Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	26 aug 2013
Sal	TER1
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	Christoph Kessler, 0703-666687, 013-282406
Besöker salen ca kl.	09:30
Examinator/kursansvarig	Christoph Kessler, IDA
Kursadministratör (namn + tfnnr + mailadress)	Carita Lilja, 013-281463, carita.lilja@liu.se
Tillåtna hjälpmedel	Engelsk ordbok, miniräknare
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	Se förstasidan / see first page

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

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem / *Concurrent programming and operating systems*

26 aug 2013, 08:00–12:00 TER1

Jour: Christoph Kessler (070-3666687, 013-282406), visiting ca. 09:30

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.

1. (6 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) What is a *thread pool*?
   How can using a thread pool improve performance? (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)

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	7	5
P_2	1	5	3
P_3	5	1	4
P_4	7	3	2
P_5	11	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.

   (b) There exist several strategies for CPU scheduling on a multiprocessor system. One of them is *affinity-based scheduling*. How does it work (basic idea, no details) and what is the motivation for it? (1p)

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

Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released.

The current number of available connections could be kept track of by a counter in shared memory, initialized to N:

```
shared int N_available_connects = N;
```

Basically, each incoming connection request could create a new thread that would execute the following function:

```
connection *void handle_connection_request( void )
{
  while (N_available_connects == 0)
     ;    // wait...
  N_available_connects = N_available_connects - 1;
  return make_new_connection();
}
```

and upon closing a connection, the thread would call the following function:

```
void close_connection( connection *c )
{
   release_connection(c);
   N_available_connects = N_available_connects + 1;
}
```

(a) Show by an example scenario that, without using additional synchronization, the above code can lead to a race condition, resulting in undesired behavior. (0.5p)

(b) Give a properly synchronized solution (C/pseudocode), using either *test&set* or *atomic_swap* instructions.

Explain why your solution guarantees absence of race conditions.

Explain whether your solution is fair or not, i.e., whether it guarantees that no connection request could be postponed indefinitely. (3p)

(c) Suggest (pseudocode/English) how to extend your solution to avoid busy waiting in the case where no connections are available. (1p)

(d) Give a solution using a *monitor* abstraction. Use C or pseudocode notation with appropriate keywords to identify the monitor components, and explain your code. (1.5p)

4. (2 p.) **More synchronization**

(a) Explain how mutual exclusion synchronization can interfere with priority based scheduling. (2p)

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

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

(b) 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, 5 processes $P_1 ... P_5$ 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	0 0 0 0	4 3 0 5
P_3	1 2 1 0	2 2 1 3
P_4	1 0 0 2	2 2 0 4
P_5	1 0 0 2	2 0 1 4

I.e., currently, process P_1 holds already one B and one D out of the 2 Bs and 4 Ds that it eventually may need in the worst case, etc. One A is currently still available, etc.

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

(ii) In the situation given above, process P_1 now asks for 1 instance of D, in addition to the B and D that it already has. Is it safe to grant the request? Why or why not? (calculation) (1.5p)

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

(a) Consider a page-based virtual memory system with a page size of 1024 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); (1p)

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

(c) Characterize an important property of pages that are particularly suitable to choose as a victim when freeing a used frame in order to reduce page replacement overhead. Explain why. What kind of data structure could help with this choice? (1.5p)

(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 is/are the main (technical) purpose(s) of *opening* a file?
   What kernel data structures does the `open()` system call manipulate and how, and what does it return? (2p)

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

   (c) 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.) **Protection and Security**

   (a) How does memory segmentation support protection? (1p)

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

Good luck!