Department of Computer Science BOSTON UNIVERSITY

CS-350 - Fundamentals of Computing Systems

Midterm Exam #2
Fall 2023

Name: Jue Hong Lee

BU Username: jhonglee BU ID: U21565203

NOTE: *Please use only the provided space and the included extra page and/or grids to answer the questions. If you do use the extra page/grids, make sure you reference them properly in your solutions. The very last page can be detached for your convenience.*

Problem #1:	/14
Problem #2:	/23
Problem #3:	/22
Problem #4:	/26
Problem #5:	/20
Total Score:	80 → 83 /105

Remarks:

- This is a closed-book/notes exam.
- Basic calculators are allowed.
- You have 80 minutes to complete your exam.
- There are 105 total points.
- If your score is more than 100, it is capped at 100.
- Show your work for full marks.
- Problems and sub-problems weighted as shown.
- **Explain all your assumptions clearly.**

Geonho

Problem 1

Label each of the statements below with either **True (T)** or **False (F)**:

	Statement	T/F
a.	Consider a task continuously performing disk transactions targeting the same cylinder ID. Such a task might cause a different task that is also using the disk to starve if the scheduler employed by the disk is CFQ.	F
b.	There exists a taskset that is not schedulable under Global EDF but that is schedulable under EDF-FF.	T
c.	The capacity of a complex system that can be modeled as a network of queues can be much higher than capacity of the bottleneck.	T
d.	Guaranteeing mutual exclusion among N tasks sharing a data structure is important mainly to maximize good average performance. outcome	F
e.	Shortest Remaining Time (SRT) scheduling achieves freedom from starvation by prioritizing jobs that are approaching their completion.	F
f.	It is reasonable to expect that even a complex scheduling policy will exhibit an overhead comparable to FIFO scheduling when the system is operating at low utilization.	T
g.	Consider two stateless (CPU) resources A and B with the same exact parameters and subject to the same workload. We expect the steady-state utilization of A to be lower than that of B if A employs SJN scheduling while B employs FIFO scheduling.	F
h.	A taskset is scheduled on a single-processor system using a generic algorithm *ALG*. If *ALG* guarantees that all the jobs have a worst-case response time (WCRT) lower than their respective relative deadlines, then the same taskset is schedulable under EDF. 1	T
i.	When predicting the length of future jobs using Exponentially Weighted Moving Averages (EWMA), increasing the parameter α results in discarding older observations faster.	T
j.	Under the same workload, switching to a better scheduling policy for a stateful resource can lower the steady-state utilization of said resource. T overhead T	T

Note: There are 10 questions. A correct answer will get you 2 points; an incorrect answer -1 points; a blank answer 0 points. The final score is capped at 14.

$7 \quad 14 - 3 = 11$

Problem 2

A multi-threaded server uses multiple worker threads (code omitted) that process requests from a shared queue `the_queue`. Worker threads use the `get_from_queue(...)` procedure to fetch the next request to process from the shared queue. When handling a request, the worker thread busywaits for the length of the request, and then responds to the client with the same request ID.
A parent thread (code omitted) is responsible for adding requests into the queue. The requests are added to the queue in no particular order, but will be sorted upon dequeue inside the `get_from_queue(...)` procedure reported below.
The `recompact_queue(struct queue * the_queue, uint32_t pos)` function (omitted) simply ensures that the queue is recompacted after removing the element in position `pos`.

```c
struct request {
   uint64_t req_id; /* Integer request ID set by the client */
   struct timespec req_timestamp; /* time when request sent by client  */
   struct timespec req_length; /* time length of the request */
};

struct request_meta {
   struct request request;
   struct timespec receipt_timestamp; /* time when request enqueued */
   struct timespec start_timestamp;   /* time when request starts service */
   struct timespec completion_timestamp; /* time when request completed */
};

struct queue {
   size_t wr_pos; /* Last item in the queue  */
   size_t rd_pos; /* First item in the queue  */
   size_t max_size; /* Maximum size of the queue  */
   size_t available; /* Number of available spots in the queue  */
   struct request_meta * requests; /* Array of queued requests  */
};

/* Add a new request <request> to the shared queue <the_queue> */
struct request_meta get_from_queue(struct queue * the_queue)
{
   struct request_meta retval;
   uint32_t ret_pos;

   sem_wait(queue_notify);
   sem_wait(queue_mutex);

   ret_pos = find_pop_position(the_queue); /* Discussed in the questions below! */
   retval = the_queue->requests[ret_pos];
   recompact_queue(the_queue, ret_pos);

   sem_post(queue_mutex);
   return retval;
}
```

