# CS 130 Homework 1

□□□

TOTAL POINTS

**90 / 100**

QUESTION 1

## 1 Question 1 15 / 15

✓ **- 0 pts Correct**

**- 5 pts** 20ms not answer

**- 5 pts** 25ms not answer

**- 5 pts** 30ms not answer

QUESTION 2

## 2 Question 2 15 / 15

✓ **+ 5 pts Correct**

**+ 3 pts** Partial Correct

**+ 1 pts** Base

✓ **+ 5 pts Correct**

**+ 3 pts** Partial Correct

**+ 1 pts** Base

✓ **+ 5 pts Correct**

**+ 3 pts** Partial Correct

**+ 1 pts** Base

**+ 0 pts** Wrong Anwser

QUESTION 3

## 3 Question 3 15 / 15

✓ **- 0 pts Correct**

**- 2 pts** did not mention 'it is not possible for both processes to be in their critical section'

**- 2 pts** did not mention 'this assumes P0 run first and requires strict alternation of the two processes'

**- 8 pts** incomplete explanation

QUESTION 4

## Question 4 40 pts

### 4.1 4 (a) 0 / 10

**- 0 pts** Correct

✓ **- 10 pts Wrong**

### 4.2 4 (b) 10 / 10

✓ **- 0 pts Correct**

**- 10 pts** Wrong

### 4.3 4 (c) 10 / 10

✓ **- 0 pts Correct**

**- 10 pts** Wrong

### 4.4 4 (d) 10 / 10

✓ **- 0 pts Correct**

**- 10 pts** Wrong

QUESTION 5

## 5 Question 5 15 / 15

✓ **- 0 pts Correct**

**- 3 pts** Not Generalized

**- 5 pts** Insufficient Proof

**- 7 pts** Proof not correct

**- 15 pts** Wrong

# Operating systems Homework 1

## Problem 1

> [15 points] Consider a system that has two CPUs, each CPU having two threads (hyperthreading). Suppose three programs, P0, P1, and P2, are started with run times of 5, 10 and 20 msec, respectively. How long will it take to complete the execution of these programs? Assume that all three programs are 100% CPU bound, do not block during execution, and do not change CPUs once assigned.

### Answer

- These programs will be executed for 20 msec, or 25 msec, or 30 msec, or 35 msec, depends on how OS schedule them.
    - If P0, P1 on one CPU, P2 on another CPU, then it will cost max(5+10, 20) = 20 msec
    - If P0, P2 on one CPU, P1 on another CPU, then it will cost max(5+20, 10) = 25 msec
    - If P1, P2 on one CPU, P0 on another CPU, then it will cost max(5, 10+20) = 30 msec
    - If P0, P1, P2 all on the same CPU, then it will cost (5+10+20) = 35 msec
- So, it may be executed for 20, 25, 30, or 35 msec.

## Problem 2

> [15 points] The readers and writers problem can be formulated in several ways with regard to which category of threads can be started when. Carefully describe three different variations of the problem, each one favoring (or not favoring) some category of threads. For each variation, specify what happens when a reader or a writer becomes ready to access the database, and what happens when a thread is finished. (Hint: we already saw one variation, which is called “writer preferred” on slide 38th of Lecture 8.)

### Answer

The 3 different variations of this question is `reader preferred`, `writer preferred` and the `symmetric` case.

- Case 1: Writer preferred
    - In this case, for reader, it will wait until no writers are writing or waiting. If no writers are writing or waiting, reader will access the database. which means, reader will wait until all writers are finished. And when an active thread is finished, if `waiting_writer_count > 0`, a writer will start, else (no writers waiting), reader will start.
    - In this case, writer has more priority, so when reader want to access the database, It need to wait until no writer is writing or waiting. And when thread is finished, writers will start priorier than readers.
- Case 2: Reader preferred
    - In this case, for readers, when they want to access the database, they will only check if there is a writer is writing now. If not, they will immediately access the database. And if a writer finished, they

## 1 Question 1 **15 / 15**

✓ **- 0 pts** **Correct**

**- 5 pts** 20ms not answer

**- 5 pts** 25ms not answer

**- 5 pts** 30ms not answer

# Operating systems Homework 1

## Problem 1

> [15 points] Consider a system that has two CPUs, each CPU having two threads (hyperthreading). Suppose three programs, P0, P1, and P2, are started with run times of 5, 10 and 20 msec, respectively. How long will it take to complete the execution of these programs? Assume that all three programs are 100% CPU bound, do not block during execution, and do not change CPUs once assigned.

