Exam Reminders

- We still have a few students that still need to take the conflict.
  - Thanks for not discussing it on Piazza.

- We will discuss only one exam problem today (doesn’t appear on the conflict version of the midterm).

- Grades on Compass 2g on Friday.
MP5

In MP5, 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’
MP5

In MP5, 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: First Come First Serve
  - pri: Priority Scheduling
  - ppri: Preemptive Priority Scheduling
  - sjf: Shortest Job First
  - psjf: Preemptive Shortest Job First (by Remaining Time)
  - rr#: Round Robin
Every modern scheduler uses a priority queue to prioritize what task to run next.

[Part 1] requires you to implement a priority queue library, libpriqueue.
MP5

- libpriqueue contains nine required functions:
  - State-related functions:
    - priqueue_init(), priqueue_destroy()
    - priqueue_size()
libpriqueue contains nine required functions:

- **State-related functions:**
  - priqueue_init(), priqueue_destroy()
  - priqueue_size()

- **Adding and removing elements:**
  - priqueue_offer()
  - priqueue_remove(), priqueue_remove_at()
libpriqueue contains nine required functions:

- **State-related functions:**
  - priqueue_init(), priqueue_destroy()
  - priqueue_size()

- **Adding and removing elements:**
  - priqueue_offer()
  - priqueue_remove(), priqueue_remove_at()

- **Accessing elements:**
  - priqueue.peek(), priqueue.poll()
  - priqueue.at()
The priqueue_init() function takes in a comparer function:

```c
void priqueue_init(
    priqueue_t *q,
    int(*comparer)(const void *, const void *)
)
```

This comparer function is the same function as `qsort()`. It compares two elements, returns an integer if one element is less than, equal to, or greater than the other element.

We’ll look into programming this later.
priqueue_t q;
priqueue_init(&q, comparer);

int i10 = 10, i20 = 20, i30 = 30;
priqueue_offer(&q, &i20);
priqueue_offer(&q, &i30);
priqueue_offer(&q, &i10);

for (i = 0; i < priqueue_size(&q); i++)
    printf("%d ", *((int *)priqueue_at(&q, i)));
printf("\n");
priqueue_destroy(&q);
MP5

priqueue_t q;
priqueue_init(&q, comparer);

int i10 = 10, i20 = 20, i30 = 30;
priqueue_offer(&q, &i20);
priqueue_offer(&q, &i30);
priqueue_offer(&q, &i10);

for (i = 0; i < priqueue_size(&q); i++)
    printf("%d ", *((int *)priqueue_at(&q, i)));
printf("\n");
priqueue_destroy(&q);
MP5

int compare(const void *a, const void *b) {
}

int compare(const void *a, const void *b) {
    int i1 = *((int *)a);
    int i2 = *((int *)b);
}

MP5
int compare(const void *a, const void *b) {
    int i1 = *((int *)a);
    int i2 = *((int *)b);

    if (i1 < i2) return -1;
    else if (i1 == i2) return 0;
    else return 1;
}

// Sample Output:
// 10 20 30
int compare(const void *a, const void *b) {
    int i1 = *((int *)a);
    int i2 = *((int *)b);
    
    if (i1 > i2) return -1;
    else if (i1 == i2) return 0;
    else return 1;
}

// Sample Output:
// 30 20 10
MP5

- You now have a priority queue that can prioritize elements based on any function you program.

- Now, it should be simple to implement a scheduler. In [Part 2], you’ll implement a second library: `libscheduler`.
You need to fill in 3 scheduling functions:
- scheduler_new_job()
- scheduler_job_finished()
- scheduler_quantum_expired()

Note that these are the only times that the scheduler needs to make a decision!

