CSUS, College of Engineering and Computer Science
Department of Computer Science
CSC/CPE 139 – Operating System

Programming Assignment 3 – User-level Threading

Goal: In this project, you will implement a user-level threads package in C.

Acknowledgements: Significant portions of this project have been adapted from a project given by Prof. Urgaonkar of Penn State University, which was further adapted from Prof. Saran of IIT Delhi.

Hardware/Software: You need access to a machine with Intel-based CPU, running UNIX/Linux, and loaded with C-related software (see below). It is tested that virtual machines with Linux environment is OK to run. You can reuse the virtual machine image from the textbook.

Instructions: Here is a list of things to do.

1. Refresh your understanding of linking, compiling, and debugging (e.g., gcc, gdb, make) C programs.

2. Learn about the following:

- *sigsetjmp* and *siglongjmp*. You should carefully read the man pages for these functions. You may choose to use variants/enhanced versions of these functions, if available. Check out the sample file (demo.c), also available in the same folder as this document, for a simple example of context switching in user space.
- How signals are sent/handled by C programs (system calls *signal* () and *kill* ()). Check out a second sample program (timer.c), also available in the same folder as this document, which implements a very simple signal handler. Play with it to reinforce your understanding of the creation and handling of signals.
- Function pointers in C. See below for why they are needed in this assignment.

3. Implement context switching using *sisetjmp* and *silongjmp* as discussed in class. Implement two preemptive scheduling algorithms:

- Round-robin
- Lottery scheduling.

4. Design data structures for thread entities.

5. Implement the following functions:

- int CreateThread (void (*f) (void)) - this function creates a new thread with f() being its entry point. The function returns the created thread id (>= 0) or -1 to indicate failure. It is assumed that f() never returns. A thread can create other threads. The created thread is appended to the end of the ready list (state = READY). Thread ids are consecutive and unique, starting with 0.
- void Go() - This function is called by the main process to start the scheduling of threads. It is assumed that at least one thread is created before Go() is called. This function is called exactly once and it never returns.
- int GetMyId() - This function is called within a running thread. The function returns the thread id of the calling thread.
- void Dispatch(int sig) - the thread scheduler. This function is called by the interval clock interrupt and it schedules threads using FIFO order. It is assumed that at least one thread exists all the time - therefore there is a stack at any time. It is not assumed that the thread is in ready state.
- int GetStatus(int thread_id, status *stat) - this call fills the status structure with thread data. Returns id on success and -1 on failure.
- void SleepThread(int sec) - the calling process will sleep until (current-time + sec). The sleeping time is not considered wait time.
- void CleanUp() - shuts down scheduling, prints relevant statistics, deletes all threads and exits the program.

  Other functions by yourself upon any needs.

6. Finally, implement the following function:

- void *GetThreadResult(int tid) waits till a thread created with the above function returns, and returns the return value of that thread (pardon the alliteration!) tid is the id of thread. This function, obviously, waits until that thread is done with. Use semaphores (we would have revised these in class by the time you get to this part of the implementation) to ensure GetThreadsResult waits for the thread to return.

7. Some important constants to use:

- MAX_NO_OF THREADS 100 /* in any state */
- STACK_SIZE 4096
- TIME_QUANTUM 1*SECOND

Outputs: Some outputs I would like to see:

You will have to think of writing functions that various threads in your demonstration will execute to best bring out the following aspects.

1. Demonstration of the proper working of round robin and lottery scheduling policies.
2. For each of the threads that are created: (a) thread id, (b) state transition sequence (running, ready, sleeping). (c) number of bursts (none if the thread never ran), (d) total execution time in msec (N/A if thread never ran), (e) total requested sleeping time in msec (N/A if thread never slept), (f) average execution time quantum in msec (N/A if thread never ran), (g) average waiting time (status = READY) (N/A if thread never ran).

Deliverable: Submit the following through Canvas, in a **tar-ball**:

1) Source files i.e. .c files, .o file, and a report;
2) The report should contain source code and necessary screenshots. Please submit only an **electronic version** to **Canvas**.