View

GTThreads: A Preemptive User-level Threads Library

The end goal of this project is to write a preemptive user-level threads library with mutex exclusion. Along the way, you will become familiar with the pthreads li and solve the classic mutual exclusion puzzle of "The Dining Philosophers".

Contents

Setup

1. Gain access to a well-maintained linux machine. If you do not have this already, then it is recommended that you follow the instructions for downloading a installing VirtualBox and the AOS Virtual Image.
2. Download gtthreads.tar.gz, which contains the starter code for the project.
3. You can confirm that you have pthreads installed on your linux platform with the command

   ldconfig -p | grep libpthread

Collaboration

For this and all projects, students must work on their own.

Warm-up

To familiarize yourself with the pthreads library, you will debug a simple test program named producer_consumer.c. You may read about the pthread api or cons the man pages on your virtual platform. For example, to learn about pthread_join, type

```
man pthread_join
```

in a terminal.

There are five bugs in total (you may consult the videos for the project to help you find them.) After finding these bugs, provide answers to the questions given in QA_producter_consumer.txt in that file.

Dining Philosophers

A good exercise for understanding the perils of deadlock and how to avoid them is the classic "Dining Philosophers" problem originally posed by Dijtkstra:

> There are five philosophers sitting around a table. Being philosophers, they like to think, but they also have to eat every now an again. Each philosopher gets his own rice bowl, but there only five chopsticks, one laid between each pair of neighboring philosophers. When wishing to eat, a philosopher needs to pick up the two chopsticks on either side of him. When wishing to think, he places the chopsticks back in their places for his neighbors to use.

The file dining.c contains starter code for the problem. Your task is to complete the given program by filling in the code where indicated. The output should read directions for a play, where the action involves philosophers thinking, picking up chopsticks, eating, and putting them down.

The directions for the play must be valid. That is, a philosopher can only pick up a chopstick if it is available and only put one down if he has it. Equally important that no philosopher should be starved. A particularly bad situation is deadlock, where all five philosophers pick up the chopstick to their left and wait for the one

their right to be set down. Of course, this chopstick will never be set down because it is in the left hand of another philosopher. In this deadlocked situation, the philosophers starve. Take care that your program avoids this problem.

There is no requirement to implement fairness among the philosophers, as long as everyone gets to eat when chopsticks are available. For instance, one way to a deadlock would be just to prevent one philosopher from ever picking up chopsticks. Another is to use a single global mutex that only allows one philosopher to e time. Neither of these are good solutions.

GTthreads

The main part of the project is to implement a user-level thread library with an api similar to POSIX threads or pthreads. The specifications for implementing this are found in the two files gtthread_sched.c and gtthread_mutex.c.

The following strategies are highly recommended:

- Use the provided steque data structure (in files steque.[ch]) for both your scheduling queue and your mutex locks.
- Use swapcontext, makecontext, and getcontext functions, which are illustrated in the example program tennis.c, to switch among your threads.
- Use setitimer, sigaction, sigprocmask, and related functions, which are illustrated in the example program defcon.c, to achieve pre-emption among your thr Consult this post and the wikipedia article for more information.

It is also suggested that you work incrementally. For instance, you might the task into the following pieces:

1. Implement and test the library without pre-emption or mutexes and use gtthread_yield only to change threads.
2. Add preemption via the signal handler.
3. Add the mutex capabilities.

Grading

Deliverable	Grade
producer_consumer.c	5%
QA_producer_consumer.txt	5%
dining.c	10%
gtthreads library	80%

This page was last edited on 2014/04/25 08:51:02.