CS 333: Operating Systems Lab
Autumn 2018

Lab #9 finally, pthreads!

Goal

In this lab we will learn about *pthreads*, a user space library for multi-threaded programming and use it to solve some synchronization problems.

References:
`https://computing.llnl.gov/tutorials/pthreads/`
`http://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html`

1. hello pthreads!

In this section, we will use the `pthread` (POSIX threads) library to understand the working of threads, share data across threads and synchronization.

(a) Read the sample program `threads.c` provided. In this program, 100 threads are created which execute a function that updates a shared counter. pthreads are declared using `pthread_t`. To start a thread, use `pthread_create()`. This function takes four arguments: address of the pthread variable representing this thread (pthread ID), attributes of the new thread, start routine (function) and parameters for the start routine. On `pthread_create` a new thread is created with the above arguments and commences execution at the start routine. The main thread waits for the thread to exit and then reaps the thread using `pthread_join()`.

Compile the program using the following command:
`gcc threads.c -lpthread`

Read the following man pages to understand pthreads in more detail:
`pthreads`, `pthread_create`, `pthread_join`, `pthread_detach`, `pthread_exit`, `pthread_kill`, `pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_mutex_trylock`, `pthread_spin_lock`, `pthread_spin_unlock`,
`pthread_cond_signal`, `pthread_cond_wait`, `pthread_cond_broadcast`, `pthread_cond_init`, ...

(b) Next, we will use pthread locks to synchronize access to shared data across threads.

Write a program `threads-with-mutex.c` which synchronizes access to the shared counter in `threads.c` using a mutex lock. You can declare the mutex as a global variable.
The functions of interest are: `pthread_mutex_init()`, `pthread_mutex_lock())`, `pthread_mutex_unlock()`, and `pthread_mutex_destroy()`.

2. argumentative pthreads

The `pthread_create()` function allows a single argument to the start function. The argument has to passed by reference as a void pointer. Think about how to pass multiple arguments to the start function.

Write a program `nlocks.c` that does the following,

- Creates and initializes 10 shared counters and 10 locks.
- The parent process creates 10 threads.
- Each thread, adds 1 to its corresponding data value, a 1000 times. E.g., thread 0 updates data[0].
- The parent also updates the data items in the following manner—adds 1 to data[0], then to data[1] etc. for all 10 data items and then repeats for a total of 1000 iterations.

- With correct synchronization each of the data values should be `2000`.
- Note: The index to be use to process data items by each thread has to carefully passed to the start routine as a parameter.

3. to read or to write, is that the question?

Assume a situation where several threads read and write to a shared data item. Reads do not necessarily need locked access, no modifications on reads! But with parallel reads and writes, locks are required. With a **reader-writer** lock, multiple readers can concurrently access the data. However, a writer must not access the data concurrently when either readers or writers access the data. A writer has to wait till all the readers or writer complete access to the shared data, while a reader does not need to wait if readers are currently active. It would be unfair for a reader to jump in immediately, ahead of a waiting writer. If that happened often enough, writers would starve. As part of this exercise, implement a **reader-writer-locks-with-writer-preference**, where new readers do not access the shared data in case of writers waiting to write, even if readers are active (holding the reader lock).

The following are the semantics of a reader-writer lock with writer preference.

1. With readers already present, a new reader can read only if no writer is already waiting to write.
2. If a writer is waiting to write, writers get preference, readers have to wait till all writers clear out.
3. A writer cannot write if readers are active (holding the reader lock) or another writer is active (holding the writer lock).

Use the sample program `reader-writer.c` as skeleton code to get started. Note that the implementation should use `pthread` conditional variables and mutex.
Check the tar for sample inputs and outputs.

Submission Guidelines

- All submissions via moodle. Name your submission as: **<rollno_lab9>.tar.gz**
- The tar should contain the following files in the following directory structure:

```
<roll_number_lab9>/
|____threads-with-mutex.c
|____nlocks.c
|____reader-writer.c
|__outputs/
|____<outputs of sample runs (exercise 1, 2, & 3)>
```

- We will evaluate your submission by reading through your code and executing it over several test cases.
- **Deadline: Monday, 8^{th} October 2018 - 05:00 PM**.

4. extra sync.

- Implement the producer-consumer using pthread condition variables and mutex locks. Assume a bounded buffer for production.
- Assume two threads, with functions hydrogen and oxygen. Each thread periodically creates a hydrogen or oxygen atom and when two hydrogen atoms and one oxygen atom is available, water is formed. Assume, that incomplete formations, block further partial molecule completions. Implement this using semaphores.