CMPUT379 – Assign 4
Project Report

Brady Pomerleau

Objectives:

To get an introduction to multithreading concepts, including thread syncronization, shared memory usage, and deadlock avoidance.

Design Overview:

- a4tasks
  - a simulator for multiple tasks running as separate threads using shared resources.
    - tasks are implemented as pthread threads in Linux
    - tasks require resources in order to “run”
  - pool of resources is shared among all tasks in simulation
  - tasks and resources are specified in a data file
    - resources are specified by a line of the following format:
      resources [str1:#] [str2:#] [str3:#] ...
      - str*i* is the name of the resource type
      - # is the max number of resources available of that type
      - 0 to NRES_TYPES (= 10) may be specified
    - tasks are specified by a line of the following format:
      task <name> <runTime> <idleTime> [resource1:#] [resource2:#] ...
      - name is the name of the task
      - a resource and # pair specify the number of a certain resource type needed in order for program to run
      - once all of the listed resources are acquired by the task, task waits for time = runTime in millisecsj
      - after runTime elapsed, task releases all held resources and waits idleTime before trying to acquire resources again
    - all fields are separated by one or more space characters (there is no space between colon-separated values

        - blank lines and lines beginning with a # are ignored
        - a maximum of NTASKS = 25 tasks may be specified
    - tasks can be in one of three states:
        - WAIT: task is attempting to aquire the necessary resources for running
        - RUN: task has aquired needed resources and is waiting runTime milliseconds
        - IDLE: task has finished “running”, has released held resources, and is now waiting for idleTime milliseconds
    - program is invoked on the command line as follows:
        - a4tasks inputFile monitorTime NITER
    - a special thread called “monitor” prints the state of each task every “monitorTime” milliseconds
    - each task cycles through the RUN state NITER times, then remains in the IDLE state until program completion
    - the intial thread waits until all tasks have completed the correct number of iterations, then cancels the monitor thread and prints the final state of all of the tasks and resources

Project Status:

All functionality is implemented as described.

Notes:

I only used a single mutex to lock both the resource objects and the task objects because most times both objects were accessed/modified at the same time, and doing so avoided a deadlock hazard (if two mutexes used (one for each object), a deadlock hazard exists if the order of locking is not consistently enforced in the code).

Testing and Results:

The following tests were conducted on university lab computers:

Test	Description	Result
t1.dat	Run t1.dat. Observe that all tasks run correct number of iterations, printing format is correct	success
t2.dat	Run t2.dat. Obeserve that tasks that should not be able to run together (limited resources)	success

	don’t	
t3.dat	Run t3.dat. Obeserve that tasks requesting more resources than exist (or of type that doesn’t exist) never run (program hangs with t6 and t7 in WAIT state)	success
t4.dat	Run t4.dat. Observe that program handles incorrect input format (ignores incomplete lines)	success
t5.dat	Run t5.dat. Observe that program runs without any resources specified	success
t6.dat	Run t6.dat. Observe that extra resource types (beyond NRES_TYPES) are ignored, and that task lines are ignored after NTASKS number of tasks have been specified	success

Acknowledgements:

To complete this assignment I used the following resources:

- Course textbooks:
  - Operating Systems Concepts ver.9
  - Advanced Programming in the Unix Environment ver.3
- various C programming errors troubleshooting accomplished using Google search and forums (primarily stackoverflow.com)