Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	15 jan 2011
Sal	U10, U11
Tid	08-12
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/Kursansvarig	Christoph Kessler
Telefon under skrivtid	0703-666687
Besöker salen ca kl.	10:00
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*

15 jan 2011, 08:00–12:00 U10, U11

Jour: Christoph Kessler (070-3666687, 013-282406), visiting ca. 10:00

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.
- 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. (7 p.) **Interrupts, processes and threads**

   (a) Define the terms *process*, *kernel thread* and *user thread*, and explain the differences between them. (3p)

   (b) What is a *process control block (PCB)*?
   What is its purpose?
   What data is contained in the PCB of a single-threaded process? (at least 4 relevant items are expected) (2p)

   (c) What is the purpose of *Direct Memory Access (DMA)*? How is it realized in a computer system, and how is it used? In what cases (condition) does it improve system performance? (2p)

2. (6 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. (1p)

(c) Use a *mutex lock* to protect the critical section(s). (Show the resulting C code). (1.5p)

(d) 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)

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

3. (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	3
P_3	3	2	5
P_4	9	3	2
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 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.

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

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

Consider the Dining Philosophers problem as discussed in the lectures, where each philosopher first tries to obtain the chopstick to her left. Describe how each of the four conditions applies in this scenario, thereby showing that a deadlock could occur. (3p)

(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)

(c) How can the occurrence of a deadlock be *detected* when only one instance of each resource type exists in a system? (1p)

5. (9.5 p.) **Memory management**

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

   (b) Several 32-bit processors use *three-level paging* to address the large page table problem, where the page table itself is stored in main memory. (If you don't recall three-level paging, you may answer, with reduced points, this and the following question for one- or two-level paging.)

   Explain three-level paging for a 32-bit virtual address space and a memory page size of 1024 bytes. Show the structure of virtual (logical) addresses and explain (with a well commented drawing) how to compute physical memory addresses. (2.5p)

   (c) As in one-level paging, a TLB can be used with three-level paging to speed up address calculations for frequently accessed pages. Given the average time t_m = 60ns for a physical memory access, time t_{TLB} = 1 ns for a TLB access, and an assumed TLB hit rate of 0.99 = 99%, determine the effective memory access time in three-level paged memory. Explain your calculation carefully. (1.5p)

   (d) Describe the principle of *segmentation*.

   Why would one prefer a segmented memory model instead of a paged memory model?

   And which drawback does segmentation have, compared to paging? (2p)

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

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

6. (4 p.) **File systems**

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

   (b) Describe one case where the file system is *not* an appropriate abstraction for secondary storage, and explain why. (1p)

   (c) What is the purpose of disk scheduling? (1p)

   (d) Name and describe one disk scheduling algorithm of your choice (but *not* FIFO/FCFS, which is a trivial one). (1p)

7. (1.5 p.) **OS Structures and Virtualization**

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

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

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

Good luck!