Lab 8 Scheduler Part 2

EECS 678 Staff

libscheduler.h

```c
/** @file libscheduler.h
 */

#ifndef LIBSCHEDULER_H_
#define LIBSCHEDULER_H_

/**
  Constants which represent the different scheduling algorithms
*/
typedef enum {FCFS = 0, SJF, PSJF, PRI, PPRI, RR} scheme_t;

void  scheduler_start_up              (int cores, scheme_t scheme);
int   scheduler_new_job               (int job_number, int time, int running_time, int priority);
int   scheduler_job_finished          (int core_id, int job_number, int time);
int   scheduler_quantum_expired       (int core_id, int time);
float scheduler_average_turnaround_time();
float scheduler_average_waiting_time  ();
float scheduler_average_response_time ();
void  scheduler_clean_up              ();

void  scheduler_show_queue            ();

#endif /* LIBSCHEDULER_H_ */
```

- job_t

libscheduler.h cont.

- _start_up: Initializes the scheduler
- _new_job: Called when a new job arrives
- _job_finished: Called when a job completes execution
- _quantum_expired: When the scheme is set to RR, called when the quantum timer has expired on a core

libscheduler.h cont.

- Useful statistics
  - _average_waiting_time:
  - _average_turnaround_time:
  - _average_response_time:
- _job_t
  - Stores information on any job about to be run including statistics
  - You may need to use a globals for storing your job queue elements

libscheduler.h cont.

- _clean_up: Free any memory associated with your scheduler.
- _show_queue: This function may print out any debugging information you choose. This function will be called by the simulator after every call the simulator makes to your scheduler.

How to model the CPU cores

- You might need an additional simple data structure to model the cores
- Think of cores in terms of what they will be holding or running
- A status of a core will somehow be related to the status of the job that is/was being run on that core

Questions?

Programming Files

- src/libpriqueue/libpriqueue.c : You will be using the libpriqueue.c program written in Lab 7
- src/libscheduler/libscheduler.c : This is the primary file you will be editing
- examples.pl : A perl script of diff runs that tests your program against the 54 test output files. This file will output differences between your program and the examples.
  - Specifically looks at the last seven lines.

Simulation Flow

1. Check the finished job and schedule the new job based on the return value of scheduler_job_finished

- Here scheduling a job will assign the finished_core_id to the new_job_id
- Note this is just scheduling the job not running it.

Simulation Flow cont

2. Check for newly arrived jobs and calls the sceduler_new_job function

3. Check if quantum time has expired for RR policy

4. Execute the scheduled job/jobs at the current discrete time

Scheduler Details

- The events in a time unit will occur in this order
  1. If a job's last unit of execution occurred in the previous time unit, a scheduler_job_finished() call will be made as the first call in the new time unit.
  2. If a job has finished, the quantum timer for the core will be reset. (Therefore, scheduler_quantum_expired() will never be called on a specific core at the same unit that a job has finished.)

Scheduler Details cont.

3. In RR, if the quantum timer has expired, a scheduler_quantum_expired() will be called.

4. If any job arrives at the time unit, the scheduler_new_job() function will be called.

5. Finally, the CPU will execute the active jobs on each core.

Scheduler Rules

- When multiple cores are available to take on a job, the core with the lowest id should take the job
- A job cannot be ran on multiple cores in the same time unit. However, a job may start on one core, get preempted, and continue on a different core.

Scheduler Rules cont.

- In PSJF, if the job has been partially executed, schedule the job based on its remaining time (not the full running time).
- In RR, when a new job arrives, it must be placed at the end of the cycle of jobs. Every existing job must run some amount of time before the new job should run.

Scheduler Rulers cont.

- In all schemes except RR, if two or more jobs are tied (eg: if in PRI multiple jobs have the priority of 1), use the job with the earliest arrival time. In new_job(), we provided the assumption that all jobs will have a unique arrival time. In RR, when a job is unscheduled as a result of the quantum timer expiring, it must always be placed at the end of the queue.

Questions?

PPRI Single Core flow

- A job finished in the last time unit, resulting in a scheduler_job_finished() call to be made to your scheduler. The scheduler returns that job(id=4) should run.
- In this time unit, a new job also arrived. This results in a scheduler_new_job() call to be made to your scheduler. If the new job has greater priority, it will preempt job(j=4), which was scheduled by scheduler_job_finished(). Now job(id=5) is scheduled to run.

PPRI Single Core Flow cont.

- Only after all jobs finished and any new job arrives will the CPU execute the task. In this example, job(id=4) was never run on the CPU when it was scheduled by scheduler_job_finished(). When calculating response time, you should not consider job as responded until it runs a CPU cycle.

Compile and Run

- To compile we have provided a Makefile
- To run the simulator use
  - ./simulator -c <cores> -s <scheme> <input file>
  - ./simulator -c 2 -s fcfs examples/proc1.csv
  - See scheme_t for all schemes in libscheduler.h

Testing

- Use the command
  - pearl examples.pl
- Three workload models (in examples)
  - proc1.csv, proc2.csv, proc3.csv
- Reference output files are given in the examples directory
- Only the last 7 lines of the given output files will be compared