COMPSCI 2SD3 Midterm Test 2
SOLUTIONS

Dr. F. Franek

DURATION : 50 minutes
McMaster University (CAS) March, 2022

Please PRINT CLEARLY your name and student number

NAME: _____

Student Number:

question	mark	out of
1-10		8
21		3
22		4
total		15

This test paper includes 9 pages, 8 multiple-choice questions, and 2 written questions. You are responsible for ensuring that your copy of the paper is complete. Bring any discrepancy to the attention of your invigilator.

Special Instructions :

1. The multiple-choice questions of this test must be answered on the form at the back of this questionnaire.
2. The written questions must be answered in the space provided in this questionnaire.
3. Documents to be returned: this questionnaire and all scrap paper if used. All of these documents must bear your name and student number. Only the face page of the questionnaire need to bear your name and student number, and all the loose pages of the questionnaire, if any.
4. No memory aids or textbooks of any kind are allowed during the test, except one letter-sized sheet of information (possibly on both sides) - this crib sheet must bear your name and student number.
5. No calculators, pocket computers, smartphones, or tablets are to be utilized.
6. No unauthorized scrap paper is allowed to be used. The invigilator(s) will supply you with needed scrap paper when you ask.
7. You are not allowed to be involved in any communication of any kind concerning the questions and answers of this test with anybody except the invigilator(s) or the instructor. Any attempt of such communication will be considered a case of academic dishonesty.

continued on next page

Questions 1 – 8 are multiple-choice questions and are to be marked on the form provided at the back of this questionnaire. For each question always select only one answer, even if you think that there are more correct answers than one. In the case that there are more correct answers, selecting one of them will earn you the full credit. Some answers may be awarded a partial credit. The negative marking is not used, i.e. incorrect or missing or multiple answers earn a 0 mark. Only the answers marked appropriately on the form at the back of this questionnaire will be considered, all other will be ignored.

Question 1 [1 mark] Consider the differences between UNIX user threads and processes. Which of the following is the most appropriate statement?

A. The difference between processes and user threads is in their scheduling. User threads are scheduled by the process itself, while the processes are scheduled by the operating system.

B. The difference between processes and user threads is in their sharing of the address space. Processes do not share the address space, while threads do.

C. The difference between processes and user threads is in their context switch. User threads do not require as complex context switch as processes do.

$\Longrightarrow$D. All of the above.

E. None of the above.

Explanation:
A and B and C are all true, hence D.

Question 2 [1 mark] Consider a round-robin scheduling of jobs on a CPU with a simplification that each job is assigned quantum with the same length (ie., all jobs get assigned the same quantum). Which of the following statements is most appropriate?

A. The shorter the quantum, the faster the jobs are executed.

B. The shorter the quantum, the more jobs can be executed.

$\Longrightarrow$C. The shorter the quantum, the more responsive the jobs are.

D. All of the above.

E. None of the above.

Explanation:
A is wrong, the speed of execution has nothing to do with the quantum. B is wrong, the quantum has nothing to do with the number of jobs being executed. C is correct.

Question 3 [1 mark] Consider the following program (we omitted the header files and stuff related to using log functions for simplification):

continued on next page

```
int main() {
  int tid1, tid2;
  int* ret;
  int a;

  a=2;
  if (pthread_create(&tid1, NULL, doit,(void*)&a) != 0)
    sys_exit("pthread error\n");
  a=3;
  if (pthread_create(&tid2, NULL, doit,(void*)&a) != 0)
    sys_exit("pthread error\n");

  pthread_join(tid1,(void**)ret);
  pthread_join(tid2,(void**)ret);

  msg_exit("terminating the process\n");
  return 0;
}

void* doit(void* a) {
  msg("got %d\n",*((int*)a));
  pthread_exit(NULL);
}
```

What messages will we see on the screen when we execute the program?

A. `got 2`
   `got 2`
   `terminating the process`
B. `got 2`
   `got 3`
   `terminating the process`
C. `got 3`
   `got 3`
   `terminating the process`
D. It cannot be determined, but it will be either A or C.
$\Longrightarrow$E. It cannot be determined, but it will be either B or C.

Explanation:
If the first thread is dispatched before `a` is reset to 3, it will display `got 2` on the screen; if it is dispatched after `a` was reset to 3, it will display on the screen `got 3`. The second thread will always display `got 3`. Hence either B or C always happens, making E correct.

Question 4 [1 mark] If a signal is sent to a process with several threads, which of the threads receives the signal?

$\Longrightarrow$A. All of the threads.
B. The main thread only, i.e., the thread with the lowest tid.

continued on next page

$\Longrightarrow$C. [partial credit] The thread that is just executing when the signal is received.
$\Longrightarrow$D. None of the above.

