routine` with `arg` as its sole argument. If the *start_routine* returns, the effect shall be as if there was an implicit call to `pthread_exit()` using the return value of `start_routine` as the exit status. Note that the thread in which `main()` was originally invoked differs from this. When it returns from `main()`, the effect shall be as if there was an implicit call to `exit()` using the return value of `main()` as the exit status.

`void pthread_exit(void *value_ptr);`

The `pthread_exit()` function terminates the calling thread. In our current implementation, we ignore the value passed in as the first argument (`value_ptr`) and clean up all information related to the terminating thread. The process shall exit with an exit status of 0 after the last thread has been terminated. The behavior shall be as if the implementation called `exit()` with a zero argument at thread termination time.

`pthread_t pthread_self(void);`

The `pthread_self()` function shall return the thread ID of the calling thread.

For more details about error handling, please refer to the respective man pages.

Submission Guidelines

- The threading library must be implemented in C/C++
- To facilitate grading, you must include the pthreads header file (***#include<pthread.h>***) in your source(s). We will compile your thread library against our test application that will call your pthread functions and check whether threads are properly started and terminated.
- Your *makefile* should compile your source files into an object *threads.o* file.

```
gcc/g++ -m32 -c -o threads.o threads.c
```

- In your home directory create a folder *project2* and place all of your source files, makefile and README there. Switch to the *project2* directory and execute ***submit2***
- A confirmation mail of your submission is sent to your account on ec440.bu.edu. You can read this mail by executing ***mail***.
- In the *README* file explain what you did. If you had problems, tell us why and what.
- You are allowed to resubmit your files. The latest submission before the deadline will be graded

Oral Examination

- Deadline: Friday, October 27th, 18.00 EST
- You are required to meet with one member of the course staff (excluding Prof. Giles) during office hours to explain your solution.