Listing 1: Worker structure and dequeue operation

(a) [5 points] Consider the case in which the `find_pop_position(...)` is implemented as follows. What is the queue policy employed by the server? Motivate your answer.

```
1  uint32_t find_pop_position(struct queue * the_queue)
2  {
3     struct request_meta * requests = the_queue->requests;
4     uint32_t cur_pos = the_queue->rd_pos;
5     uint32_t to_scan = the_queue->max_size - the_queue->available;
6
7     uint32_t min_pos = cur_pos;
8     double min_arr = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_timestamp);
9
10    while (to_scan-- > 0) {
11       double cur_arr = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_timestamp);
12       if (cur_arr < min_arr) {
13          min_arr = cur_arr;
14          min_pos = cur_pos;
15       }
16       cur_pos = (cur_pos + 1) % the_queue->max_size;
17    }
18    return min_pos;
19 }
```

Listing 2: `find_pop_position(...)` implementation #1.

This is First come, First serve queue policy
It checks req_time stamp which check the time
when the server requested the request.

(b) [6 points] Consider the case in which the `find_pop_position(...)` is implemented as follows. What is the queue policy employed by the server? Motivate your answer.

```
1  uint32_t find_pop_position(struct queue * the_queue)
2  {
3     struct request_meta * requests = the_queue->requests;
4     uint32_t cur_pos = the_queue->rd_pos;
5     uint32_t to_scan = the_queue->max_size - the_queue->available;
6
7     uint32_t min_pos = cur_pos;
8     double min_len = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_length);
9
10    while (to_scan-- > 0) {
11       double cur_len = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_length);
12       if (cur_len < min_len) {
13          min_len = cur_len;
14          min_pos = cur_pos;
15       }
16       cur_pos = (cur_pos + 1) % the_queue->max_size;
17    }
18    return min_pos;
19 }
```

Listing 3: `find_pop_position(...)` implementation #2.

It is SJN. since in the while loop. It scan to find the min reqest time stamp, trly to find the minimum time stamp

(c) [7 points] Consider the case in which the `find_pop_position(...)` is implemented as follows. What is the queue policy employed by the server? Motivate your answer.

```
1  uint32_t find_pop_position(struct queue * the_queue)
2  {
3     struct request_meta * requests = the_queue->requests;
4     uint32_t cur_pos = the_queue->rd_pos;
5     uint32_t to_scan = the_queue->max_size - the_queue->available;
6
7    struct timespec ts_now;
8     clock_gettime(CLOCK_MONOTONIC, &ts_now);
9     double now = TSPEC_TO_DOUBLE(ts_now);
10
11    double arr = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_timestamp);
12    double len = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_length);
13
14    uint32_t max_pos = cur_pos;
15    double max_frac = ((now - arr) + len) / len;
16
17    while (to_scan-- > 0) {
18       double cur_frac;
19       arr = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_timestamp);
20       len = TSPEC_TO_DOUBLE(requests[cur_pos].request.req_length);
21       cur_frac = ((now - arr) + len) / len;
22
23       if (cur_frac > max_frac) {
24          max_frac = cur_frac;
25          max_pos = cur_pos;
26       }
27       cur_pos = (cur_pos + 1) % the_queue->max_size;
28    }
29    return max_pos;
30 }
```

Listing 4: `find_pop_position(...)` implementation #3.

This is HSN, Highest Slowdown Next
look at the code line 15 and 21. It is the
function to find Slow down which $\frac{\text{Now} + \text{len} - \text{arr}}{\text{len}}$
so find the highest slow down and do that
Next

(d) [5 points] Take a look at the printout produced by the server with 1 worker thread (T0) for these 5 requests. The format is the following, where (1) `<sent ts>`, (2) `<receipt ts>`, (3) `<start ts>`, and (4) `<completion ts>` are the timestamps at which the request was (1) sent by the client, (2) recevied by the parent thread, (3) started being worked on, and (4) completed by the worker thread. And where `<req. length>` is the length of the request as set by the client.

```
T0 R<request ID>:<sent ts>,<req. length>,<receipt ts>,<start ts>,<completion ts>
```

