Chalmers University of Technology and Gothenburg University

Operating Systems
EDA093, DIT 401
Exam 2018-10-27

Date, Time, Place: Saturday 2018/10/27, 08.30-12.30, Hörsalar på hörsalsvgen

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 program that can be used to make a copy of a certain file (specified by the user). This program should be run by two threads T1 and T2 belonging to two different processes P1 and P2 (T1 belongs to P1, T2 belongs to P2). T1 is the thread responsible for asking which file to copy (and the file name of the copy) while T2 is the thread responsible for making the actual copy.

   [**HINT:** process with fork, parent asks for file names and child does the copy (or the opposite), they can communicate with a pipe (among other alternatives).]

   (b) (4 p) A multi-threaded process implemented using the many-to-one threading model can run both concurrently (on the same core) and in parallel (at different cores). True or false? Explain why.

   [**HINT:** concurrent true, parallel false (because only 1 kernel thread cannot be executed in parallel at different cores).]

   (c) (4 p) Why does the OS define a "terminated" state for a process and does not simply end the process? Provide an example of when this is useful.

   [**HINT:** share information about this process with other processes (e.g., for zombie processes).]

2. (12 p)

   (a) (4 p) Discuss the effects of underestimating and overestimating the parameter Δ of the working-set window.

   [**HINT:** too small means locality is not covered, so lots of page faults. Too big means locality is over-covered, so less processes than possible will run.]

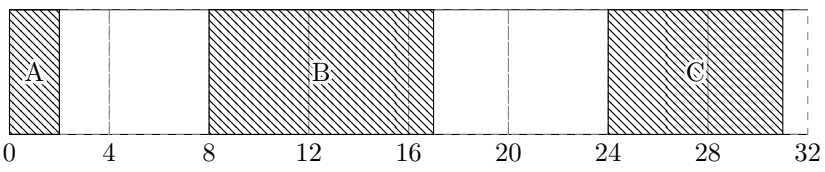
   (b) (4 p) if the pages loaded in n frames using a certain page replacement algorithm are a subset of those loaded for $n+1$ frames using the same page replacement algorithm, then the Belady's anomaly cannot be observed. True or false? Discuss why

   [**HINT:** if a page fault is not observed for a certain page with n frames, then it will also not be observed with $n+1$ frames... so the number of page faults cannot increase for an increasing number of available frames.]

   (c) (4 p) If the Least Recently Used page replacement algorithm is implemented using reference bits why is it an approximation? Can you provide an example showing it is not exact?

   [**HINT:** if two frames A and B have the same (lowest) count, the first encountered one will be replaced. But that one is not necessarily the least recently used (its reference bit for A could be 1 and then be switched to 0 while B's could always have been 0, so B should be replaced first, but that is not necessarily going to happen).]

3. (12 p) Consider the following allocation of 3 files (A of 2KB, B of 9KB and C of 7KB) on a 32KB contiguous zone on disk using 4KB block size:



(a) (3 p) What is the quantity of internal fragmentation in the presented allocation? How can we reduce it? What other system parameter is then impacted and how?
[**HINT:** 2+3+1=6KB; decrease block size to 1KB; data reading rate is reduced as more block to read.]

(b) (3 p) How much space would be used on disk in the best possible allocation with block sizes of 1KB, 8KB and 16KB? Detail your reasoning.
[**HINT:** 26KB, 4*8=32KB, 3*16=48KB. Cannot avoid internal fragmentation in the latter cases.]

(c) (3 p) Assume contiguous allocation has been used to arrive to the presented allocation. Give pros and cons of contiguous allocation and 2 examples of different operations on files that might have led to the final allocation.
[**HINT:** pros: easy to implement & fast to read file; cons: external fragmentation, need to move files when they grow, need compaction... examples: (1) deletion of two 4KB files between A and B, and B and C (2) A and B got reduced by removing some lines (3) some files had to be reallocated because they grew beyond their space...]

