CIS 620 Project 2 Spring 2019

(Due: Mar. 6)

The purpose of this project is to familiarize yourself with the design and implementation issues of a user-level thread package on PC/Linux. You need to use the `tar` command with the options `xvfz` to uncompress and extract files from `~cis620s/pub/xt.tar.gz` to your working directory.

Part I

The thread package which we discussed in the class is non-preemptive. That is, a thread can run to completion unless it yields the control to other threads. For the first part of this assignment, you need to make the threads preemptive. You can use `signal` and `ualarm` to provide a clock interrupt every 0.02 second. When the clock interrupt occurs, the interrupt handler suspends the current running thread and finds a runnable thread to run. On PC/Linux, you need `sigemptyset`, `sigaddset`, and `sigprocmask` to unblock the SIGALRM signal to allow the next SIGALRM delivered. Add more comments to the source code.

Part II

You also need to enhance the thread library with the event mechanism. The events are declared as variables with a data type `xthread_event_t`. The following three functions have to be implemented:

- `void xthread_init_ev(xthread_event_t *e);`
  The event pointed by e is initialized as *NOT_OCCURRED* when `xthread_init_ev()` is invoked.
- `void xthread_wait_ev(xthread_event_t *e) ;`
  The function `xthread_wait_ev()` will place the calling thread on a queue of threads if the event pointed by e has not occurred. Otherwise, the calling thread continues to execute and the event is set to be *NOT_OCCURRED*.
- `void xthread_set_ev(xthread_event_t *e) ;`
  When a thread does a `xthread_set_ev()` operation on the event pointed by e, ALL of the waiting threads are allowed to continue processing and the event will be reset to *NOT_OCCURRED*. If there are no queued processes, the event will be set as *OCCURRED*.

Note that if the SIGALRM occurs during the execution of thread creation/completion, event wait/set, etc., the process table may lead to an inconsistent state (why? Give an example in your report). To solve this problem, you can use

```
usec = ualarm(0,0);
...
ualarm(usec,0);
```

to disable the timer interrupt at the function entrance and restore it before the function returns.