Server printout:

```
T0 R0:7513.000000,6.000000,7513.000000,7513.000000,7519.000000
Q:[R1,R2,R3,R4]
T0 R4:7517.000000,1.000000,7517.000000,7519.000000,7520.000000
Q:[R1,R2,R3]
T0 R3:7515.000000,1.500000,7515.000000,7520.000000,7521.500000
Q:[R1,R2]
T0 R2:7514.000000,3.000000,7514.000000,7521.500000,7524.500000
Q:[R1]
T0 R1:7514.000000,4.000000,7514.000000,7524.500000,7528.500000
Q:[]
```

Of the three possible implementations of the `find_pop_position(...)` function described in the previous questions, which one is the server currently employing? Motivate your answer.

look at the sent time stamp, 2nd handled request is later than 3rd handle request so it is not first come first served 1), Also looking at the job length it is R0 would start first since all of the requests would arrive later, then the job length is 1, 1.5, 3, 4 which is shortest to longest so it is SJN 2nd one.

Problem 3

Requests at an image processing system always arrive at a first machine called the decoder (D) which parses the request's payload. At this stage, the system also checks if the response is already available in cache. If the response is not cached (cache miss), the request is sent from D to the image processing engine (I). After that, the request is sent from P to the cache machine (C) so that the result produced by I can be added to the cache. On the other hand, in case D detects a cache hit, the request is directly sent from D to C for fast response retrieval, essentially bypassing I. Cache hits at D occur with probability p_{hit}. Regardless of how a request reaches the C machine, after being served at C, the request might require additional processing. This happens with probability 0.6, in which case it will go back to the decoder (D) as if it was a new request. Otherwise, the request leaves the system and it is considered completed.
All the machines are single-processor systems. You have observed that the system is able to reach steady-state and that the utilization of the P machine is 87%, while the average request length at P is 30 milliseconds. The average service time at D is 17 milliseconds and 11 milliseconds at C.

(a) [5 points] Provide a diagram of the system which shows the inter-connections between the D, P, and C machines. Describe the queuing models used for each machine, the known parameters, and the assumptions you will employ to solve the system.

(b) [5 points] Assume that the probability p_{hit} of a cache hit (a.k.a. hit rate) is 45%. What is the throughput of the entire system?

(c) [6 points] Continue to assume that p_{hit} is 45%. What is the capacity of the system? Motivate your answer.

(d) [6 points] Is it true that a worse caching heuristic—i.e., one that leads to a decreased value of p_{hit}—might cause the system to be unable to reach steady state? If so, what is the threshold for p_{hit} under which the system is unable to reach steady state?

Problem 4

The ITER Tokamak nuclear fusion reactor needs to maintain stable plasma within the reactor's vacuum vessel where hydrogen atoms are fused to create clean energy! To do this, each section of the vacuum vessel is surrounded by an electromagnetic coil that needs to be continuosly controlled by a dedicated Plasma Control System (PCS). You are in charge of picking a processor to deploy the PCS software. The key software modules of the PCS are 4 periodic tasks: 1. Plasma Horizontal Stabilization (HS) which must run 50 times per second, 2. Plasma Vertical Stabilization (VS) which has a period of 13 ms, 3. Plasma Current Calculation (CC) which needs to run every 11 ms, and 4. Plasma Anomaly Detection (AD) with a period of 36 ms. The ITER is a very expensive one-of-a-kind machine and failure to keep the plasma stable inside the vacuum vessel can seriously damage the reactor. R-T

(a) [4 points] You are considering a high-performance single-processor machine on which you have measured the runtime of the tasks. For HS, it is between 3 ms and 4 ms. For VS it is 1±0.5 ms. For CC it falls in the range [0.5, 3] ms. Finally, for AD the runtime is between 4 ms and 7.5 ms. Complete the missing task parameters in Table 1.

Task ID	WCET (ms)	Period (ms)	Utilization (%)
HS	4ms	20ms	20% (0.20)
VS	1.5ms	13ms	12% (0.12)
CC	3ms	11ms	27% (0.27)
AD	7.5ms	36ms	20% (0.2)

50 / 1

50/1sec × 1sec/1000msec

Table 1: Task Parameters.

