Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	23 oct 2012
Sal	TER1
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)	7
Jour	Christoph Kessler, 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*

23 aug 2012, 14:00–18:00 TER1

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.
- 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. (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) What is a *process control block* (PCB)? What is its purpose? And what data does it contain (at least 4 relevant items are expected)? (2p)

   (c) The *system call API* (application programming interface) is a software abstraction for invoking OS functionality.
   Name one kind of technical details that it abstracts from. (0.5p)
   Why is such abstraction important for application programming? (0.5p)

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 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;

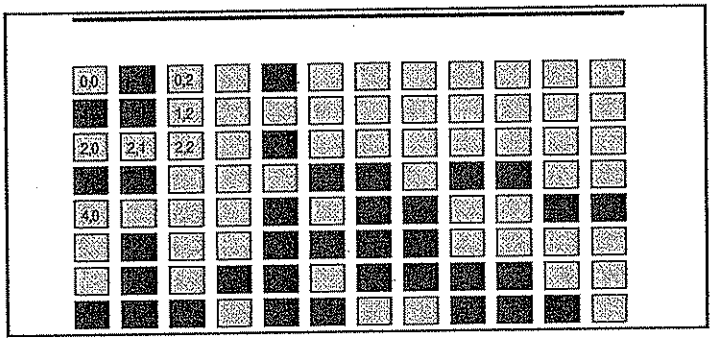
   (iii) Shortest Job First *without* preemption;

   (iv) Priority Scheduling *without* preemption.

   (v) Priority Scheduling *with* preemption.

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

Cinema seat reservation system



Once upon a time, a small cinema with a single ticket counter used a seat reservation program for a (for simplification) single movie presentation per day, with the following (simplified) core data structures and functions:

```
const int nrows = 8;              // Number of rows of seats
const int seats_per_row = 12;     // Number of seats per row
boolean seat_booked[ nrows ][ seats_per_row ]; // 2D array of seats
//  seat_booked[i][j] is 1 iff seat in row i, column j is booked
```

Initially, all seats are free. This is fixed by the following function, which is executed before the cinema ticket counter is opened every day:

```
void initialize_seats ( )
{
  for (int i=0; i<nrows; i++)
     for (int j=0; j<seats_per_row; j++)
        seat_booked[i][j] = 0;   // free
}
```

Before reserving a seat for a new visitor, the clerk at the ticket counter queries the current status of free and booked seats (i.e., the contents of `seat_booked`), which is displayed graphically (as shown above):

```
void show_seat_booking_status ( )
{
  for (int i=0; i<nrows; i++)
     for (int j=0; j<seats_per_row; j++)
        display_status( i, j, seat_booked[i][j] );
}
```

After asking the customer for his/her preferences, the clerk chooses and books a seat that was displayed as free, which is done by the following function:

```
boolean book_seat ( int i, int j )
{
  if (i<0 || i>=nrows || j<0 || j>=seats_per_row) {
     error_message("no such seat");  return 0;
  }
  if (seat_booked[i][j]) {
     error_message("seat already booked");  return 0;
  }
  seat_booked[i][j] = 1;
  return 1;  // successful - can now print the ticket.
}
```

As time goes by, the number of visitors grows and the cinema moves to a building with a larger room (= larger values for `nrows` and `seats_per_row`). To reduce queueing at the ticket counter, a second ticket counter is added, and even a web interface for self-booking is created. All these become additional clients of the seat reservation program, which now will run on a dual-core (!) server running a modern operating system with preemptive thread scheduling that supports semaphores and mutual exclusion locks. The revised seat reservation program shall be multithreaded, such that the above data structures are shared and several `book_seat` calls may be issued concurrently. It is your task to make the seat reservation program thread-safe.

(a) Identify the critical section(s) in the code above, and suggest a simple, but feasible synchronization method to protect the program against race conditions. Show the (pseudo)code of your revised function(s). Explain why your solution is free of race conditions. (2.5p)

(b) The new server also provides a hardware `atomic_swap` instruction. Suggest an alternative protection mechanism that uses `atomic_swap`, and show the revised code for function `book_seat`. (1.5p)

(c) Suggest a protection method that increases concurrency by allowing bookings of seats in *different* rows to proceed simultaneously. (Just the solution idea, no complete code). (1p)

(d) Small groups of persons, e.g. couples, usually prefer to sit next to each other. We assume that seats can only be reserved one by one using the (properly synchronized) `book_seat` routine as above. For example, for booking two adjacent seats in the *same row*, simply calling `book_seat` for the two desired seats in sequence may lead to problems – why? (example scenario) (1p).

What would be a suitable programming abstraction (technical term) for implementing an operation that either books 2 adjacent free seats in the same row together or none at all? (1p)

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 segmented virtual memory supports sharing of memory segments 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, 3, 1, 5, 4.

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

(d) What is *data access locality*?
How can it be approximated for a running process?
Why is good data locality an important property of programs that run on a system with virtual memory?
How can the operating system use this information to improve system performance? (2p)

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

   (a) What is the difference between hard links and soft (symbolic) links? (1p)

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

   (c) Describe one technique to extend indexed allocation for large files. (1p)

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

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

7. (2.5 p.) **OS Structures and Virtualization**

   (a) Define the term *layered operating system* carefully. (1p)

   (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. (4 p.) **Protection and Security**

   (a) What is an *access control list (ACL)*? What does it contain, and how is it used? (1p)

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

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

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

Good luck!