Two helper functions:
- scheduler_start_up()
- scheduler_clean_up()

Called at the beginning and end, for your convenience.
Example Workload:

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Algorithm: FCFS

Cores: 1 core
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()
[t=0]:
   ??
Job Number | Arrival Time | Running Time | Priority |
-------------|--------------|--------------|----------|
0            | 0            | 8            | 1        |
1            | 1            | 8            | 1        |
2            | 3            | 4            | 2        |

[start]: scheduler_start_up()

[t=0]:

_new_job(id=0, time=0, run=8, pri=1)

⇒ _new_job() returns what core the new job should be running on, or -1 if it should not run on a core.
## Task Scheduling Table

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()

[t=0]:

    _new_job(id=0, time=0, run=8, pri=1)

    ➔ returns 0, job(id=0) should run on core(id=0)
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()

[t=0]:
    _new_job(id=0, time=0, run=8, pri=1) = 0

[t=1]: ??
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()

[t=0]:
    _new_job(id=0, time=0, run=8, pri=1) = 0

[t=1]:
    _new_job(id=1, time=1, run=8, pri=1) = ??
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()

[t=0]:
  _new_job(id=0, time=0, run=8, pri=1) = 0

[t=1]:
  _new_job(id=1, time=1, run=8, pri=1) = -1
[start]: scheduler_start_up()
[t=0]:
    _new_job(id=0, time=0, run=8, pri=1) = 0

[t=1]:
    _new_job(id=1, time=1, run=8, pri=1) = -1

[t=2]:
    ??
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[start]: scheduler_start_up()

[t=0]:

  _new_job(id=0, time=0, run=8, pri=1) = 0

[t=1]:

  _new_job(id=1, time=1, run=8, pri=1) = -1

[t=2]:

  (Nothing happens, no calls to your program)
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[t=2]:
(Nothing happens, no calls to your program)

[t=3]:
??
### Job Table

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

**[t=2]:**

*(Nothing happens, no calls to your program)*

**[t=3]:**

```python
_new_job(id=2, time=3, run=4, pri=2) = ??
```
## Table

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[t=2]:

(Nothing happens, no calls to your program)

[t=3]:

```python
_new_job(id=2, time=3, run=4, pri=2) = -1
```
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[t=2]:

(Nothing happens, no calls to your program)

[t=3]:

\_new\_job(id=2, time=3, run=4, pri=2) = -1

[t=??]:

(Next this that happens?)
### Table:

<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[t=3]:

```python
new_job(id=2, time=3, run=4, pri=2) = -1
```

[t=8]:

```python
job_finished(core=0, job=0, time=8)
```

→ `job_finished()` is called when the CPU has ran a job to completion... returns the next job number that should be ran on the core.
<table>
<thead>
<tr>
<th>Job Number</th>
<th>Arrival Time</th>
<th>Running Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

[t=3]:

    __new_job(id=2, time=3, run=4, pri=2) = -1

[t=8]:

    __job_finished(core=0, job=0, time=8) = 1
[t=3]:
  _new_job(id=2, time=3, run=4, pri=2) = -1

[t=8]:
  _job_finished(core=0, job=0, time=8) = 1

[t=16]:
  _job_finished(core=0, job=1, time=16) = 2

[t=20]:
  _job_finished(core=0, job=1, time=20) = -1

[Done with scheduling!]:

You also need to fill in 3 statistics functions:
- float scheduler_average_response_time()
- float scheduler_average_wait_time()
- float scheduler_average_turnaround_time()
These are called at the end of the simulation.

We also provide one function debug-related function: scheduler_show_queue().
- After every call our simulator makes, we’ll call this function and you can print out any debugging information you want.
[Done with scheduling!]:

scheduler_average_waiting_time()

--> returns \( \frac{20}{3} = 6.67 \).

scheduler_average_turnaround_time()

--> returns \( \frac{40}{3} = 13.33 \).

scheduler_average_response_time()

--> returns \( \frac{20}{3} = 6.67 \).

scheduler_clean_up()

[Done!]
MP5

- For success on this MP:
  - We provide queuetest.c, a program to help you test [Part 1] independent of [Part 2].
  - We provide 54 example output files and a program, examples.pl, to run all 54 examples at once and report any errors.

- Requires a good understanding of data structures, scheduling, and pointers all in one MP.

Good luck!
Announcements

- No more class this week for CS 241
  - No sections tomorrow
  - No class on Friday (EOH)

- MP5: Due in 6 days and ~12 hours.
  - Tuesday, March 13, 2012 at 11:59pm

- Look for exam grades on Friday on Compass 2g