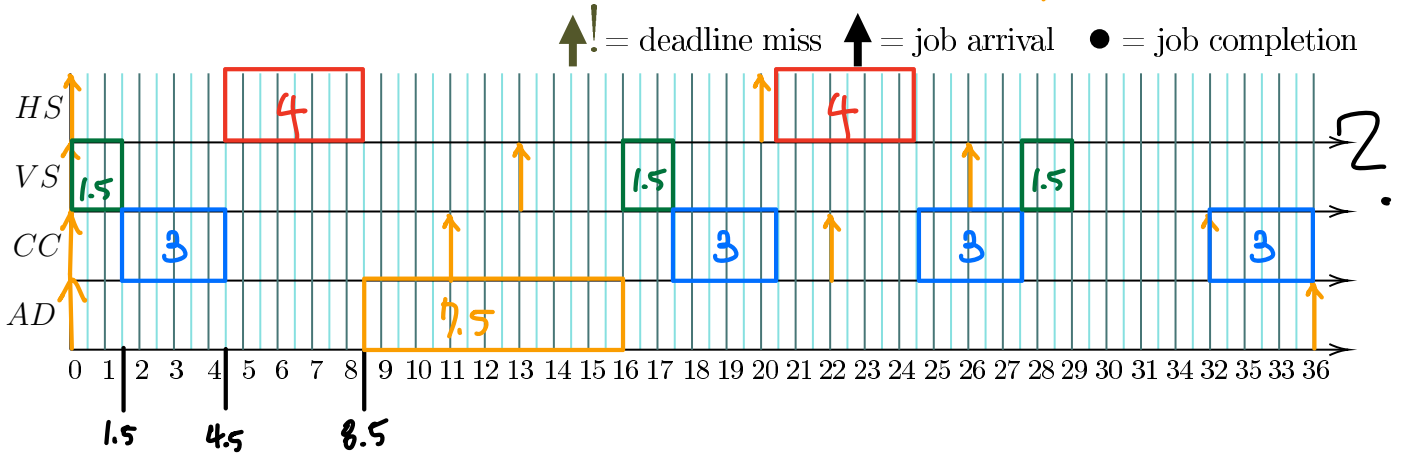
(b) [6 points] Is the system schedulable using Shortest Job Next? Motivate your answer. Use the grid provided below only if needed and use only what you see up to time 36 to conclude about schedulability.

RM

0.20+0.12+0.20+0.2 = 0.796

$n(2^{\frac{1}{n}}-1) = 0.75$ Not concluded.

Schedulable



2.

(c) **[8 points]** Is the system schedulable using Rate Monotonic? Motivate your answer. Use the grid provided below only if needed and use only what you see up to time 36 to conclude about schedulability.

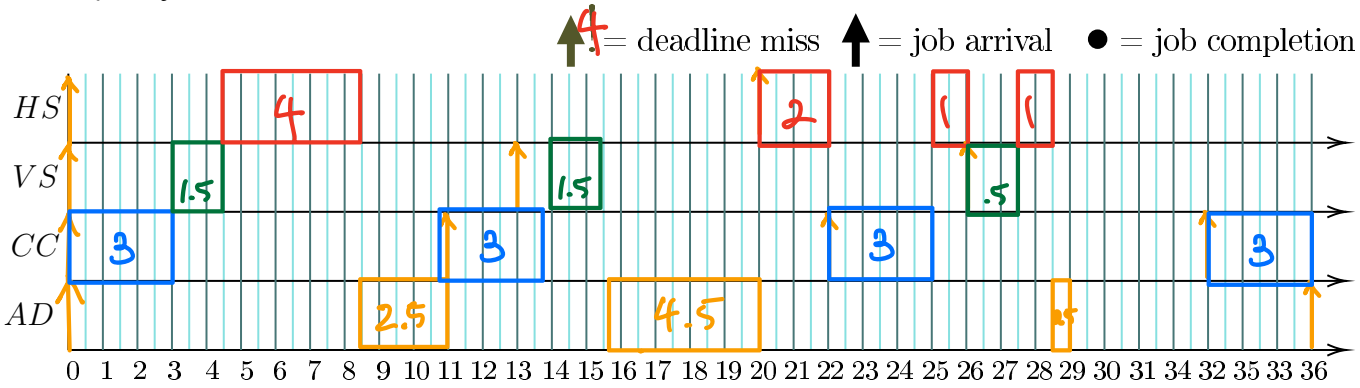
4 20
1.5 13
3 11
7.5 36

P

RM ↔ Deadline 2.

0.20+0.12+0.20+0.2 = 0.796

$m(2^{\frac{1}{m}}-1) = 0.75$ Not concluded



(d) **[8 points]** In an attempt to reduce overall power consumption, you decide to investigate the possibility of using a slower 2-processors system instead of the original machine. The clock frequency is exactly half compared to the original machine, meaning that all the runtimes are exactly twice as long. Is the system schedulable if we apply RM-FF with the tasks in the following order: CC, HS, VS, AD?