Explanation:
D is correct. C is partially correct (a partial credit for that answer), however a signal may be sent when the process is not in fact executing. Hence the signal will be received by the first thread to start executing when the process is switched on. In one of my lectures I mentioned that 'all threads receive the signal' – it was not a good formulation – what I meant was that 'any of the threads may receive the signal'. For that reason some student may got confused, and thus A is accepted as well.

Question 5 [1 mark] Consider the following program (we omitted the header files and stuff related to using log functions for simplification):

```
pthread_mutex_t flock = PTHREAD_MUTEX_INITIALIZER;

// function main ------------------------------------------------------
int main() {
  int tid1, tid2;
  int* ret;

  if (pthread_create(&tid1, NULL, doit,NULL) != 0)
    sys_exit("pthread error\n");
  if (pthread_create(&tid2, NULL, doit,NULL) != 0)
    sys_exit("pthread error\n");

  pthread_join(tid1,(void**)ret);
  pthread_join(tid2,(void**)ret);

  msg_exit("terminating the process\n");
  return 0;
}

void* doit(void* a) {
  pthread_mutex_lock(&flock);
  msg("thread %d, hit enter to continue",pthread_self());
  getchar();
  msg("thread %d, bye\n",pthread_self());
  pthread_exit(NULL);
}
```

Scenario A:

```
thread 2, hit enter to continue
thread 2, bye
thread 3, hit enter to continue
thread 3, bye
terminating the process
```

Scenario B:

```
thread 2, hit enter to continue
```

continued on next page

```
thread 2, hit enter to continue
thread 2, bye
```

Which of the two scenarios given above will we see on the screen?
Do not consider the values of thread ids; they, of course, need not be 2 and 3, but that is not to be considered relevant for this question.

A. Scenario A, because the critical section of `doit()` starts with `pthread_mutex_lock(&flock);` and ends with `pthread_exit(NULL);`. Thus each thread is "protected" during the "conversation" with the user.

B. Scenario B, since the critical section of `doit()` starts with `pthread_mutex_lock(&flock);` and ends with `pthread_exit(NULL);` and thus the mutex `flock` is never unlocked. Thus the first thread will lock `flock` and the second thread will wait forever.

C. It cannot be determined, it depends on the timing of events. If the first thread terminates before the second thread tries to lock `flock`, everything will be OK and we will see Scenario A, otherwise we will see Scenario B.

$\Longrightarrow$D. We will see neither of the given scenarios.

Explanation:
We will see

```
thread 2, hit enter to continue

thread 2, bye
```

since the first thread to lock the mutex `flock` will not unlock it (`pthread_exit()` will not do it), `flock` will remain locked and when the next thread arrives there, it will wait forever.

Question 6 [1 mark] What is a difference between the terms `deadlock prevention` and `deadlock avoidance`?

A. No difference, they essentially mean the same thing.

B. A small difference in how to make sure that deadlocks do not happen. If we prevent all the necessary conditions for a deadlock to occur, we talk of `deadlock prevention` and it guarantees no deadlocks. If we prevent only some of the conditions for a deadlock to occur, we talk of `deadlock avoidance` and it guarantees very few deadlocks, though they still can happen.

$\Longrightarrow$C. A big difference, in `deadlock prevention` we make sure that deadlocks cannot happen by preventing some or all of the necessary conditions for deadlocks to occur. In `deadlock avoidance` deadlocks can still happen, but by careful planning ahead we avoid them from happening.

D. None of the above.

Explanation:
C is a correct description of the differences between the two terms.

