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

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem / *Concurrent programming and operating systems*

8 jun 2016, 14:00–18:00, TER3

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 7 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) Given the following (Unix) C program:

```
#include <stdio.h>
#include <unistd.h>

int main()
{
   fork();
   fork();
   fork();
   printf("Hello\n");
   return 0;
}
```

   How often is `Hello` printed to the standard output when executing this program? Explain your answer carefully (just guessing the right number gives no points). (1.5p)

   (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 a computer system prevent a user process from executing privileged instructions? What hardware support is required for that? (1p)

2. (5 p.) **CPU Scheduling**

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	8	5
P_2	2	4	4
P_3	4	1	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 3 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;

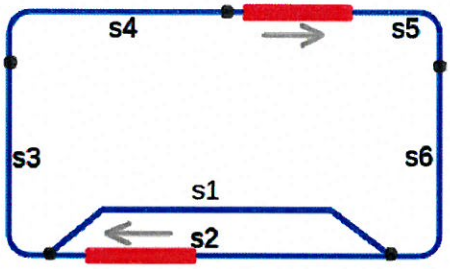
(iv) Priority Scheduling *without* preemption.

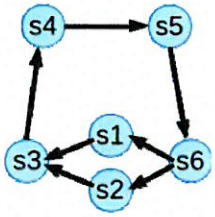
(v) Priority Scheduling *with* preemption.

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. (7 p.) **Memory management**

(a) For a paging system with a page size of N bytes, what is the maximum amount of internal fragmentation that can occur when allocating memory for a process? (0.5p)

(b) Consider a page-based virtual memory system with a page size of $2^9 = 512$ bytes where virtual memory addresses have 30 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. When using a TLB to accelerate address calculation, 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%. (1p)

(c) Explain how paging supports sharing of memory between processes. (1p)

(d) Characterize an important property of pages that makes them particularly suitable to choose as a victim when freeing a used frame in order to reduce page replacement overhead. Explain why. (1p)

(e) Why is it useful for a virtual memory system to be able to estimate the current *working set sizes* of all executing processes? (1p)

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

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

   (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( &l1 );
 mutex_lock( &l3 );
 ...
 mutex_release( &l3 );
 mutex_release( &l1 );
}

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)

6. (5 p.) **File systems**

   (a) Does a *hard 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) Can, in principle, the same file be opened by multiple processes?
   If yes, explain (draw a commented figure) how the internal data structures for opened files in the operating system provide this possibility.
   If not, explain why it is not possible. (2p)

   (d) Why is the management of unused disk space by the file system more complicated with contiguous file allocation? (1p)

7. (2 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. (2p)

8. (4 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)

   (b) Explain the main differences between a computer *virus* and a *worm*. (1p)

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

Good luck!