Linköpings universitet
IDA Department of Computer and Information Sciences
Prof. Dr. Christoph Kessler

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem /
Concurrent programming and operating systems

19 mar 2014, 14:00–18:00 TER3, TER4

Jour: Christoph Kessler (070-3666687, 013-282406), visiting ca. 16:00

Hjälpmedel / *Admitted material:*

– Engelsk ordbok / *Dictionary from English to your native language*;
– Miniräknare / *Pocket calculator*

General instructions

- This exam has 9 assignments and 5 pages, including this one.
  Read all assignments carefully and completely before you begin.
- Please **use a new sheet of paper for each assignment**, because they will be corrected by different persons.
  Sort the pages by assignments and number them consecutively.
- You may answer in either English or Swedish. **English is preferred** because not all correcting assistants understand Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points. You may thus plan about 5 minutes per point.
- Grading: U, 3, 4, 5. The preliminary threshold for passing is 20 points.

1. (6 p.) **Interrupts, processes and threads**

   (a) Define the terms *process* and *thread*.
   In particular, what are the main differences between processes and threads, and what do they have in common? Be thorough! (2p)

   (b) We discussed different thread models with different relations (mappings) between user and kernel threads. Describe one of these models that is appropriate for use on a *multiprocessor architecture*, and explain why. (1.5p)

   (c) How does the value of the *kernel mode bit* affect the execution of code by the processor?

   And by which events or operations is the kernel mode bit set and reset, respectively? (1.5p)

   (d) Two main methods for inter-process communication in a computer are *shared memory* and *message passing*. Which of the two methods is likely to have less overhead if two processes communicate frequently with each other, and why? (1p)

2. (6 p.) **CPU Scheduling**

   (a) Given a single-CPU system and the following set of processes with arrival times (in milliseconds), expected maximum execution time (ms), and priority (1 is highest, 5 is lowest priority).

Process	Arrival time	Execution time	Priority (as applicable)
P_1	0	5	5
P_2	1	4	4
P_3	4	2	2
P_4	7	3	3
P_5	9	2	1

   For each of the following scheduling algorithms, create a Gantt chart (time bar diagram, starting at $t = 0$) that shows when the processes will execute on the CPU. Where applicable, the time quantum will be 2 ms. Assume that a task will be eligible for scheduling immediately on arrival. If you need to make further assumptions, state them carefully and explain your solution. (5p)

   (i) FIFO;

   (ii) Round-robin;

   (iii) Shortest Job First *without* preemption;

   (iv) Priority Scheduling *without* preemption.

   (v) Priority Scheduling *with* preemption.

   (b) There exist several strategies for CPU scheduling on a multiprocessor system. One of them is *affinity-based scheduling*. How does it work (basic idea, no details) and what is the motivation for it? (1p)

3. (3.5 p.) **Synchronization**

   Electronic voting server

   An electronic voting system uses as central data structure a global array

```
unsigned int votes[N];    // N = number of parties
```

   for counting the number of votes for each of the N parties, initialized to zeroes.

   The voting server program, originally single-threaded, processes one vote request received from an external client at at time, by calling the function

```
void cast_vote ( unsigned int i )
{
 if (i>=N)
    error(); // invalid vote
 else
    votes[i] = votes[i] + 1; // count vote for party i
}
```

   In order to scale up to many millions of voters, the voting program should be multi-threaded (with the `votes` array residing in shared memory of the voting server process) and run on a multicore server running a modern multithreaded operating system with preemptive scheduling, so that multiple calls to `cast_vote` can execute concurrently. It is your task to make the program thread-safe and preserve its correct behavior.

   (a) Using a simple contrived scenario (Hint: use $N = 2$ parties and few votes), show with a concrete example (interleaving of shared memory accesses) that, without proper synchronization, race conditions are possible that could even cause that the wrong party wins the election. (1p)

   (b) Identify the critical section(s) and protect the program against race conditions with a *single* mutex lock. (1p)

   (c) Explain why the solution in (b) might have a scalability problem, and propose a (somewhat) better lock-based (race-free) solution. (1.5p)

4. (4 p.) **More synchronization**

   (a) What is the purpose of a *condition* (also known as *condition variable*) in the context of a critical section, what operations does it provide and what do they do, and how can conditions be used for advanced synchronization? (2p)

   (b) Explain how mutual exclusion synchronization can interfere with priority based scheduling. (2p)

5. (5 p.) **Deadlocks**

   (a) There are four conditions that must hold for a deadlock to become possible. Name and describe them briefly. (2p)

   (b) You are given a system with 4 types of resources, A, B, C and D. There are 4 instances of A, 5 instances of B, 1 instance of C and 7 instances of D. Currently, 5 processes $P_1 ... P_5$ are running, and for each process, the resources currently held and its total maximum resource need (including the already held ones) for each type are given as follows:

Process	Already held	Maximum total need
	A B C D	A B C D
P_1	0 1 0 1	0 2 0 4
P_2	0 0 0 0	4 3 0 5
P_3	1 2 1 0	2 2 1 3
P_4	1 0 0 2	2 2 0 4
P_5	1 0 0 2	2 0 1 4

   I.e., currently, process P_1 holds already one B and one D out of the 2 Bs and 4 Ds that it eventually may need in the worst case, etc. One A is currently still available, etc.

   (i) Show that the system is currently in a safe state (calculation). (1.5p)

   (ii) In the situation given above, process P_1 now asks for 1 instance of D, in addition to the B and D that it already has. Is it safe to grant the request? Why or why not? (calculation) (1.5p)

6. (8 p.) **Memory management**

   (a) Consider a page-based virtual memory system with a page size of 1024 bytes where virtual memory addresses have 32 bit size. If using *multi-level paging*,

      i. determine how many levels of paging are required, and describe the structure of the virtual addresses (purpose, position and size of its bit fields); (1p)

      ii. explain (annotated figure) how in this case the physical address is calculated by multi-level paging from a virtual address; (1p)

      iii. show how a TLB can be used to accelerate address calculation; (1p)

      iv. calculate the expected time for a paged memory access if a physical memory access costs 100ns on average and a TLB access costs 5ns, and the TLB hit rate is 90%. (0.5p)

   (b) How can segmentation and paging be combined? (1p)

   (c) Characterize an important property of pages that are particularly suitable to choose as a victim when freeing a used frame in order to reduce page replacement overhead. Explain why. What kind of data structure could help with this choice? (1.5p)

   (d) What is *thrashing* in a virtual memory system? How does it occur? And what can be done about it? (2p)

7. (5 p.) **File systems**

   (a) What is/are the main (technical) purpose(s) of *opening* a file?
   What kernel data structures does the `open()` system call manipulate and how, and what does it return? (2p)

   (b) Is the file allocation method *indexed allocation* prone to *external* fragmentation? Give a short explanation. (1p)

   (c) Name and describe one disk scheduling algorithm of your choice (but *not* FIFO/FCFS, which is a trivial one), and describe its (expected) effect on disk throughput and disk access latency compared to FIFO/FCFS. (2p)

8. (1.5 p.) **OS Structures and Virtualization**

   (a) What does a *hypervisor* (also known as *virtual machine monitor, VM implementation*) do?

   Illustrate your answer with a commented figure that shows where the hypervisor is positioned in the system software stack and with which other system entities it interacts. (1.5p)

9. (1 p.) **Protection and Security**

   (a) Given a memory system with segmentation. Which access right(s) is/are appropriate for a process's stack segment, and which one(s) is/are not? Motivate your answer. (1p)

Good luck!