### Answer

- These programs will be executed for 20 msec, or 25 msec, or 30 msec, or 35 msec, depends on how OS schedule them.
    - If P0, P1 on one CPU, P2 on another CPU, then it will cost max(5+10, 20) = 20 msec
    - If P0, P2 on one CPU, P1 on another CPU, then it will cost max(5+20, 10) = 25 msec
    - If P1, P2 on one CPU, P0 on another CPU, then it will cost max(5, 10+20) = 30 msec
    - If P0, P1, P2 all on the same CPU, then it will cost (5+10+20) = 35 msec
- So, it may be executed for 20, 25, 30, or 35 msec.

## Problem 2

> [15 points] The readers and writers problem can be formulated in several ways with regard to which category of threads can be started when. Carefully describe three different variations of the problem, each one favoring (or not favoring) some category of threads. For each variation, specify what happens when a reader or a writer becomes ready to access the database, and what happens when a thread is finished. (Hint: we already saw one variation, which is called “writer preferred” on slide 38th of Lecture 8.)

### Answer

The 3 different variations of this question is `reader preferred`, `writer preferred` and the `symmetric` case.

- Case 1: Writer preferred
    - In this case, for reader, it will wait until no writers are writing or waiting. If no writers are writing or waiting, reader will access the database. which means, reader will wait until all writers are finished. And when an active thread is finished, if `waiting_writer_count > 0`, a writer will start, else (no writers waiting), reader will start.
    - In this case, writer has more priority, so when reader want to access the database, It need to wait until no writer is writing or waiting. And when thread is finished, writers will start priorier than readers.
- Case 2: Reader preferred
    - In this case, for readers, when they want to access the database, they will only check if there is a writer is writing now. If not, they will immediately access the database. And if a writer finished, they

will wake up all readers, then these readers will access the database, regardless of the waiting writers.
    - In this case, readers has more priority, it will not consider about the waiting writers.

- Case 3: Symmetric
    - This case is a combination of 1 and 2. for readers, when they want to access the database, they will only check if there is a writer is writing now. If not, they will immediately access the database(same as case 2). But for writers, when a writer finished, it will check if there is another writer waiting, if there is, it will let the writer start(same as case 1).
    - In this case, readers and writers have same priority. If readers are now reading, It will keep other readers reading until no readers. But if writer is now writing, it will keep writing until no other writers.

# Problem 3

> [15 points] Consider the following solution to the mutual-exclusion problem involving two processes P0 and P1. Assume that the variable turn is initialized to 0. Process P0's code is resented below.

```
/* Other code */
while (turn != 0) { } /* Do nothing and wait. */
Critical Section /* . . . */
turn = 0;
/* Other code */
```

> For process P1, replace 0 by 1 in above code. Determine if the solution meets all the required conditions for a correct mutual-exclusion solution.

## Answer

- This solution satisfies the mutual exclusion requirements.
    - If `turn == 0`, then P0 will do the critical section, and P1 should wait until `turn == 1`
    - If `turn == 1`, then P1 will do the critical section, and P0 should wait until `turn == 0`
    - So, P0 and P1 will not into the critical section at the same time, which satisfies mutual-exclusion.
- For the initial state `turn = 0`, P0 will run first in its critical section, and P1 will wait.
- But it needs these two threads must run alternatively, and P0 should run first. Otherwise, if P1 runs first, and not alternatively, P1 will occupied the CPU, and always wait, it will cause a infinite loop, and P0 has no chance to run, and will also cause bad sychronization.

# Problem 4

> [40 points] Five batch jobs. A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead. (a) Round robin. (b) Priority scheduling. (c) First-come, first-served (run in order 10, 6, 2, 4, 8). (d) Shortest job first.

## 2 Question 2 **15 / 15**

- ✓ **+ 5 pts** **Correct**
- **+ 3 pts** Partial Correct
- **+ 1 pts** Base
- ✓ **+ 5 pts** **Correct**
- **+ 3 pts** Partial Correct
- **+ 1 pts** Base
- ✓ **+ 5 pts** **Correct**
- **+ 3 pts** Partial Correct
- **+ 1 pts** Base
- **+ 0 pts** Wrong Anwser

will wake up all readers, then these readers will access the database, regardless of the waiting writers.
    - In this case, readers has more priority, it will not consider about the waiting writers.

