What Linux platform do you use?

Linux x86_64 GNU/Linux Ubuntu 14.10

How the preemptive scheduler is implemented.

For implementing GTThreads with an API similar to PThreads, a preemptive round robin scheduler is designed which enqueues new threads in a list. Before any function is called, the gtthread_init(long period) function is called which initializes the signal handler and the timer. It also creates the main thread with thread id as 0 and adds it to the start of the list. Each thread is assigned a time quantum for which it is allowed to run. If the thread uses up its quantum, it is preempted by catching the SIGVTALRM signal, and then the scheduler is called to swap context to the next incomplete/un-exited/un-canceled thread in the list.

For swapping the context, the scheduler first stores the context of the current thread in a temp thread and then uses that to switch context with the next thread it needs to schedule.

How to compile your library and run your program.

To compile the library, the following command should be used -

→ Extract the files from the tar, and run *make*.

→ This should generate the file gtthread.a

→ The we can use the command as given in the project description to run the file:

```
gcc -Wall -pedantic -I{...} -o test1 test1.c gtthread.a
```

where test1.c is the test program to be run with the library file for testing the API.

→ The dining_philosopher runnable object file is generated using the MakeFile.

Else for running the dining philosopher program, please use the command -

```
gcc -Wall -pedantic -I{...} -o dining_philosophers dining_philosophers.c gtthread.a
```

How you prevent deadlocks in your Dining Philosophers solution.

I initially assign which philosopher will need which chopsticks exactly to eat the rice. For e.g. the Philosopher P1 would need Chopstick 1 and Chopstick 2, Philosopher 2 will need Chopstick 2 and Chopstick 3 and likewise. And then to prevent deadlock in the system, each philosopher is only allowed to pick up the lower numbered chopstick first and then the higher numbered chopstick. Basically this implementation would always ensure that at least one Philosopher can eat while the others wait or think.

In my program, I am running the loop for the dining_philosphers problem infinite number of times.

Any thoughts you have on the project, including things that work especially well or which don't work.

Currently my implementation of GTThreads works well for the test cases I could find. But my program doesnt remove/detach a thread thats completed or exited from the list, but I just mark it appropriately so that the scheduler doesn't schedule that thread again. I hope that's fine for the required implementation.