Chalmers University of Technology and Gothenburg University

Operating Systems
EDA093, DIT 401

Exam 2022-08-16

Date, Time: Tuesday 2022/08/16, 14.00-18.00

Course Responsible:
Vincenzo Gulisano (031 772 61 47)

Auxiliary material: You may have with you
- An English-Swedish, Swedish-English dictionary.
- No other books, notes, calculators, etc.

Grade-scale ("Betygsgränser"):
CTH: 3:a 30-39 p, 4:a 40-49 p, 5:a 50-60 p
GU: Godkänd 30-49p, Väl godkänd 50-60 p

Exam review ("Granskningstid"):
Will be announced after the exam.

Instructions

- Do not forget to write your personal number, if you are a GU or CTH student and at which program ("linje").
- Start answering each assignment on a new page; number the pages and use only one side of each sheet of paper.
- Write in a **clear manner** and **motivate** (explain, justify) your answers. If it is not clear what is written, your answer will be considered wrong. If it is not explained/justified, even a correct answer will get **significantly** lower (possibly zero) marking.
- If you make **any assumptions** in answering any item, do not forget to clearly state what you assume.
- The exam is organized in groups of questions. The credit for each group of questions is mentioned in the beginning of the respective group. Unless otherwise stated, all questions in a group have equal weight.
- Answer questions in English, if possible. If you have large difficulty with that and you think that your grade can be affected, feel free to write in Swedish.

Good luck !!!!

1. (12 p)

   (a) (6 p) Assume the following setup: a computer C has a dual-core CPU, CPU0 is the first CPU while CPU1 is the second CPU. C runs 2 copies of the same operating system: OS0 and OS1. OS0 runs entirely and exclusively on CPU0. OS1 runs entirely and exclusively on CPU1. C's main memory is partitioned into disjoint partitions, one for OS0 and one for OS1. Make a detailed example showing that OS1 can access stale data because of OS0 if I/O is not synchronized.

   [**HINT:** possible answer: OS0 reads file F from disk into memory. OS0 modifies the file in memory. OS0 tells the OS to persist the file. OS0 postpones the write to complete other I/O. OS1 reads the file from the disk into memory. OS1 and OS0 views of the same file are now different.]

   (b) (6 p) Imagine there exist two different variants of the lottery scheduling algorithm. Variant V0 distributes lottery tickets to processes and picks the next process to run as the one having the next drawn ticket. Variant V1 distributes 2 types of tickets, to users and to processes (the numbering of tickets is independent for each user e.g., user U0 can have tickets T0, T1, and T2 while user U1 can have tickets T0, T1, T2, and T3). Variant V1 makes the first draw to pick the user and, subsequently, a second draw to choose the user's process. Can V0 mimic the behavior of V1? Explain and, if you think it can, make an example.

   [**HINT:** Yes. The probability with which a process P from a certain user U is chosen in V1 is Prob(P)*Prob(U). The same probability value can be given by accurately distributing tickets to processes by V0. Example: Tickets distribution in V1: U0 3 tickets, U0-P0 2 tickets, U0-P1 2 tickets, U1 4 tickets, U1-P0 3 tickets, U1-P1 3 tickets, U1-P2 3 tickets. Probabilities in V1: 3/14 for each one of U0's processes and 4/21 for each one of U1's processes. LCM of 12 and 21 is 42. In V0, draw from 42 tickets, give 9 to each one of U0's processes and 8 to each one of U1's processes.]

2. (12 p)

   (a) (4 p) Write two small sets of instructions for 2 threads. One is adding a certain amount to a shared variable that refers to a bank account balance. The other is removing a certain amount from it. Show that these threads, if running without any synchronization mechanism, can lead to undesired behavior when their execution is concurrent on the same CPU core (with context switches alternating their execution). Assume that both read and write operations are atomic.

   [**HINT:** This is basically the initial example from the synchronization lectures.]