- Case 3: Symmetric
    - This case is a combination of 1 and 2. for readers, when they want to access the database, they will only check if there is a writer is writing now. If not, they will immediately access the database(same as case 2). But for writers, when a writer finished, it will check if there is another writer waiting, if there is, it will let the writer start(same as case 1).
    - In this case, readers and writers have same priority. If readers are now reading, It will keep other readers reading until no readers. But if writer is now writing, it will keep writing until no other writers.

# Problem 3

> [15 points] Consider the following solution to the mutual-exclusion problem involving two processes P0 and P1. Assume that the variable turn is initialized to 0. Process P0's code is resented below.

```
/* Other code */
while (turn != 0) { } /* Do nothing and wait. */
Critical Section /* . . . */
turn = 0;
/* Other code */
```

> For process P1, replace 0 by 1 in above code. Determine if the solution meets all the required conditions for a correct mutual-exclusion solution.

## Answer

- This solution satisfies the mutual exclusion requirements.
    - If `turn == 0`, then P0 will do the critical section, and P1 should wait until `turn == 1`
    - If `turn == 1`, then P1 will do the critical section, and P0 should wait until `turn == 0`
    - So, P0 and P1 will not into the critical section at the same time, which satisfies mutual-exclusion.
- For the initial state `turn = 0`, P0 will run first in its critical section, and P1 will wait.
- But it needs these two threads must run alternatively, and P0 should run first. Otherwise, if P1 runs first, and not alternatively, P1 will occupied the CPU, and always wait, it will cause a infinite loop, and P0 has no chance to run, and will also cause bad sychronization.

# Problem 4

> [40 points] Five batch jobs. A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead. (a) Round robin. (b) Priority scheduling. (c) First-come, first-served (run in order 10, 6, 2, 4, 8). (d) Shortest job first.

## 3 Question 3 **15 / 15**

✓ **- 0 pts** **Correct**

**- 2 pts** did not mention 'it is not possible for both processes to be in their critical section'

**- 2 pts** did not mention 'this assumes P0 run first and requires strict alternation of the two processes'

**- 8 pts** incomplete explanation

## Answer

- Generally, we assume that all jobs arrive in time `t = 0`, and a time slice that one thread can hold is 1s.
- Round robin:
  - The execution sequence is:

```
t       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
thread  A B C D E A B C D E  A  B  D  E  A  B  D  E  A  B  E  A  B

t       24 25 26 27 28 29 30
thread  E  A  E  A  E  A  A
```

  - mean_process_turnaround = (30 + 23 + 8 + 17 + 28) / 5 = 21.2s
- Priority scheduling:
  - The execution sequence is:

```
t       1–6    7–14    15–24    25–26    27–30
thread   B      E        A        C        D
```

  - mean_process_turnaround = (24 + 6 + 26 + 30 + 14) / 5 = 20s
- First-come, first-served (run in order 10, 6, 2, 4, 8):
  - The execution sequence is:

```
t       1–10    11–16    17–18    19–22    23–30
thread   A        B        C        D        E
```

  - mean_process_turnaround = (10 + 16 + 18 + 22 + 30) / 5 = 19.2s
- Shortest job first:
  - The execution sequence is:

```
t        1–2    3–6    7–12    13–20    21–30
thread    C      D      B        E        A
```

  - mean_process_turnaround = (2 + 6 + 12 + 20 + 30) / 5 = 14s

4.1 4 (a) **0 / 10**

- **0 pts** Correct

✓ **- 10 pts** **Wrong**

## Answer

- Generally, we assume that all jobs arrive in time `t = 0`, and a time slice that one thread can hold is 1s.
- Round robin:
  - The execution sequence is:

```
t       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
thread  A B C D E A B C D E  A  B  D  E  A  B  D  E  A  B  E  A  B

t       24 25 26 27 28 29 30
thread  E  A  E  A  E  A  A
```

  - mean_process_turnaround = (30 + 23 + 8 + 17 + 28) / 5 = 21.2s
- Priority scheduling:
  - The execution sequence is:

```
t       1–6    7–14    15–24    25–26    27–30
thread   B      E        A        C        D
```

  - mean_process_turnaround = (24 + 6 + 26 + 30 + 14) / 5 = 20s
- First-come, first-served (run in order 10, 6, 2, 4, 8):
  - The execution sequence is:

```
t       1–10    11–16    17–18    19–22    23–30
thread   A        B        C        D        E
```

  - mean_process_turnaround = (10 + 16 + 18 + 22 + 30) / 5 = 19.2s
- Shortest job first:
  - The execution sequence is:

```
t        1–2    3–6    7–12    13–20    21–30
thread    C      D      B        E        A
```

