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

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem /
Concurrent programming and operating systems

10 jun 2014, 14:00–18:00, TER2

Jour: Christoph Kessler (070-3666687, 013-282406), on travel;

Erik Hansson, course assistant (076-8467951), 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.
- **How much to write?** No general policy, but as a rule of thumb: Questions for 0.5p can typically be answered properly in a single line. Correct and concise answers to questions for 1p usually require a few lines. Code and figures should be commented properly.
- Grading: U, 3, 4, 5. The preliminary threshold for passing is 20 points.

1. (5.5 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) 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)

   (c) The *system call API* (application programming interface) is a software abstraction for invoking OS functionality.

   Name one kind of technical details that it abstracts from. (0.5p)

   Why is such abstraction important for application programming? (0.5p)

   (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) Name and describe one common CPU scheduling technique for a *multiprocessor system*. (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 multithreaded (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) Explain what a *readers-writers lock* is, how it is used, and in what situations it could provide better system performance than ordinary mutex locks, and why. (2p)

   (b) Explain (and illustrate with an example scenario) how mutual exclusion synchronization can interfere with priority based scheduling. (2p)

5. (4 p.) **Deadlocks**

   (a) What is the difference between *deadlock* and *starvation*? (1p)

   (b) Consider the following pseudocode:

```
mutex_lock_t l1, l2, l3;

void T1( void )
{
 mutex_lock( &l1 );
 mutex_lock( &l2 );
 ...
 mutex_release( &l2 );
 mutex_release( &l1 );
 mutex_lock( &l3 );
 ...
 mutex_release( &l3 );
}

void T2( void )
{
 mutex_lock( &l2 );
 mutex_lock( &l3 );
 ...
 mutex_release( &l3 );
 mutex_release( &l2 );
}

void T3( void )
{
 mutex_lock( &l3 );
 mutex_lock( &l1 );
 ...
 mutex_release( &l1 );
 mutex_release( &l3 );
}

void main ( void )
{
 initialize mutex locks l1, l2, l3;
 create 3 threads that execute T1(), T2(), T3() respectively
}
```

   There are 3 threads that concurrently execute the functions `T1`, `T2` and `T3` respectively, which need to acquire and release mutual exclusion locks in order to perform their work (...).

   Is this program deadlock-free?

   If yes, give a formal argument why.

   If not, give a formal argument why, and a counterexample. (2 p)

(c) Most current operating systems do not implement the Banker's algorithm for deadlock avoidance but instead shift this task to the application programmer. Name 2 limitations of the Banker's algorithm that are the main reason for this. (1p)

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

   (a) Consider a page-based virtual memory system with a page size of 2048 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); (1.5p)
      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) Explain how paging supports sharing of memory between processes. (1p)

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

   (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 information is usually contained in a *file control block* (FCB)? (At least 4 different items are expected) (1p)

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

   (c) Describe one technique to extend indexed allocation for large files. (1p)

   (d) 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. (2 p.) **OS Structures and Virtualization**

   (a) What is the main idea of the *microkernel* approach to OS structuring, what is its main advantage and what is its main drawback? (2p)

9. (2 p.) **Protection and Security**

   (a) Name two different (software or hardware) measures to prevent *buffer-overflow* vulnerabilities. Explain briefly (1 line each) how they work. (1p)

   (b) How can using *virtual machines* increase the security of a system? (1p)

Good luck!