Försättsblad till skriftlig tentamen vid Linköpings Universitet

	(fylls i av ansvarig)
Datum för tentamen	13 jan 2012
Sal	TER2
Tid	14-18
Kurskod	TDDB68
Provkod	TEN1
Kursnamn/benämning	Processprogrammering och operativsystem
Institution	IDA
Antal uppgifter som ingår i tentamen	8
Antal sidor på tentamen (inkl. försättsbladet)	6
Jour	Erik Hansson (kursassistent), 013-281467 (vid behov: Christoph Kessler, bortrest, 0703-666687)
Besöker salen ca kl.	16:00
Examinator/kursansvarig	Christoph Kessler, IDA
Kursadministratör (namn + tfnnr + mailadress)	Gunilla Mellheden, 013-282297 e. 0705-979044, gunme@ida.liu.se
Tillåtna hjälpmedel	Engelsk ordbok, miniräknare
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	

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

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem /
Concurrent programming and operating systems

13 jan 2012, 14:00–18:00 TER2

Jour: Erik Hansson, course assistant (013-281467), visiting ca. 16:00
Christoph Kessler (on travel, 070-3666687)

Hjälpmedel / *Admitted material:*

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

General instructions

- This exam has 8 assignments and 5 pages, including this one.
  Read all assignments carefully and completely before you begin.
- It is recommended that you use a new sheet of paper for each assignment, because they will be corrected by different persons.
  Sort the pages by assignment, number them consecutively and mark each one on top with your exam ID and the course code.
- 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.
  Students in international master programs and exchange students will receive ECTS grades. Due to the anonymization of written exam correction, ECTS grades will be set by one-to-one translation from swedish grades (5=A, 4=B, 3=C, U=FX), according to the regulations by Linköping university.

1. (4.5 p.) **Interrupts, processes and threads**

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

   (b) 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)

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	5	4
P_2	1	7	2
P_3	3	2	5
P_4	9	3	3
P_5	10	1	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.

3. (7 p.) **Synchronization**

   A *barrier synchronization* is a function that does not return control to the caller until all p threads of a multithreaded process have called it.

   A possible implementation of the barrier function uses a shared `counter` variable that is initialized to 0 at program start and incremented by each barrier-invoking thread, and the barrier function returns if `counter` has reached value p.

   We assume that p can be obtained by calling a function `get_nthreads()`, that load and store operations perform atomically, and that each thread will only call the barrier function once.

The following code is given as a starting point:

```
static volatile int counter = 0;   // shared variable

void barrier( void )
{
  counter++;
  while (counter != get_nthreads())
     ;  // busy waiting
  return;
}
```

(a) Show by an example scenario with $p = 2$ threads (i.e., some unfortunate interleaving of thread execution over time) that this implementation of `barrier` may cause a program calling it (such as the following) to hang. (0.5p)

```
void main( void )
{
 ... // create p threads
 ...
 barrier();
 ...
}
```

(b) Identify the critical section(s) in this implementation, and use a *mutex lock* to protect the critical section(s). (Show the resulting C code). (1.5p)

(c) Can you guarantee correct execution without using mutex locks, by using atomic *fetch-and-add* instead? If yes, show how to modify the code above. If not, explain why. (1.5p)

(d) Suggest a suitable way to extend the (properly synchronized) code to avoid busy waiting. Show the resulting pseudocode (1.5p)

(e) The barrier solution as given above can only be used once in a program execution (why?). Suggest a way to generalize the above solution (properly synchronized) so that it works even if there occur *several* barrier synchronizations in the same program, such as in

```
void main( void )
{
 ... // create p threads
 ...
 barrier();
 ...
 barrier();
 ...
}
```

Explain your solution, and motivate why it will not hang. (2p)

4. (6 p.) **Deadlocks**

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

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

(c) 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, 4 processes $P_1 ... P_4$ 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	1 2 1 0	2 2 1 3
P_3	1 0 0 2	2 2 0 3
P_4	1 0 0 2	2 0 1 3

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

(ii) In the situation given above, process P_2 now asks for 1 instance of D. Is it safe to grant the request? Why or why not? (calculation) (1.5p)

5. (5.5 p.) **Memory management**

(a) Which kind of fragmentation (external or internal) can occur in contiguous memory allocation? Explain your answer. (1p)

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

(c) 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, 1, 5, 6, 2, 5.

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

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

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

   (a) Can a file system create a *hard link* to a file in a mounted file system? Justify your answer! (1p)

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

   (c) Describe one case where the file system is *not* an appropriate abstraction for secondary storage, and explain why. (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 performance and disk access latency compared to FIFO/FCFS. (2p)

7. (3.5 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)

   (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. (3.5 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) Unix uses *password salting* in user authentification. What does that mean, and what is its purpose? In particular, which security problem can be (partly) avoided by salting? (1.5p)

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

Good luck!