Critical Section Problem: Synchronisation Solutions Hardware

28 Sep 2018

Critical Section Problem Solutions

- Synchronization Hardware
- Semaphores

Hardware Solution to Critical Sec. Problem

- Hardware solution: Locking mechanism
- support for implementing the critical section code.
  - Protecting critical regions via locks.
- Simple hardware support via “disabling interrupts”
- Idea:
  - Currently process/running code would continue without preemption.
  - Drawback
    - Generally too inefficient on multiprocessor systems:
    - Operating systems using this are not broadly scalable: Response time
- Modern machines provide special atomic (non-interruptible) hardware instructions:
  - Either test memory word and set value at once.
  - Or swap contents of two memory words.

Interrupt Disabling Solution

- On a Uniprocessor system, Interrupts if disabled, while in Critical Section (CS), we cannot interleave execution with other processes that are in Remainder Section (RS)

```
Process Pi:
...
  disable interrupts
    critical section
  enable interrupts
    remainder section
...
```

- On a Multiprocessor: mutual exclusion is not preserved:
  - CS is now atomic but not mutually exclusive (interrupts are not disabled on other processors)
  - For, each processor has its own interrupt vector

Hardware Solution: Special Machine Instructions

- In Hardware, access to a memory location excludes others accesses to that same location
- Hardware solution:
  - Extending this mutual exclusive access to bunch of instructions (accesses: read and write)
  - machine instructions that perform 2 actions atomically (indivisible) on the same memory location (e.g., reading and writing).
- On Multiprocessors: Execution of such an instruction is also mutually exclusive

Solution to Critical Section Problem using Locks

- Solution with a locking possibility:

```
do {
        acquire lock
            critical section
        release lock
            remainder section
} while (TRUE);
```

TestAndSet Synchronization Hardware

- The Boolean function represents the essence of the corresponding machine instruction.
- Test and set (modify) the content of a word atomically
  - Simple boolean type is as following:

```
boolean TestAndSet(boolean *target) {
        boolean rv = *target; // returned value
        *target = TRUE; //set value
        return rv;
}
```

1. Executed atomically.
2. Returns the original value of passed parameter i.e. LOCK var.
3. Set the new value of passed parameter to “TRUE”.

MutEx with TestAndSet: Illustration

```
boolean TestAndSet(boolean *target) {
        boolean rv = *target; // returned value
        *target = TRUE; //set value
        return rv;
}
```

- Shared data:

  boolean lock = FALSE;

  Process P_i

```
do {
    while (TestAndSet(&lock));
        critical section
    lock = FALSE;
        remainder section
} while (TRUE);
```

Hardware Sol. 2: Swap Synchronization

- Atomically swap (exchange) two variables:

```
void Swap(boolean *a, boolean *b) {
        boolean temp = *a;
        *a = *b;
        *b = temp;
}
```

- The procedure represents the essence of the corresponding machine instruction.