Tentamen i TDDD82 Säkra mobila system (Systemprogramvara)

2019-06-13

- Inga hjälpmedel är tillåtna.
- Kom ihåg att svaren på samtliga uppgifter måste MOTIVERAS, och att motiveringarna skall vara uppställda på ett sådant sätt att det går att följa hur Du tänkt. OMOTIVERADE SVAR GER 0 POÄNG OM INGET ANNAT SÄGS.
- Ansvarig: Mikael Asplund (nåbar på tel. 0700-895827).
- Maxpoäng är 30 poäng. För betyg 3 krävs minst 15 poäng, för betyg 4 krävs 20 poäng och för betyg 5 krävs 25 poäng.

Lycka till!!!

1. **Synchronization** (8p)

Consider a shared stack of positive integers, to be implemented as a shared array of sufficiently large size and a shared stack pointer. The stack operations `push` and `pop` can be called concurrently by multiple threads. When the stack is found to be empty, operation `pop` should return an error code (-1).

(a) Write (unprotected) code for the stack initialization and for the operations `push` and `pop`. (Use C, Java or pseudocode.) (2p)

(b) Describe a contrived scenario with two threads where the concurrent execution of these unprotected operations leads to an unexpected result (i.e., a race condition). (2p)
Identify precisely the variables and statements that could potentially be involved in any race condition (that is, describe the critical section(s)).

(c) You are given a single-processor system where multi-tasking is implemented by a preemptive scheduling algorithm, and where an atomic test-and-set operation is available. Here, there exist actually two different hardware-supported mechanisms to protect critical sections. Name and explain these two variants, and show for each case how your code of part (a) is to be modified to protect its critical section(s) against race conditions. (2p)

(d) Write a *monitor* solution for the shared stack (based on your code above). Use pseudocode notation with appropriate keywords to identify the monitor components, and explain your code. (2p) (2p)

2. **Processes** (7p)

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

(b) Two main methods for inter-process communication in a computer are *shared memory* and *message passing*. (4p)

For each of them, give a short explanation of how it works and how the operating system is involved, i.e., which important system calls are to be used and what they do.

Which of the two methods is likely to have less overhead if two processes communicate frequently with each other, and why?

3. **Deadlocks** (5p)

Consider the following resource allocation problem in a system with 3 resources (R1-R3), and 4 processes (P1-P4). The table indicates the currently allocated resources and in parenthesis the maximum possible demand.

	R1	R2	R3
P1	0 (7)	2 (5)	3 (4)
P2	1 (4)	2 (4)	2 (2)
P3	1 (6)	0 (4)	3 (7)
P4	1 (1)	0 (3)	0 (1)

The currently available resources are: [5, 3, 1]. Use Banker's algorithm to determine if the request [1, 2, 0] from Process P2 should be granted.

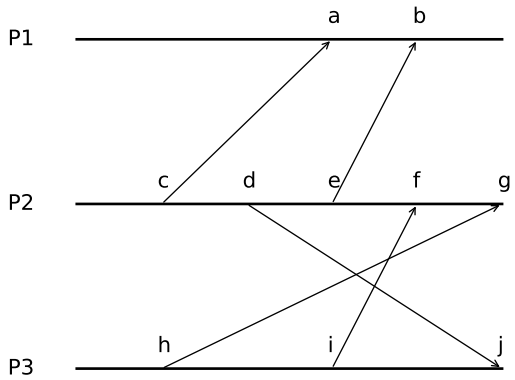
4. **Quality of Service** (5p)

(a) Applications can have varying QoS requirements. Explain what it means if an application is tolerant or intolerant, and give one example of each type of application. (2p)

(b) Describe the main parameters of the token bucket traffic shaping method. (3p)

5. **Distributed systems** (5p)

Consider the three timelines for nodes P1, P2 and P3 in the figure below. The arrows between the lines indicate messages as they are sent and received by the respective nodes.



(a) Use Lamport's logical clock algorithm to set timestamps for the events a-j. (2p)

(b) Let $C(e)$ denote the Lamport clock for event e. For each bullet point below, either identify a pair of events (from events a-j) that match the requirements, or explain why they do not exist. (3p)

- $e_1 \rightarrow e_2$ and $C(e_1) < C(e_2)$
- $e_1 \rightarrow e_2$ and $C(e_1) > C(e_2)$
- $e_1 \nrightarrow e_2$ and $C(e_1) < C(e_2)$

6. **Dependability** (5p)

Use the terminology from IFIP Working Group 10.4 to analyse the fault-error-failure chain in the example below (The Evening Herald, May 30 2019). Classify the fault as permanent/transient/intermittent.

> Computer glitch blamed for chaos on commuter trains
>
> A software upgrade is thought to have caused a signalling fault that affected around 20,000 rail commuters.
>
> Many people were left stranded and waiting for information on Tuesday morning after first hearing about the problem on the radio.
>
> And although the matter has now been resolved, Irish Rail said it couldn't guarantee that it wouldn't happen again.
>
> Company spokesperson Jane Cregan told the Herald that new software designed to increase the network's signalling efficiency did the exact opposite.
>
> As a result, the computer at Central Traffic Control in Connolly "just went blank".
>
> "Every signal controlled by this computer automatically went to red and stopped working,she said.
>
> There were a number of updates carried out in recent days on the system and we think one of them caused this fault to occur."
>
> Ms Cregan added that during the crisis 15 to 20 operators were deployed around the country with manual access to signalling panels.