Schedulable

HS 8 20 0.4
VS 3 13 0.23
CC 6 11 0.54
AD 15 36 0.41

1.5928

$2(\sqrt{2}-1) = 0.828$

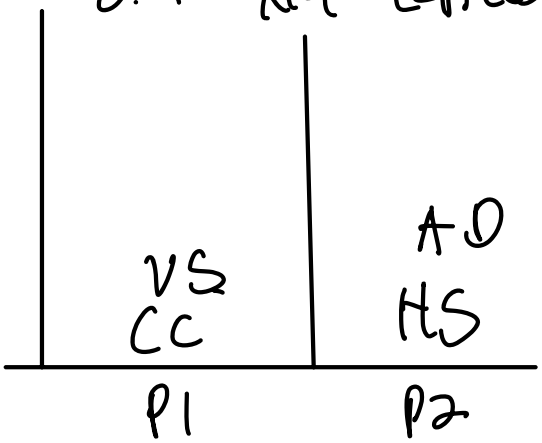
0.69 Not Conclusive

0.1176

0.94545

$2(2^{\frac{1}{2}}-1)$

0.82



Problem 5

A colleague of yours has just discovered the power of multi-threading. They have implemented the following concurrent threads (T1 and T2) to solve some mysterious problem of theirs. The two threads are executed concurrently on a single-processor system with an unknown scheduler.

```
/* Shared Variables - START */
uint32_t result = 1;
uint8_t  lock = 0;
/* Shared Variables - END */
```

Listing 5: Global and shared variables definition.

```
Thread T1:
   int thread1_main (void) {
      [T1 ENTRY SECTION]
      /* Critical section begin */
      if (result == 1) {
         int loc = result + 10;
         result = loc;
      }
      /* Critical section end */
      [EXIT SECTION]

      printf("Result = %d\n", result);
   }
```

Listing 6: Code of T1.

```
Thread T2:
   int thread2_main (void) {
      [T2 ENTRY SECTION]
      /* Critical section begin */
      if (result == 1) {
         int loc = result * 4;
         result = loc;
      }
      /* Critical section end */
      [EXIT SECTION]
   }
   //
   //
```

Listing 7: Code of T2.

(a) [5 points] Initially, your colleague has not implemented the `[T1 ENTRY SECTION]`, `[T2 ENTRY SECTION]`, and `[EXIT SECTION]`. You can consider all of them empty. Is it possible for the code to print the output "`Result = 4`"? If so, show the interleaving of code execution that leads to that output using the table below.

Step	Thread T1	Thread T2
1		if (result == 1) {
2		int loc = result * 4
3		result = loc
4	if (result == 1) {	
5	printf("Result = %d\n", result);	
6		
7		

(b) [7 points] With the same implementation as before, is it possible for the code to produce the output: "`Result = 44`"? If so, show the interleaving of code execution that leads to that output using the table below.

Step	Thread T1	Thread T2
1	if (result == 1) {	
2		if (result == 1) {
3	int loc = result + 10	
4	result = loc	
5		int loc = result * 4
6		result = loc
7	printf ("Result = %d\n", result);	

(c) [8 points] After running the code many times and achieving very inconsistent results, your colleague has finally decided to implement their own protocol for mutual exclusion. Listing 8 reports the implementation for `[T1 ENTRY SECTION]`; Listing 9 reports the implementation for `[T2 ENTRY SECTION]`; and Listing 10 reports the implementation for the common `[EXIT SECTION]`;

It seems however that sometimes thread T2 remains stuck. Can you show in the provided table a possible execution interleaving that illustrates the issue? *NOTE: No need to use all the rows in the table if you can illustrate the problem with less than 12 rows.*

```
1 lock = 1;
2 while (true) {
3    if (lock == 1)
4       break;
5 }
```

Listing 8: T1 Entry Section

```
1 lock = 2;
2 while (true) {
3    if (lock == 2)
4       break;
5 }
```

