COS3 lektion 6

Nis Sarup

10. oktober 2010

6 Synchronization

6.1 Background

- Race-conditions are bad.
- For Shared Memory to work, the processes involved needs to be synchronized.

6.2 The Critical-Section Problem

- Each process has a critical section, and only one process can make changes in its critical section at any one time.
- Nonpreemptive kernels do not suffer from race conditions. (at least the kernel-threads do not).

6.5 Semaphores

- A Semaphore is an integer variable accessed only through wait() and signal().
- A semaphore can be binary (0 or 1) or an integer of any size.
- A process do a wait() on the semaphore and if the count goes below zero, the wait command halts it until another process does a signal() and the semaphores counter increases, signaling that a resource have become available.
- This approach produces "busy waiting"or spinlock, which is a bad thing.
- Spinlock can be overcome by adding a waiting queue to the semaphore and any waiting processes can then block while waiting. (Wait() and Signal() must then be able to add/block processes and remove/unblock processes)
- Interrupt must be disabled while wait() and signal() are running lest two processes can call one of them at the same time, thus resulting in a race condition.

- Deadlock is when two, or more processes are wait()ing for each other to signal().
- Starvation is when a process never leaves the semaphores queue.
- Priority inversion is when a high-priority process is made to wait longer than necessary by a lower priority process.

6.6 Classical Synchronization Problem

- Different synchronization problems and solutions.

6.8 Monitors

- A monitor is a high-level construct that ensures that deadlocks and starvation does not occur.

6.9 Atomic Transaction

- Atomic: One block of instructions is executed in its entirety, or not at all.
- After a process executes an (atomic) set of instructions it commits() the changes and the changes are finally saved. If no commit() the changes are rolled back to what they were before the atomic instructions were started.
- Concurrent Atomic Transactions:
  - Serial Schedule: One atomic transaction follows another. (Too restrictive)
  - Locking Protocol: Each transaction locks a data-item. In some cases another transaction may read from it, in others not.
  - Two-Phase Locking Protocol: In the Growing Phase a transaction can obtain locks but mot release any. Vice-versa in the Shrinking Phase.
  - Timestamp-Based Protocols: Use timestamp to determine the order of transactions.