Försättsblad till skriftlig tentamen vid Linköpings Universitet

	(fylls i av ansvarig)
Datum för tentamen	27 aug 2011
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/Kursansvarig	Christoph Kessler
Telefon under skrivtid	0703-666687
Besöker salen ca kl.	16:00
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

27 aug 2011, 14:00–18:00 U10, 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.
- 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. (6.5 p.) **Interrupts, processes and threads**

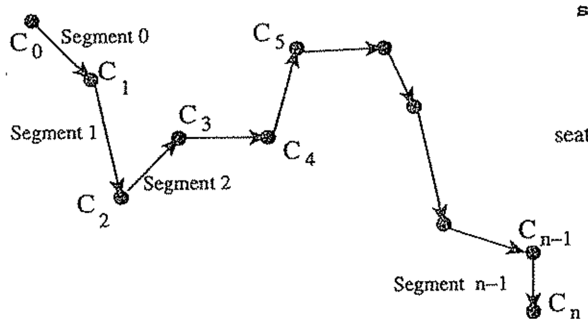
   (a) Define the terms *process*, *kernel thread* and *user thread*, and explain the differences between them. (3p)

   (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 a *thread pool*?
   How can using a thread pool improve performance? (1.5p)

2. (7.5 p.) **Synchronization**

   A train with S seats goes from city C_0 to city C_n with stops in C_1, C_2, ..., C_{n-1} (see the figure). *Segment* i is the direct connection between cities C_i and C_{i+1}, for $0 \leq i \leq n-1$.



seat:

Segment →

seat ↓	0	1	2	3	4		...			i		...			n−1
0	0	1	1	1	1	0	0	1	1	1	0	1	0	1	1
1	1	1	0	1	1	1	1	1	0	0	1	1	1	1	1
2	1	0	1	1	0	1	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	1	1	1	0	0	1	0	0	0	0	1	1	1
⋮															
k															
⋮															
S−1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0

The railway company allows to book seats for any partial contiguous subsequence of segments $i...j$ with $0 \leq i \leq j \leq n$. Booked seats can also be canceled.

For this purpose, a server program had been written. As main data structure representing the S seats over all n segments, a two-dimensional integer array `seat`$[S][n]$ is used, where `seat`$[k][i]$ is 1 if seat k is booked for the segment i, and 0 otherwise (see the figure). Initially, all array entries are set to 0 before any booking requests are accepted for processing.

The following C function implements a first-fit algorithm that searches for a free seat from segment i to j (precondition: $0 \leq i \leq j < n$), and either books and returns the booked seat number if one was available, or returns -1 otherwise:

```
int book ( unsigned int i, unsigned int j )
{
 int k, t;
 for (k=0; k<S; k++) { // try all seats
    int found_k = 1;
    for (t=i; t<=j; t++)
       if (seat[k][t]) {  // if seat k is taken anywhere in i...j,
          found_k = 0;    // we cannot book it.
          break;          // try next k
       }
    if (found_k) {        // seat k was available for i...j:
       for (t=i; t<=j; t++)  // now book it
          seat[k][t] = 1;
       return k;
    }
 }
 // if we arrive here, no seat was available for i...j:
 return -1;
}
```

Canceling a seat for segments $i...j$ (again with precondition $0 \leq i \leq j < n$) is done by the following routine:

```
void cancel ( unsigned int k, unsigned int i, unsigned int j )
{
 int t;
 for (t=i; t<=j; t++)
    seat[k][t] = 0;
}
```

The railway company has now acquired a new multiprocessor server. The entire seat booking program (not shown) is being multithreaded so that it can concurrently process multiple requests for booking or canceling that come in from various clients. The above `seat` data structure is to be held in (shared) memory, and it is your job to make the accesses to it (i.e., functions `book` and `cancel`) thread-safe, which means that concurrent calls do not lead to erroneous behavior such as double bookings of a seat.

(a) Give a concrete example that demonstrates how, without proper synchronization, two concurrent calls to `book` could lead to a double booking of the same seat. (1p)

(b) Identify the critical section(s) in the functions `book` and `cancel`. (1.5p)
(You can assume that load and store instructions are atomic.)

(c) Protect the code against race conditions with a single mutex lock. (1p)

(d) Assume that the rate of incoming concurrent calls to `book` and `cancel` is very high. Design a more fine-grained synchronization mechanism for the above data structure (i.e., try to make the critical section(s) short). Show the resulting code.
Explain how this modification can improve system throughput in comparison to a coarse-grained synchronization approach. (2.5p)

(e) What is a *reader-writer lock*?

Under what conditions are reader-writer locks more suitable than ordinary mutual exclusion locks in order to increase system throughput? (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	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 4 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. (4.5 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) 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( &l3 );
 mutex_lock( &l1 );
 ...
 mutex_release( &l1 );
 mutex_release( &l3 );
}

void main ( void )
{
 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. (1.5 p)

5. (8 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) Explain how segmented virtual memory supports sharing of memory segments between processes. (1p)

   (e) 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)

6. (4.5 p.) **File systems**

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

   (b) Where is the FCB contents stored after a file has been opened? (0.5p)

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

7. (1 p.) **OS Structures**

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

8. (3 p.) **Protection and Security**

   (a) What is a *Trojan Horse attack*? Explain the term in general and give one example scenario. (2p)

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

Good luck!