Listing 9: T2 Entry Section

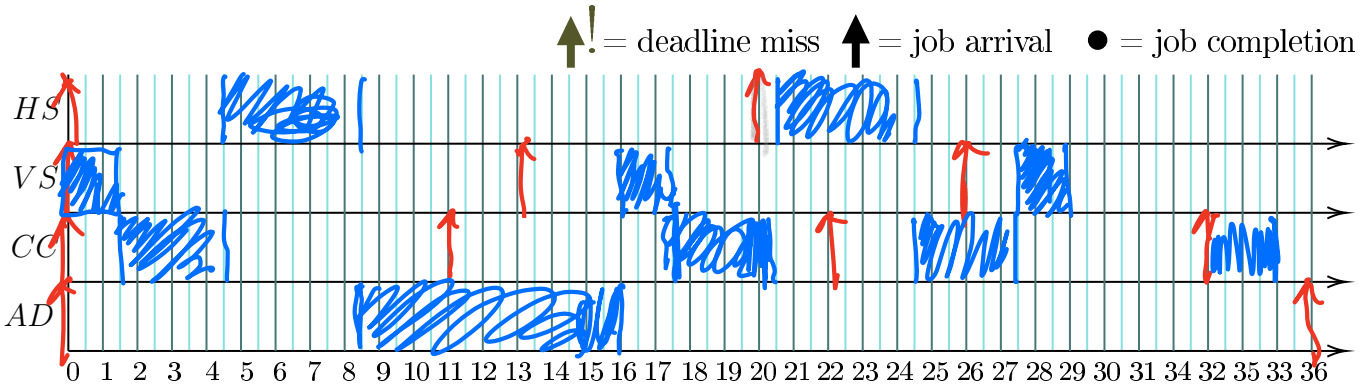
```
1 lock = 0;
```

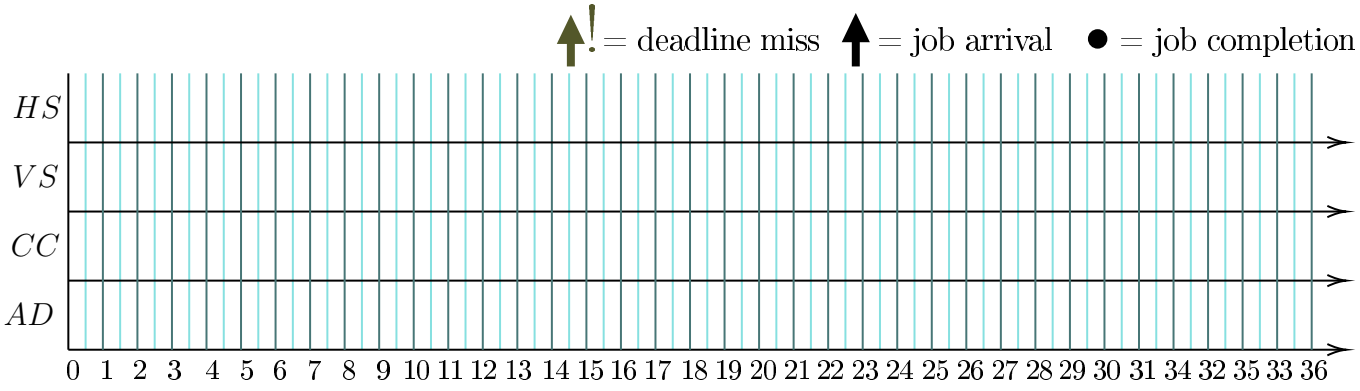
Listing 10: Common Exit Section

Step	Thread T1	Thread T2
1		lock = 2
2	lock = 1	
3		while (true) {
4	while (true) {	
5		if (lock == 2)
6	if (lock == 1)	
7	break;	
8		if (lock == 2)
9	...	
10	lock = 0;	
11		if (lock == 2)
12	lock = 1;	

sem 1
sem. type: 2
sem. queue: 1
0
wait( )
↗ po

[EXTRA GRIDS (*Use only if necessary and reference properly.*)]





Step	Thread T1	Thread T2
1		
2		
3		
4		
5		
6		
7		

[EXTRA BLANK PAGE]

Some Schedulability Tests

- Minimum Slowdown for job j_i at time t: $\frac{t-a_i+C_i}{C_i}$
- m tasks schedulable with RM if: $U \leq m(2^{1/m}-1)$
- Tasks schedulable with RM-FF on N CPUs if: $U \leq N(\sqrt{2}-1)$
- Tasks schedulable with EDF-FF on N CPUs if: $U \leq \frac{\beta N+1}{\beta+1}$ where $\beta = \lfloor \frac{1}{\max_k U_k} \rfloor$

Some Useful Numbers

- $2^{1/2} = 1.414213562$
- $2^{1/3} = 1.25992105$
- $2^{1/4} = 1.189207115$
- $2^{1/5} = 1.148698355$
- $2^{1/6} = 1.122462048$
- $2^{1/7} = 1.104089514$
- $2^{1/8} = 1.090507733$