   (b) (6 p) Does this code satisfy the mutual execution property? Motivate your answer with a formal proof.

```
shared var flagA,flagB: boolean; // Indicate A's,B's
    // intention to enter their critical section. They are
    // both set to false initially.

A's code {
    repeat
        while flagB==true do nothing;
        flagA = true;
```

```
        // critical section
        flagA = false;
        // remainder
    forever
}

B's code {
    repeat
        while flagA==true do nothing;
        flagB = true;
        // critical section
        flagB = false;
        // remainder
    forever
}
```

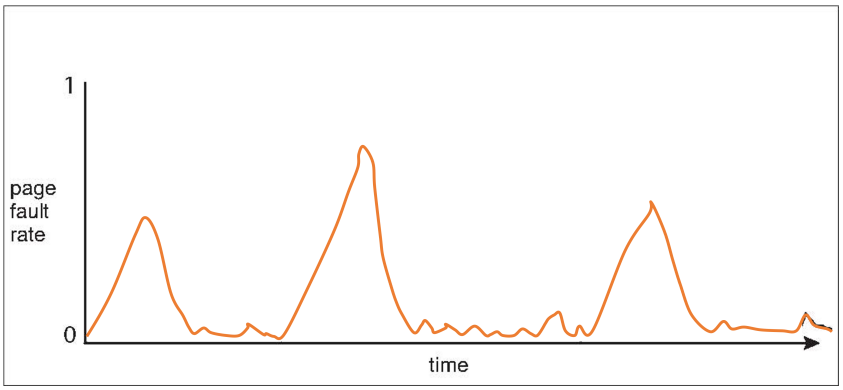
[**HINT:** it does not, proof in slides about synchronization (lecture 5)]

(c) (2 p) Which 4 conditions need to hold simultaneously for a deadlock to occur?
[**HINT:** Mutual exclusion, hold and wait, no preemption, and circular wait]

3. (12 p)

(a) (4 p) Explain what is the connection between locality and thrashing.
[**HINT:** Locality is the set of pages currently accessed by a process. If the sum of sizes of localities is more than available memory then thrashing.]

(b) (4 p) You are given the following graph. Make some hypothesis about the reason why the page fault rate has such a behavior.



[**HINT:** It could be that spikes are in correspondence with changes in the locality of the process. New locality, accessing pages that were not accessed before, page faults to retrieve them. Another case could be there is a lack of memory and during that time some other process is stealing frames.]

(c) (4 p) Calling *exec* in a child process just after the parent invokes *fork* might lead to a waste of time during the copying of the parent process memory (waste because the child is going to load another process in such memory). Which mechanism can be used to optimize this? Explain how such a mechanism works.
[**HINT:** Copy on Write (Virtual memory lecture).]

4. (12 p)

(a) (4 p) Suppose you have two OSs with different APIs to handle the waiting queues of I/O devices. OS1 offers 2 methods: *void add(PCB x)* and *PCB get()*, which allow to add and get PCBs in FIFO order (let's consider there's only one I/O device for simplicity) . OS2 offers a third method *PCB get(PCB x)*. Can both APIs be used if you want to serve I/O requests in arbitrary order? Explain.

[**HINT:** The second is more convenient, but the first can work too, you can write a procedure that keeps getting until it gets the right PCB, temporarily maintain other PCBs, and then pushes back the latter.]

(b) (4 p) In which cases would you prefer message passing for interprocess communication and in which cases would you prefer shared memory? Why?

[**HINT:** See slides in the Processes lecture. Shared-memory faster, more customizable, message passing safer / handled by the OS]

(c) (4 p) A denial-of-service attack can be executed by creating a process that continuously replicates itself to deplete available system resources. Can you write the C code such a process would use?

[**HINT:** It's a fork bomb, keep calling fork in a while loop]

5. (12 p)

(a) (4 p) What is the difference between a guest-induced and a hypervisor-induce page fault? What are the differences in the way they are handled?

[**HINT:** Check slides from VM lecture. guest-induced from the virtual OS, hypervisor from the hypervisor. Guest-induced need to be re-injected to the OS.]

(b) (4 p) Explain in detail how a type-1 hypervisor virtualizes memory.

[**HINT:** Check slides from VM lecture. Shadow page table, different handling of it (read-only pages vs waiting for page faults]

(c) (4 p) Explain in detail how the ballooning technique works.

[**HINT:** Check slides from VM lecture.]