(d) (3 p) What are the two most commonly used data structures to keep track of free space? Present them briefly. Which one is more space efficient and when?
[**HINT:** linked-list of free blocks and bitmap of blocks (cf Lecture 9 slide 43). linked-list needs 1 pointer per free block whereas bitmap needs 1 bit per block. the best one depends on how many free blocks they are, the fewer the better linked-list is.]

4. (12 p)

(a) (2 p) Consider a CPU scheduling algorithm that favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?
[**HINT:** will favor the former because of their commonly short CPU bursts; however, the CPU-bound programs will not starve, as the I/O-bound programs will relinquish the CPU relatively often to do their I/O, unless the latter are too many; but then there are risks for thrashing]

(b) (3 p) What is the reason for having different time-quantum sizes on different levels of a multilevel queuing system?
[**HINT:** Processes that need more frequent servicing, eg interactive ones, can be in a queue with a small quantum. Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing (efficiency)]

(c) (3 p) What does the term "processor affinity" mean in the context of multiprocessor scheduling? Describe advantages and limitations of processor affinity.
[**HINT:** it means to avoid migration of a process from its assigned processor to another; + overhead savings; - limiting possibilities for load balancing among processors]

(d) (4 p) Discuss the EDF policy (Description, properties: advantages and limitations/disadvantages).
[**HINT:** Describe the method; explain main thm; hence EDF it can serve as schedulability test; further, it implies dynamic priorities, hence it places higher demands on the ready-queue data structure.]

5. (12 p)

(a) (2 p) What is a race condition? Explain the term and provide an example.

[**HINT:** when the outcome of executing code depends on timing and can give erroneous results, e.g. the example of unprotected concurrent access to bank balance shared data.]

(b) (2 p) Consider two threads, A and B. Thread B must execute operation opB only after thread A has completed operation opA. How can you guarantee this synchronization using semaphores?

[**HINT:** In slides about synch]

(c) (4 p) We have three threads, A, B, C (taking care of operations opA, opB, opC respectively) that can arrive to execute in any order. The initial values for the semaphores that they use, are: `semaphores semA=1, semB=1, semC=0`.

```
thread A:
  wait(semC);
  wait(semB);
  opA;    // some operation
  signal(semB);
  signal(semA);

thread B:
  wait(semA);
  wait(semB);
  opB;
  signal(semB);

thread C:
  wait(semA);
  wait(semB);
  opC; //some operation
  signal(semB);
  signal(semA);
  signal(semC);
```

Are the following executions possible or not and why? (i) opA opB opC (ii) opB opC opA (iii) opC opA opB (iv) opB opA opC

[**HINT:** (i) no: A blocks at semC until C has executed opC; (ii) no: B "consumes" from semA but does not produce in it, hence C will block; (iii) true (show execution); (iv) same as ii, now A blocks at semC until C has executed opC; ie if B executes first, no other thread can proceed]

(d) (4 p) Consider 2 threads, A and B, which must forever take turns executing operation opA and operation opB respectively. Thread A must be the one that executes opA first. How can you guarantee that using semaphores? Formulate the safety and progress requirements and argue about the correctness of your solution.

[**HINT:** Arguing as for the producer consumer problem and extending question (b); Now the threads work in a loop, using semaphores SA and SB: in each of their respective loop iterations, A must wait for opB, ie wait(SB) (except the first time, hence SB is initialized to 1) and B must wait for opA, ie wait(SA), with SA initialized to 0. After opA (resp opB),

A executes signal(SA) (resp B executes signal(SB), to generate the required signal to enable the other one to proceed. i.e. `A: do wait(SB); opA; signal(SA) forever` and `B: do wait(SA); opB; signal(SB) forever` `init SA = 0, SB = 1`. We must show that no thread will block forever (progress) and that it cannot be the case that we have more than one iteration of A between successive iterations of B (safety)]