1 Mutual exclusion

Mutual exclusion is a program object that prevents simultaneous access to a shared resource. This concept is used in concurrent programming with a critical section. Processes have to get permission before entering their critical section. (request a lock, hold the lock, release the lock).

1.1 Approaches

- **Software based** Peterson's solution
- **Hardware based** `test_and_set (), swap_and_compare()`
- **Os based** mutexes and semaphores
- **Monitors**: software construct within the programming languages

1.2 Peterson's solution

Petersons solution is a **software based** solution which worked well on older machines Two shared variables are used:

- turn: indicates **which process is next** to enter its critical section
- boolean flag[2]: indicates that a process is **ready** to enter its critical section

Two processes execute in **strict alternation** (can be generalised to multiple processes).

1.3 Disabling interupts

Disable interrupts whilst executing a critical section and prevent interruption (i.e., interrupts from timers, I/O devices, etc.). While disabling interrupts may be appropriate on a single CPU machine. This is **insufficient** on modern multi-core/multi processor machines.
Implement `test_and_set () and swap_and_compare()` instructions as a set of atomic (uninterruptible) instructions

- Reading and setting the variable(s) is done as one complete set of instructions
- If `test_and_set () or compare_and_swap()` are called simultaneously, they will be executed sequentially

They are used in in combination with global lock variables, assumed to be true if the lock is in use. **However** they are hardware instructions and (usually) not **directly accessible** to the user.
Rembember, the OS hides the bare metal from the user. Other disadvantages include:

- Busy waiting is used
- Deadlock is possible, e.g. when two locks are requested in opposite orders in different threads

The OS uses the hardware instructions to implement higher level mechanisms/instructions for mutual exclusion, i.e. mutexes and semaphores

1.4 Mutexes

Mutexes are an approach for mutual exclusion provided by the operating system containing a boolean lock variable to indicate availability. The lock variable is set to true if the lock is available (process can enter critical section), false if not. Two atomic functions are used to manipulate the mutex:

- `acquire()`: called before entering a critical section, boolean set to false.
- `release()`: called after exiting the critical section, boolean set to true again

Functions `acquire()` and `release()` must be atomic instructions. No interrupts should occur between reading and setting the lock. The process that acquires the lock must release the lock (in contrast to semaphores see later)

The key **disadvantage** of mutex locks is that calls to `acquire()` result in busy waiting (although this appears to be OS dependent).Detrimental for performance on single CPU systems.
The key advantages of mutex locks include:

- Context switches can be avoided (short critical sections)
- Efficient on multi-core/multi-processor systems when locks are held for a short time only

Reference section

holder

holder