Develop a Pthread like Library - GTTHREADS

Pavan Kulkarni
CS 6210 Project 1

College of Computing, Georgia Tech

1 Platform Used

- Linux
- Developed and Tested on Kubuntu 12.04

2 The Pre-emptive Scheduler

- . The scheduler is a pre-emptive round robin scheduler which provides each of the thread a user specified quantum of time to execute.
- The Main thread is treated just like any other threads and hence it is also queued into the data structure when the *gtthread_init* method is called.
- The new threads that are created are given a context and stored in the thread Queue from where they are picked and executed by the scheduler.
- A signal handler is written for **SIGVTALRM** such that after every quantum period of time the signal is issued and it is handled by the scheduler.
- The scheduler checks for all the conditions in the Queue and then swaps the existing thread with the one that needs to be executed next.
- On *gtthread_yield* the scheduler is invoked by raising SIGVTALRM and hence again the scheduler will take the responsibility of swapping a new process in.

3 How to compile the library and Run it

1. Use the Makefile submitted
2. Run the command *make clean* followed by *make*
3. The library file generated can be compiled as follows
   "gcc -I'directory where the .h files are placed' main.c gtthread.a -o output"
4. ./output

4 Prevent Deadlocks in the Dining Philosopher

- The Dining Philosopher is a problem which exposes both Deadlock and concurrency problem.
- The Race condition problem due to concurrency is solved by enclosing getting and releasing chopsticks between locks.
- The deadlock can be exposed when all philosophers acquire one stick and all wait for another hence locked in a forever wait.
- I solved the deadlock problem by making the philosophers eat only if its left or right is not eating. The philosopher when it enters the EAT section it decrements a counter to less than 0 and now when a new philosopher tries to enter it spins on the condition that the counter should be greater than 0. The *up* function increments this counter again when it is done with the EAT section. So the philosopher spinning on this count is released and it tries to enter the EAT section.
- The spinning variables are made volatile so that they are fetched everytime and not optimized by the compiler.

5 Thoughts on the project

- The setcontext and swapcontext problems were a bit hard to debug on some occasions. As to what exact context was being loaded and problems being encountered.
- A wait on mutex function implementation would have been nice.
- The project really gave me a good insight into how context switch happens between threads and what parameters need to be saved.