CS 241 Section Week #4
(09/24/09)

Topics This Section

- MP #2
- MP #3
- 5-state Model
- Review of Scheduling
- Problems

MP #2

MP #2

- qsort()
  - Define your own comparison function
  - int compare (const void *e1, const void *e2) {
    return (*(int*)e1 - *(int*)e2); }
- Merge
  - You need to remove duplicates
  - How do you do that?
- Pthreads
  - pthread_create and pthread_join
  - You DO NOT want to do this:

```
for ( …) {
    pthread_create(…);
    …
    pthread_join(…);
}
```

MP #3

MP3 Forward

In MP3, you will add code to a simulator for a CPU scheduler.

- We provide you with the code for the simulator.
  - You don't need to understand this code to understand this MP.
  - You should consider the simulator a 'black box'
- You need to implement these algorithms:
  fcfs, sjf, psjf, pri, ppri, rr#

MP3 Forward

- You need to fill in 3 scheduling functions:
  - new_job()
  - job_finished()
  - quantum_expired()

  Note that these are the only times that the scheduler needs to make a decision!
- A clean_up() function to clean up any memory your program may've allocated
- A show_queue() function to help you debug your program
- You need to create your own job queue

MP3 Forward

- You also need to fill in 3 statistics functions:
  - float average_response_time()
  - float average_wait_time()
  - float average_init_wait_time()

  These are called at the end of the simulation.

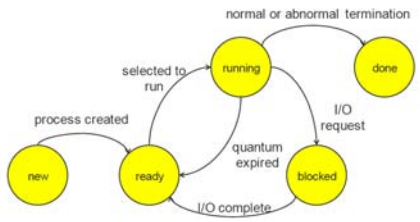
MP3 Forward

- For success on this MP:
  - Carefully read README.txt for details!
  - Look at the example runs and compare your results (e.g. using 'diff')!
- This MP is harder than all previous MPs!!
- Requires a good understanding of data structures, scheduling, and pointers all in one MP!

Good luck!

Five State Model

5-State Model - Transitions



Review of Scheduling

Scheduling

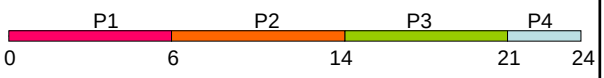
- The CPU Scheduler decides which thread should be in the running state. It is called when:
  - A thread is created or finishes
  - A clock interrupt occurs
  - An I/O interrupt occurs
  - A thread yields

Scheduling

- The algorithms that we usually talk about are:
  - First-Come First-Serve (FCFS)
  - Shortest Job First (SJF)
  - Priority
  - Round Robin (RR)

FCFS Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

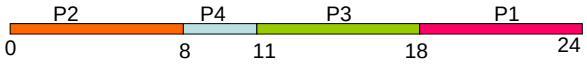


SJF Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

Priority Example

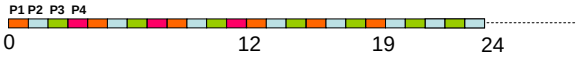
Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0



RR(1) Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

Quanta = 1 time unit



Scheduling

- Scheduling algorithms can be preemptive or non-preemptive
  - Non-preemptive: each thread chooses when to yield to the processor (e.g. when done or system call)
  - Preemptive: scheduler forces the thread to yield (e.g. time quantum expires in RR)

Scheduling

- Metrics for a single job
  - Initial waiting time = time from job submission until it's running
  - Waiting time = total time that the job is not running but queued
  - Response time = time b/t the job's entry and completion

Problems

Problem #1

Job	Duration	Priority #
J1	6	2
J2	4	1
J3	5	1

These three jobs are going to arrive at our scheduler 1 time unit apart from each other (i.e. one job at time 0, one at time 1, and one at time 2), but the order hasn't been decided yet.

Problem #1

We want to guarantee that the jobs finish in the order
J1 then J2 then J3

Problem #1

Which arrival order(s) guarantee this if the scheduler uses:

1) FCFS?
2) non-premptive SJF?
3) preemptive SJF? (use remaining time, and ties are broken by arrival time)
4) RR-1? (arriving jobs are placed on ready queue immediately)
5) non-preemptive priority?
6) preemptive priority?

Problem #1 Solution

Job	Time	Priority	
1		6	2
2		4	1
3		5	1

1. FCFS: 1 2 3

2. n-p-SJF: 1 2 3 (1 needs to arrive first, then 2 will
1 3 2 beat 3)

3. p-SJF: 1 3 2 (1 needs to arrive first, and tie breaks will give it control; then 2 beats 3)

4. RR-1: 1 3 2 (1 needs to run the longest, then 3, then 2, and it barely works out)

5. n-p-Prio: 1 2 3 (1 must come first because of low prio, and then 2 and 3 must follow in order)

6. p-Prio: none (2 or 3 will always preempt 1)

Problem #2

For the SJF and RR examples, calculate:

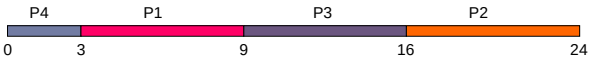
1) Average initial waiting time
2) Average waiting time
3) Average response time

Are either of these clearly better?
When would you use each?

SJF Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

Metrics for Non-preemptive:Shortest Job First



P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

Average initial waiting time (AIWT):
(0+3+9+16)/4 = 7
Average waiting time (AWT):
(0+3+9+16)/4 = 7
Average response time (ART):
(3+9+16+24)/4 = 13

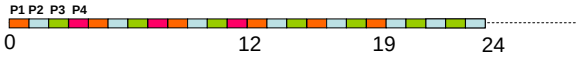
P4 response time: 3
P1 response time: 9
P3 response time: 16
P2 response time: 24

(Stephen Kloder CS241 Spring 2007)

RR(1) Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

Quanta = 1 time unit



Metrics for Round Robin Example

P1 waiting time: 13
P2 waiting time: 16
P3 waiting time: 16
P4 waiting time: 9

Average Initial Waiting time (AIWT):
(0 + 1 + 2 + 3) / 4 = 1.5
Average waiting time (AWT):
(13+16+16+9)/4 = 13.5
Average response time (ART):
(12+24+23+12)/4 = 17.75

P1 response time: 19
P2 response time: 24
P3 response time: 23
P4 response time: 12

(Stephen Kloder CS241 Spring 2007)