Chalmers University of Technology and Gothenburg University

Operating Systems
EDA093, DIT 401

Exam 2019-08-20

Date, Time, Place: Monday 2019/08/20, 14.00-18.00, SB Multisal

Course Responsible:
Vincenzo Gulisano (031 772 61 47),
Marina Papatriantafilou (031 772 54 13)

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) (4 p). Write the code of a C program that, once started, results in (1) an initial instance A that copies itself to a second instance B and in which (2) A shares information with B using a variable initialized by A before copying itself into B.

   [**HINT:** fork + pipe.]

   (b) (4 p). Are the following statements true or false? Explain why.

      i. One CPU can only run one program at the time.
      ii. One program can run in parallel and concurrently at the same time.
      iii. One process can run in parallel and concurrently at the same time.
      iv. Two processes of the same program can run sharing nothing.

   [**HINT:** False, one core. Yes, because it can have multiple-threads. Yes, same as before. Yes, two instances of the same program (e.g., fork) with no global variables share nothing, for instance.]

   (c) (4 p). Suppose that the speedup / scalability you observe for a parallel program is higher than that predicted by Amdahl's Law. What could be the reason behind such a behavior?

   [**HINT:** Some resources increase with higher parallelism (e.g., caches). In this sense, having 2 threads, each with a smaller potion of data to process, might result in having better cache behavior (and thus higher-than-expected scalability).]

2. (12 p)

   (a) (4 p) Explain why too large and too short working-set windows affect the accuracy with which trashing is detected by the OS.

   [**HINT:** too large: might detect trashing even if many pages not needed to encompass processes' localities are wasted. too small: might not detect thrashing even if processes have high page fault rates.]

   (b) (4 p) Discuss how the page fault rate is expected to behave if the working-set window is large enough.

   [**HINT:** Increase when changing locality but then going flat to almost 0 until the next locality.]

   (c) (4 p) Discuss what global and local page replacement are and their trade offs.

   [**HINT:** Check corresponding slide(s) from Virtual Memory lecture.]

3. (12 p)

   (a) (4 p) Describe what memory mapped I/O is.

   [**HINT:** Check corresponding slide(s) from I/O Systems lecture.]

   (b) (4 p) Describe at least 2 ways in which non-maskable interrupts can help you fix bugs in your code.

   [**HINT:** Debug-related interrupts (which allows you to debug programs). Alternative: exceptions (like floating-point exceptions).]

   (c) (4 p) Describe the six main steps initiating and taking place during a DMA transfer from a device to the main memory.

   [**HINT:** Check corresponding slide(s) from I/O Systems lecture.]

4. (12 p)

(a) (4 p) Describe (i) four optimization goals of scheduling in single-processor systems and (ii) two additional ones in multiprocessor systems.

[**HINT:** (i) CPU utilization, waiting time for threads/processes, throughput, fairness (ii) minimize migration, optimize synchronization waiting time]

(b) (8 p) For the goals that you described in the previous question, describe scheduling methods that aim at addressing them. Explain carefully what these methods achieve and justify the claims.

[**HINT:** maximizing CPU utilization is related to minimizing overhead, i.e. context switching, hence longest-job first is helpful; round-robin guarantees "flat" fairness, Linux algorithm provides weighted fairness; shortest job/process first maximizes throughput (proof sketch on slides for SJF/SPF and average waiting time); gang scheduling policies target both goal in multiprocessors]

5. (12 p)

(a) (4 p) A compare-and-swap (CAS) is an instruction available in common hardware, with slight variations in behaviour. One possible behaviour is given in the following pseudocode ("*" denotes access through a pointer):

```
function CAS(p : pointer to int, old : int, new : int) returns bool {
    if *p != old {
        return false
    }
    *p := new
    return true
}
```

Show a method that solves the critical section problem for arbitrary number of threads using CAS. Use pseudocode in the description and argue about the properties of the solution, with respect to mutual exclusion, progress and fairness.

[**HINT:** It can be used to simulate a TestAndSet. Solution idea: see pseudocode slides 17 (or 46, with fairness) in synchronization lecture. Argumentation as for the mutual exclusion algorithm by Peterson. It can also be used to simulate Lamport's bakery algo]

(b) (4 p) (i) Describe the four necessary conditions for a deadlock to occur among threads/processes, in the context of allocation of reusable resources. (ii) How does the knowledge of these conditions help in preventing deadlocks?

[**HINT:** Mutual exclusion, circular wait, no preemption, hold and wait. If one of them is not possible in a RA method, then deadlock is prevented.]

(c) (4 p) Can the critical section problem be solved by (i) mutex locks? (ii) binary semaphores? If so explain how including pseudocode, if not explain why not.

[**HINT:** yes, as on slide 17 in synch-lecture]