Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	13 jan 2010
Sal	U14
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*

13 jan 2010, 08:00–12:00 U14

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 for each assignment. Number all your sheets, and mark each sheet 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 the central administration where appropriate, following the Linköping university rule for translation from swedish grades (5=A, 4=B, 3=C, U=FX).

1. (7.5 p.) **Interrupts, processes and threads**

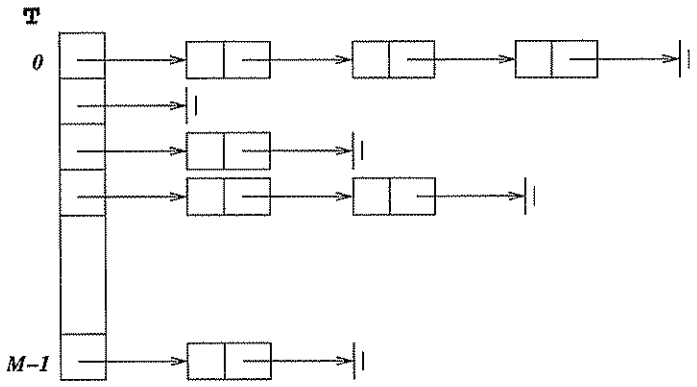
   (a) Define the terms *process*, *kernel thread* and *user thread*, and explain the differences between them. (3p)

   (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)
   What data is contained in a *thread control block (TCB)* of a thread in a multithreaded process? (at least 2 relevant items are expected) (0.5p)

   (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. (5.5 p.) **Synchronization**

   Consider an ordinary *hash table* T of size M designed for a single-threaded environment. A hash function h that maps data items to the range $\{0, ..., M-1\}$ is given. Data items u, v hashed to the same hash value $h(u) = h(v) = i$ are chained in a singly-linked list, such that list $T[i]$ contains all items with hash value i (see the picture). Each list item contains a data element and a `next` pointer to the next item in the list (NULL for the last element in a list). Each $T[i]$ points to the first item in its list (or $T[i]$ is NULL if the ith list is empty), $i = 0, ..., M-1$.



   A sequential implementation is given as follows. List elements are represented by the following data type:

```
struct htelem {
    struct dataitem *item;
    struct htelem *next;
};
```

   The operation `insert`(T, u) inserts a data item u into hash table T at the beginning of list $T[h(u)]$:

```
void insert ( struct htelem *T[], struct dataitem *u )
{
  int i = h(u);
  struct htelem *e = (struct htelem *) malloc(sizeof(struct htelem));
  e->item = u;
  e->next = T[i];
  T[i] = e;
}
```

The boolean operation `lookup`(T, u) checks whether item u is currently stored in T or not:

```
int lookup ( struct htelem *T[], struct dataitem *u )
{
  int i = h(u);
  struct htelem *p;
  for (p = T[i]; p!=NULL; p = p->next )
     if (equal( u, p->item ))
        return 1;
  return 0;
}
```

where the boolean function `equal`(u, v) tests for equality of two data items u and v.

(a) Give an example scenario of a *race condition* with two threads that concurrently insert items into an unprotected shared hash table on a multi-tasking single-processor system with preemptive scheduling. I.e., give two different interleavings of the execution of two threads that lead to incorrect behavior, possibly even to a corrupted data structure or a run-time error. (1p)

Your task will now be to make the hash table implementation thread-safe to allow a hash table stored in shared memory to be accessed properly by multiple threads:

(b) Identify the critical section(s) in your pseudocode and suggest how to protect them properly against race conditions by using a single mutex lock. Show the revised (pseudo) code. (2p)

(c) What is the disadvantage of a single-lock solution if there are many threads accessing T frequently, and why? (0.5p)

Describe a multi-lock approach that solves that problem. Why is the multi-lock approach better? (1p)

(d) Threads executing an `insert` operation modify the status of T, while threads executing a `lookup` do not change it. Assume that `lookup` operations occur much more often than `insert` operations. Suggest a lock-based protection technique that exploits this fact to improve the degree of concurrency compared to your previous solution. (No details, no code. Just give the technical term and the basic idea how it works.) (1p)

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	2	7	5
P_3	4	2	3
P_4	7	3	2
P_5	8	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. (2p)

(b) You are given a system with 4 types of resources, A, B, C and D. There are 3 instances of A, 5 instances of B, 1 instance of C and 6 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 0 0 2	2 2 0 3
P_3	1 2 1 0	2 2 1 3
P_4	0 0 0 2	1 0 1 4

(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 B and 1 of D. in addition to the B and D it already has. Is it safe to grant the request? Why or why not? (calculation) (1.5p)

5. (8 p.) **Memory management**

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

   (b) 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, 2, 1, 3, 6, 1, 2, 5.

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

   (c) What is *Belady's anomaly* in page frame allocation for virtual memory?

   Give a simple argument that proves that an *optimal* page replacement strategy can never have Belady's anomaly. (2p)

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

   (e) What is *data access locality*, and why is it an important property of programs that run on a system with virtual memory? (1.5p)

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

   (a) Does a *hard link* to the file `exam.pdf` still work after the command `mv exam.pdf archive/exam.pdf`? Why or why not? (1p)

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

   (c) What is the basic idea and motivation of the Unix *inode* structure? (1.5p)

   (d) How does the file system implementation keep track of unused disk space? Sketch one possible technique for free-space management. (1p)

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

   (a) What are the two main disadvantages of strict layering (with more than just very few layers) in operating systems? (1p)

   (b) What is the main idea of the *microkernel* approach to OS structuring, and what is its main drawback? (1.5p)

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!