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

TENTAMEN / *EXAM*

TDDB68

Processprogrammering och operativsystem / *Concurrent programming and operating systems*

21 oct 2010, 14:00–18:00 TER3, TER4

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 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. (6 p.) **Interrupts, processes and threads**

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

   (b) Given the following (Unix) C program:

```
#include <stdio.h>
#include <unistd.h>

int main()
{
   fork();
   fork();
   fork();
   printf("Hello\n");
   return 0;
}
```

   When executed, how many times will `Hello` appear on the standard output? Explain your answer! (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.

   Why is such abstraction important for application programming? (1p)

2. (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)

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	5	2	1
P_4	9	3	5
P_5	10	1	2

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 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 0 0 2	2 2 0 3
P_3	1 2 1 1	2 2 1 3
P_4	0 0 0 1	1 0 1 3

   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.

   (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 one more instance of D, in addition to the one B and one 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 paged memory system with a translation lookaside buffer (TLB). What is the effective memory access time if the physical memory access time is 90ns, a TLB lookup takes 10ns and the average TLB hit rate is 90%? Explain your calculation. (1p)

   (c) Paging for large virtual address spaces and small page sizes leads to the large-page-table problem. Explain *one* technique how this problem can be solved. Be thorough! (2.5p)

   (d) 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, 2, 1, 5, 6, 7, 2, 5.

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

   (e) What is *thrashing* in a virtual memory system? How does it occur? And what can be done about it? (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) Name and shortly describe a file allocation method that avoids external fragmentation. (1p)

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

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

7. (3 p.) **OS Structures and Virtualization**

(a) There are two main disadvantages of using strict layering (with more than just very few layers) for structuring operating systems. Explain one of them thoroughly. (1p)

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

Draw also a commented figure that shows where the hypervisor is positioned in the system software stack and with which other system entities it interacts. (2p)

8. (2.5 p.) **Protection and Security**

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

(b) What is a *Trojan Horse attack*? Explain the term in general and sketch one example scenario. (1.5p)

Lab bonus: If you are first-time registered on TDDB68 in autumn 2010 and have passed (according to webreg) the entire Pintos lab series by 27 october 2010, we will add 4 bonus points to your result in this exam.

Good luck!