  - mean_process_turnaround = (2 + 6 + 12 + 20 + 30) / 5 = 14s

4.2 4 (b) **10 / 10**

✓ **- 0 pts** **Correct**

**- 10 pts** Wrong

## Answer

- Generally, we assume that all jobs arrive in time `t = 0`, and a time slice that one thread can hold is 1s.
- Round robin:
  - The execution sequence is:

```
t       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
thread  A B C D E A B C D E  A  B  D  E  A  B  D  E  A  B  E  A  B

t       24 25 26 27 28 29 30
thread  E  A  E  A  E  A  A
```

  - mean_process_turnaround = (30 + 23 + 8 + 17 + 28) / 5 = 21.2s
- Priority scheduling:
  - The execution sequence is:

```
t       1–6    7–14    15–24    25–26    27–30
thread   B      E        A        C        D
```

  - mean_process_turnaround = (24 + 6 + 26 + 30 + 14) / 5 = 20s
- First-come, first-served (run in order 10, 6, 2, 4, 8):
  - The execution sequence is:

```
t       1–10    11–16    17–18    19–22    23–30
thread   A        B        C        D        E
```

  - mean_process_turnaround = (10 + 16 + 18 + 22 + 30) / 5 = 19.2s
- Shortest job first:
  - The execution sequence is:

```
t        1–2    3–6    7–12    13–20    21–30
thread    C      D      B        E        A
```

  - mean_process_turnaround = (2 + 6 + 12 + 20 + 30) / 5 = 14s

4.3 4 (c) **10 / 10**

✓ **- 0 pts** **Correct**

**- 10 pts** Wrong

## Answer

- Generally, we assume that all jobs arrive in time `t = 0`, and a time slice that one thread can hold is 1s.
- Round robin:
  - The execution sequence is:

```
t       1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
thread  A B C D E A B C D E  A  B  D  E  A  B  D  E  A  B  E  A  B

t       24 25 26 27 28 29 30
thread  E  A  E  A  E  A  A
```

  - mean_process_turnaround = (30 + 23 + 8 + 17 + 28) / 5 = 21.2s
- Priority scheduling:
  - The execution sequence is:

```
t       1–6    7–14    15–24    25–26    27–30
thread   B      E        A        C        D
```

  - mean_process_turnaround = (24 + 6 + 26 + 30 + 14) / 5 = 20s
- First-come, first-served (run in order 10, 6, 2, 4, 8):
  - The execution sequence is:

```
t       1–10    11–16    17–18    19–22    23–30
thread   A        B        C        D        E
```

  - mean_process_turnaround = (10 + 16 + 18 + 22 + 30) / 5 = 19.2s
- Shortest job first:
  - The execution sequence is:

```
t        1–2    3–6    7–12    13–20    21–30
thread    C      D      B        E        A
```

  - mean_process_turnaround = (2 + 6 + 12 + 20 + 30) / 5 = 14s

4.4 4 (d) **10 / 10**

✓ **- 0 pts** **Correct**

**- 10 pts** Wrong

# Problem 5

> [15 points] In the dining philosophers problem, let the following protocol be used: An evennumbered philosopher always picks up his left fork before picking up his right fork; an odd-numbered philosopher always picks up his right fork before picking up his left fork. Will this protocol guarantee deadlock-free operation?

## Answer

- That protocol can guarantee deadlock-free.
- We can consider the simplest case: only 2 philosophers.

```
            / fork1 \
philosopher1         philosopher2
            \ fork2 /
```

  - If p1 go first, he will pick up fork2. Now, p2 notice that fork2 is missing, so he will wait until fork2 is available; while p1 will get fork1 and 2. Then p1 will eating, after he finished, p2 will get fork1 and 2 and then eating.
  - If p2 go first, he will pick up fork1. Now, p1 notice that fork1 is missing, so he will wait until fork1 is available; while p2 will get fork1 and 2. Then p2 will eating, after he finished, p1 will get fork1 and 2 and then eating.
- For more complicated cases, it will also guarantee that there are always >= 1 free forks, and at least one philosopher will hold 2 forks and be able to eat. No such case that each philosopher held only one fork, and cause deadlock.
- So, There will be no deadlocks.

## 5 Question 5 **15 / 15**

✓ **- 0 pts Correct**

**- 3 pts** Not Generalized

**- 5 pts** Insufficient Proof

**- 7 pts** Proof not correct

**- 15 pts** Wrong