Brief Introduction to pThreads

The IEEE POSIX (Portable Operating System Interface) is a standard for enabling application portability across systems. We will use the pThread API/library to explore multithreaded programs. Threads are a sequence of instructions within a program and can be executed independently of other code. Multithreaded programs are designed to leverage parallelism and increase the performance of the program. In a Master/slave design, a master thread creates a pool of worker thread(s) that are dispatched to accomplish a task (computation, I/O, etc.). Whereas, in the peer design model, the master joins the worker threads to work cooperatively to accomplish a task.

Threads work with local data and shared resources. Access to shared resources must be controlled and coordinated to ensure integrity of data. Some pthreads synchronization primitives include mutex and condition variables. Mutex offer mutual exclusion to shared resources, while condition variables enable threads to coordinate access to shared resources, enabling threads to wait and signal each other.

POSIX threads use mesa, in contrast to hoare, style implementation for signal in condition variables, due to its simplicity. In mesa style implementation, the signaled thread does not immediately become the running thread,it is moved from the pool of blocked threads to the pool of ready threads. Therefore, by the time the signaled thread is selected to run, it is possible that the condition that the thread was waiting for may have changed again. It is important to consider these events when working with condition variables.

This is a brief introduction to pthreads and the POSIX pthread API for thread management and synchronization.

Threads

Thread management

You should know how to create a thread and manage its life cycle. A POSIX thread is called pthread_t (pthread type) and the POSIX API for thread management are pthread_create, pthread_exit, pthread_join.

Key points to remember about thread management API

1. pthread_create(pthread_t * t, pthread_attr_r *a, (void *) start routine, (void *) argument_to_start_routine)
    a. The threads can be created with a set of default attributes using NULL as the second argument to pthread_create in place of *a.
    b. Start_routine must be void *. Void pointers in c enable the declaration of a “generic” function type or data type, primarily for portability.
    c. The forth argument is a pointer to the arguments, if any, of the start_routine. If there are multiple arguments to pass to the start_routine, create a structure to hold all the parameters and pass a pointer as the argument_to_the_start_routine. Ensure data integrity by sending unique instances of arguments to keep the data intact for every thread.
2. phtread_exit should be used to delay thread termination to ensure all threads spawned by the exiting thread finish execution before the thread actually terminates.
3. pthead_join(pthread_t s, void ** return_value) is used to make the calling thread wait for thread s (first argument to pthread_join) to complete execution and retrieve the return value, if any, in return_value. This adds a level of determinism when designing multithreaded programs.

Synchronization primitives

The following important points must be remembered about synchronization primitives in POSIX.

A mutex is simply a lock that can be used to control access to a shared resource. The mutex can be declared and initialized using the macro PTHREAD_MUTEX_INITIALIZER for statically allocated mutexes with default attributes. On the other hand, pthread_mutex_init(pthread_mutex_t *m, pthread_mutexattr_t *a) can be used to initialize a mutex with a set of attributes. To explore the default attributes please refer to the man pages. Remember to use pthread_mutex_unlock to release the mutex otherwise, other threads cannot access the mutex.

Condition variables are declared and initialized similar to mutexes and also have a set of attributes that can be changed using the get/set functions for the attributes. Please refer to the man pages for details about the attributes of condition variables. It is important to remember to use while loops rather than if-statements when setting a thread to wait for a condition variable. This is essential, since mesa style implementation of signals in condition variables, only move the signaled thread from the blocked state to the ready state, nut the running state.

Online references:

1. Posix Programmer's Guide found online here.
2. Alfred Park, Randu tutorial on pthreads found here
3. Blaise Barney tutorial on pThreads is here

Compiling and running your code

1. You must include the pthread header file in your course code
    a. #include <pthread.h>
2. You might also have to inform the linker about the use of the pthread library (-lpthread)
    a. gcc helloThread.c -o name_of_executable -lpthread
    b. Depending on the functions you use, you may also have to include math.h and unistd.h for math and sleep functions respectively and link at compiling time with -lm

Working in the student environment in the University of Waterloo

1. Open a terminal in windows or mac and start a secure session using the ssh tool.
2. At the command line enter `ssh your_watid@linux.student.cs.uwaterloo.ca`
3. Note: you can replace linux with the name of a specific machine. This is useful when you write extensions for OS/161 in future programming assignments, where you will need to open two secure connections to the same machine to debug your code for OS/161.
4. Once you are logged in. You are in a linux environment and can write and compile c programs using gcc compiler.
5. Your linux environment has a default text editor known as vi or vim. You will find many resources online to learn the commands required to use the text editor. Another option is to write the code in a text editor in your local machine and then securely copy (scp) it to your student environment.

Working in MAC

Open a terminal, install xcode if you don't already have it.

To verify you installed correct: *Type* "xcode-select --version" on the Terminal command line

- If installed Correct: **xcode-select version 2384** will appear
- Files are located & accessible at: **Library/Developer/CommandLineTools** in your mac

If you don't have the correct version

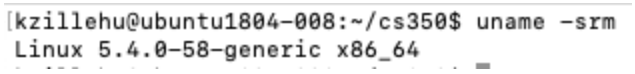
> **Option 1**
> **To install Direct with Terminal** *scroll to* **Finder menu bar**> *click* **Go**> *Select* **Utilities**> Open_ **Terminal.app**> **Type** "xcode-select --install" on Terminal Command Line

References

1. https://developer.apple.com/forums/thread/670389
2. https://linuxize.com/post/how-to-check-the-kernel-version-in-linux/

Working in Linux

1. Sometimes you need to know what version of unix you have, type unam -srn to get details about the linux distribution you have. An example on a mac and in the student environment. Darwin is Apple's Unix-like operating system, compatible with POSIX.



2. Linux shell - sometimes it is important to know which linux shell you are using. Try ps -p $$, where $$ refers to currently running process
3. Sometimes, you may want to change the prompt in your terminal. An example of a temporary change illustrated, where "$" can be any string you would to be your prompt. To make permanent changes to your prompt update your bash file (shown here).

```
kzillehu@ubuntu1804-008:~/cs350/pthreadsExamples$ export PS1="$"
$clear
```

Visualizing your threads

There are tools that enable you to see the load on your machine. Try experimenting with top. In your linux, unix or unix-like environment type top into the command line. You will be displayed all system threads and the various metrics maintained by the system in the columns displayed. Some of the columns are PID (process identification number), memory used, memory allocated, command, etc. Once top has been started,

1. press d to change the rate at which top is refreshed.
2. Press f to change the columns to display. The fields in bold are displayed. Use spacebar to select or deselect for display.
3. Press k to kill process enter the PID of the process you would like to kill and send it SIGKILL(9) or SIGMTERM(15) to terminate process forcefully or graciously.
4. Press q to quit top.

top -H -u kzillehu (to show process from a certain user) //-H for all threads in the system
//-u for a specific user