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

# TENTAMEN / *EXAM*

# TDDB68

## Processprogrammering och operativsystem / *Concurrent programming and operating systems*

9 june 2015, 14:00–18:00, TER1, TER2

**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 8 assignments and 6 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. (7 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) Why do user-level threads (in contrast to kernel-level threads) promote portability of applications? (1p)

   (c) Two main methods for inter-process communication in a computer are *shared memory* and *message passing*.

   For each of them, give a short explanation of how it works and how the operating system is involved, i.e., which important system calls are to be used and what they do.

   Which of the two methods is likely to have less overhead if two processes communicate frequently with each other, and why? (2.5p)

   (d) How does an operating system prevent a process from monopolizing a processor, and what hardware support is required for that? (1.5p)

2. (7 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 | 6 | 5 |
| $P_2$ | 2 | 3 | 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 4 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.

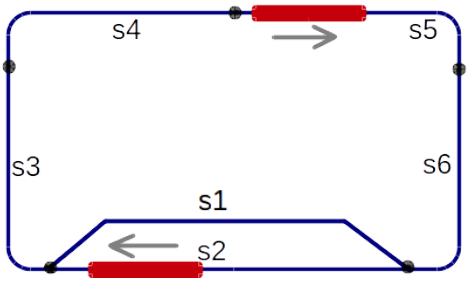
   (v) Priority Scheduling *with* preemption.

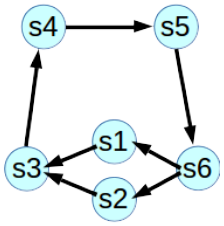
   (b) Strictly priority-based schedulers introduce the problem of process *starvation*. What does this mean? Shortly describe one technique (there exist several ones) that can reduce or completely remove this starvation problem. How does this technique interfere with the priority mechanism? (2p)

3. (6 p.) **Synchronization**

**Train collision avoidance control**

Railway systems need a safety mechanism for preventing train collisions. A common solution consists in partitioning the railway network into $N$ junction[1]-free rail segments each of a certain fixed length (usually, a few kilometers, at least as long as the longest train plus the maximum distance to brake a train down to halt if the next rail segment is not free). See the figure (left) below for a simple example.




Left: A simple railway system with $N = 6$ segments $s1$,...,$s6$, two junctions, and currently two trains circulating in clockwise direction. — Right: The corresponding segment graph.

Assume here for simplicity that:

- all segments are unidirectional;
- between any two junctions there is at least one full-length segment;
- at junctions, the adjacent segments have 2 predecessors or successors respectively.

Hence, the railway system can be modeled as a *directed graph* with the segments as nodes, where edges connect segments to their successor segments, see the figure (right).

At any time there shall be at most one train within a segment, and a train can only proceed to its next segment if it is free. Trains may proceed at arbitrary speed and remain within their current segment for an arbitrary amount of time, e.g. when halting in a train station.

The central railway control server stores the graph of segments in (shared) memory and uses (e.g.) a boolean `flag` in the node data structure to indicate if the segment is currently free or occupied. Before entering its next railway segment, a train asks the control server for permission, by having a server thread call

```
void request_entry_to_segment( struct node *segm )
{
  while (segm->flag == 1)  // 1 means OCCUPIED
     ;
  segm->flag = 1;
}
```

If the desired segment `segm` is marked free, the control server marks it as occupied and grants access to the requesting train by returning from the call. If the segment is occupied, the call blocks and thus the train must halt in its current segment and wait until the call eventually returns to signalize that the next segment is now ready for entering.

Once the train proceeded to its next segment and has left the previous one, it signalizes this to the control computer by having a server thread call

[1]junction = *växel* in Swedish, a point where a railway track branches into two, or two tracks join into one.

```
void indicate_exit_from_segment( struct node *segm )
{
  segm->flag = 0;  // FREE
}
```

on the previous segment `segm`.

In order to handle multiple incoming requests from multiple trains on a large railway network concurrently, the control computer uses a multithreaded program. It is your job to make the control program race-free (and, if possible, also deadlock-free).

(a) Give a simple, concrete example scenario to show that, without any further synchronization, race conditions are possible that may lead to a train collision (a segment being double-booked). (1p)

(b) Which hardware primitive, if available on the control server, could be suitably used here for synchronization to avoid race conditions, and why? Show the resulting pseudocode for `request_entry_to_segment` and `indicate_exit_from_segment`. (2p)

(c) Which software construct could be used for synchronization to avoid busy waiting? Show the resulting pseudocode for `request_entry_to_segment` and `indicate_exit_from_segment`. (2p)

(d) For either (b) or (c), could a deadlock occur with your solution? If yes, give an example scenario. If not, explain why it is not possible. (1p)

4. (8 p.) **Memory management**

(a) Consider a page-based virtual memory system with a page size of $2^{11} = 2048$ bytes where virtual memory addresses have 32 bit. 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 1ns, and the TLB hit rate is 90%. (0.5p)

(b) Explain how the possibility of sharing memory pages among multiple processes can help to speed up the start-up of a child process at a `fork` system call. (1.5p)

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

(d) Given a virtual memory system with 4 page frames, how many page faults occur with the *Least-Recently Used* replacement strategy when pages are accessed in the following order:

1, 2, 3, 4, 5, 1, 3, 4, 2, 3, 1, 5, 4.

(Justify your answer. Just guessing the right number is not acceptable.) (1.5p)

(e) How can the program design affect the performance on a system with virtual memory? (1.5p)

5. (3 p.) **Deadlocks**

   (a) 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)

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

   (a) Does a *soft link* to the file `exam.pdf` still work after the command `mv exam.pdf archive/exam.pdf`? Why or why not? (1p)

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

   (c) Where is the FCB (file control block) contents stored after a file has been opened? (0.5p)

   (d) What is the basic idea and motivation of the Unix *inode* structure? (1.5p)

   (e) Sometimes it may become necessary to run a file system consistency check. What does "inconsistency" of a file system mean, and what could possibly have caused it? (1p)

7. (2 p.) **OS Structures and Virtualization**

   (a) Give one of the main disadvantages of strict layering (with more than just very few layers) in operating systems. (0.5p)

   (b) 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)

8. (2 p.) **Protection and Security**

   (a) The *Heartbleed* bug discovered 2014 in OpenSSL was a so-called *buffer-overread* vulnerability. What is a buffer-overread vulnerability, and how could an attacker benefit from exploiting such a vulnerability? (principle only, no details of OpenSSL) (1p)

   Given a segmented memory system, could it be prevented by careful setting of segment access rights? Explain why or why not. (1p)

Good luck!