TENTAMEN / EXAM

TDDB68
Processprogrammering och operativsystem / 
Concurrent programming and operating systems

TDDB72
Processprogrammering, operativsystem och realtidsystem / 
Concurrent programming, operating systems and real-time 
systems

19 dec 2005, 14:00–18:00,

Jour: Christoph Kessler (070-3666687, 013-282406)

Hjälpmedel / Admitted material:
– Engelsk ordbok / Dictionary from English to your native language;
– Miniräknare / Pocket calculator

General instructions

• This exam has 7 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 for each assignment. Number all your sheets, 
  and mark each sheet on top with your name, personnummer, 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 grade 3 is 20 points.
- OBS C:are antagna före 2001: Om du vill ha ditt betyg i det gamla betygsystemet (U, G, VG) skriv detta tydligt på omslaget av tentan. Annars kommer vi att använda det nya systemet (U, 3, 4, 5).
1. (5 p.) **Processes and threads**

   (a) Define the three terms *process*, *kernel thread* and *user thread*, and explain the differences between these. (2.5p)

   (b) We discussed different thread models with different relations between user and kernel threads. Describe the models one-to-one, many-to-one and many-to-many, and briefly explain their advantages and disadvantages. (2.5p)

2. (7 p.) **Synchronization**

Consider the following familiar situation of a ticket-based waiting-queue management system handling a set of customers who want to access a single shared service desk. We assume that there is only one service desk, which can only serve one customer at a time, and once a customer got access to the service desk, he/she will be serviced completely. Each customer who wants to access the service desk must first get a *waiting ticket* from a (shared) ticket automaton. After having served a customer, the leaving customer increments a globally visible counter indicating the *currently active ticket number*. Only the customer whose waiting ticket number equals this counter value will be served next. The others have to *wait* until their ticket number becomes active.

This analogy can be used to guide the implementation of so-called *fair locks* that can be used instead of ordinary locks to guard a critical section, providing mutually exclusive access to a shared resource while guaranteeing servicing incoming access requests in FIFO resp. FCFS (first come, first served) order. As you may guess, the customers correspond to threads, and the service desk to a shared resource in a computer system.

   (a) Sketch a realization of this mechanism based on two shared counters, such as

   ```
   shared unsigned int ticket; /* next ticket to be given out */
   shared unsigned int active; /* the currently active ticket */
   ```

   that are initially 0 and incremented according to the scenario above. Write pseudocode (routines) to get a new waiting ticket, to wait for the own ticket number to appear on the active counter, and to advance the active counter. (The shared resource itself needs not be coded explicitly).

   Identify the critical sections in your code. (You may assume for simplicity that a read to an unsigned integer word is an atomic operation.) (2p)
(b) The classical counting semaphore construct with \texttt{wait()} and \texttt{signal()} (or \texttt{P()} and \texttt{V()}) operations, as presented in the lectures and the course book, is based on an integer counter and provides mutual exclusion, but is not immediately applicable to model the counters above. Why not? (1p)

(c) Write a \textit{monitor} solution such that critical sections are protected. (Use pseudocode notation with appropriate keywords to identify the monitor components, and explain your code. The shared resource needs not be coded explicitly.) (2.5p)

(d) Assume that the system supports at most \( P = 100 \) threads that can access this shared resource. (Note that then there can be at most 100 consecutive tickets given out at any time.) Design a suitable set of \textit{condition variables} (how many do you need?) for your monitor solution above that wakes up a thread exactly when its ticket is about to be called up, and extend your monitor solution above to use the condition variables accordingly. (1.5p)

(e) \textbf{Bonus question:} Certain multiprocessors provide an \textit{atomic} fetch-and-increment instruction, whose semantics is defined by the following pseudocode routine:

\begin{verbatim}
unsigned int FetchAndIncr ( shared unsigned int *ps )
{
    unsigned int ret;
    ret = *ps;
    (*ps)++;
    atomic (1 instruction)
    return ret;
}
\end{verbatim}

Assume that, thanks to hardware support, any number of \texttt{FetchAndIncr} accesses (even to the same shared memory location) can execute simultaneously. Suggest how to use the \texttt{FetchAndIncr} instruction to implement a fair lock. What is the advantage of this solution over the monitor solution of question 2c? (+1p)

3. (7 p.) \textbf{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, 4 is lowest priority).

\begin{table}[h]
\centering
\begin{tabular}{llll}
Process & Arrival time & Execution time & Priority (as applicable) \\
\hline
\textit{P}_1 & 0 & 7 & 3 \\
\textit{P}_2 & 2 & 4 & 2 \\
\textit{P}_3 & 3 & 5 & 1 \\
\textit{P}_4 & 5 & 2 & 4 \\
\end{tabular}
\end{table}

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. (5p)

(i) FIFO;

(ii) Round-robin;

(iii) Shortest Job First \textit{without} preemption;

(iv) Priority Scheduling \textit{without} preemption.

(v) Priority Scheduling \textit{with} preemption.
(b) What is a Multilevel-Feedback-Queue scheduler? Describe how it works and the reasons for using such a scheduler in a general-purpose operating system. (2p)

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) You are given a system with 3 types of resources, A, B and C. There are 3 instances of A, 5 instances of B and 6 instances of C. Currently, 4 processes $P_1$...$P_4$ are running, and for each process, the resources currently held and its total maximum resource need for each type are given as follows:

<table>
<thead>
<tr>
<th>Process</th>
<th>Already held</th>
<th>Maximum total need</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$P_1$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

(ii) Process $P_2$ now asks for 1 instance of $B$ and 1 of $C$. Can the request be granted? Why or why not? (1.5p)

(c) Current operating systems do not implement the Banker’s algorithm for deadlock avoidance but instead shift this task to the application programmer. What limitation of the Banker’s algorithm is the main reason for this? (1p)

5. (8 p.) Memory management

(a) What is a translation lookaside buffer (TLB), and what is its purpose in the context of a paged memory system? Draw a diagram and explain in detail how the TLB works. (3p)

(b) What is virtual memory with demand paging and how does it work? What are its advantages and disadvantages? (3p)

(c) Describe the principle of segmentation. Why would one prefer a segmented memory model instead of a paged memory model? (2p)

6. (2 p.) Disk I/O

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

(b) Name and describe one disk scheduling algorithm of your choice. (1p)

7. (5 p.) Protection and Security

(a) What is a buffer-overflow attack? Describe the vulnerability, the factors that contribute to it, and give a scenario of how it can be exploited by an attacker to hijack the control of a running server program. (3p)

(b) Suggest at least one (software or hardware) measure to prevent such buffer-overflow vulnerabilities. Explain briefly how it works. (1p)
(c) Initially, the UNIX password file /etc/passwd used to be readable for all users, but in more recent versions it can only be read by the system administrator, due to security concerns. Why? (1p)

Good luck!