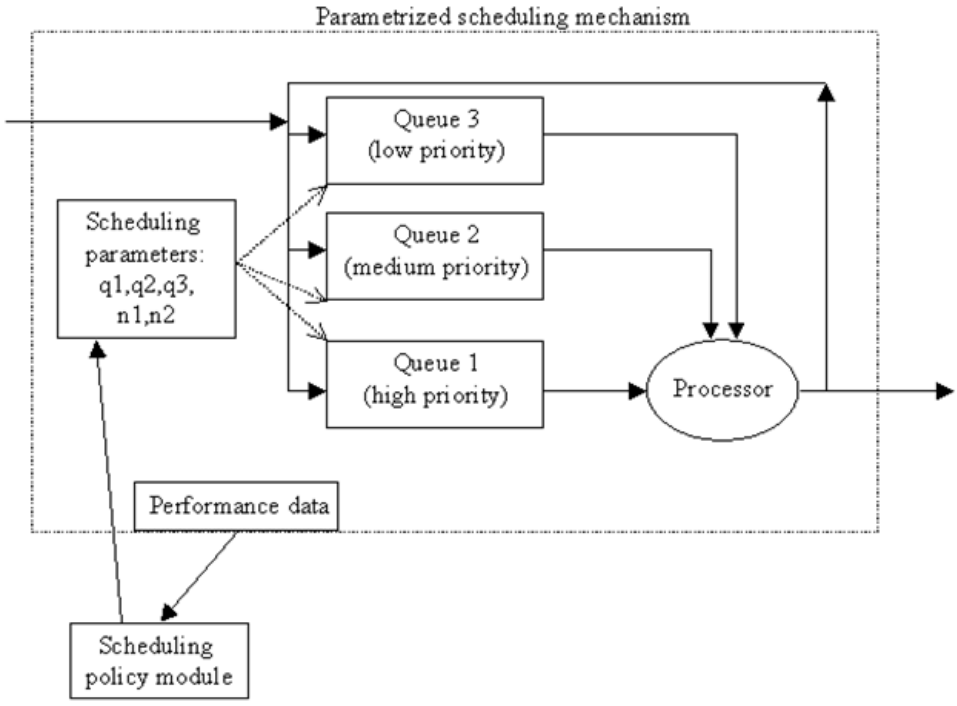
continued on next page

Question 7 [1 mark] Consider the HRRN (Highest-Response-Ratio-Next) scheduling system. It is based on the simple SJF (Shortest-Job-First) scheduling model, but it is designed to improve it. What does it improve?

A. In comparison to SJF, HRRN improves the responsivness of the jobs.
B. In comparison to SJF, HRRN improves the turn-around of the jobs (so more jobs can be run).
$\Longrightarrow$C. In comparison to SJF, HRRN improves the chances of longer jobs to get a turn and thus prevent so-called "starvation".
D. All of the above.
E. None of the above.

Explanation:
Scheduling has nothing to do with responsivness of jobs (that depends on the quantum), and so A is wrong. It has nothing to do with turn-around (again, that depends on the quantum), so B is wrong. C is correct.

Question 8 [1 mark] Consider the 3-queue parametrized scheduling system we discussed in class and schematically described by the following diagram:



What must be the values of the parameters making the system behave like a 2-queue scheduling system?

A. The parameters cannot be set to achieve it.
B. With three queues it can never behave like a 2-queue system regardless the value of parameters.

continued on next page

$\Longrightarrow$C. $n1 = 1$, $n2 = \infty$, $q1 \leq q2 < \infty$.
D. $n1 = 2$, $n2 = \infty$, $q1 \leq q2 < \infty$.
E. None of the above.

Explanation:
Of course, C is correct. A job stays in queue 1 just for one turn and then it stays in queue 2 for the rest of the execution. The quantum for queue 1 is shorter or equal the quantum for the queue 2.

continued on next page

Questions 9 – 10 are questions that require a written answer. The answers are to be written in the space following each question.

Question 9 [3 marks] Discuss the advantages and disadvantages of using processes over user threads in UNIX.
Answer:

The main difference between processes and threads is in whether the address spaces is copied or not. For processes it is copied, for threads it is not. The main advantages and disadvantages are derived from this simple fact.

	Processes	Threads
Protection of data	Adv	Disadv
Needs critical section	No - Adv	Yes - Disadv
Needs more resources (memory)	Yes - Disadv	No - Adva
Complex context switch	Yes - Disadv	No - Adv
Inter communication	Complex - Disadv	Easy - Adv

continued on next page

Question 10 [**4 marks**] Write a simple program that dispatches two threads. Each thread computes the value 7*x where x is passed to the thread and returns the value it computed. The first thread is passed value 2, the second thread is passed value 3. The program displays the value computed by the first thread and the value computed by the second thread and terminates. Do not bother with header files and functional prototypes in your program.

A sample solution:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void* doit(void*);

// function main ------------------------------------------------
int main() {
  int tid1, tid2;
  int a=2;
  int b=3;
  void *ret1,*ret2;

  pthread_create(&tid1, NULL, doit,(void*)&a);
  pthread_create(&tid2, NULL, doit,(void*)&b);

  pthread_join(tid1,&ret1);
  pthread_join(tid2,&ret2);

  printf("first thread computed %d\n",*((int*)ret1));
  printf("second thread computed %d\n",*((int*)ret2));

  return 0;
}

// function doit ------------------------------------------------
void* doit(void* a) {
  int* b=(int *)malloc(sizeof(int));
  int* c=(int*) a;
  (*b)=(*c)*7;
  return ((void*)b);
}
```

THE END.