Embedded Operating Systems

Multi-threading

- Refer Day13 notes

Synchronization

Synchronization

Mutex

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is similar to "binary semaphore".
- Mutex can be unlocked by the same process/thread, which had locked it.

Semaphore vs Mutex

- S: Semaphore can be decremented by one process and incremented by same or another process.
- M: The process locking the mutex is owner of it. Only owner can unlock that mutex.
- S: Semaphore can be counting or binary.
- M: Mutex is like binary semaphore. Only two states: locked and unlocked.
- S: Semaphore can be used for counting, mututal exclusion or as a flag.
- M: Mutex can be used only for mutual exclusion.

Critical Section

- Section (block) of a code that should be executed by single process at a time. If multiple processes/threads execute it concurrently, then unexpected results may happen.
- Example: Doubly linked list -- add node at position

SUNBEAM INFOTECH

```
// create new node (nn)
// trav till position - 1 (trav)
// get address of next node (temp)
// add node between trav and temp
nn->next = temp;
nn->prev = trav;
trav->next = nn;
temp->prev = nn;
```

- Critical section problem can be solved using mutex.

```
init(mutex);
// ...
lock(mutex);
// ...
// ... critical section
// ...
unlock(mutex);
// ...
```

Spinlock

- Spinlock is harware/architecture based synchronization mechanism.
- Two processes cannot access spinlock simultaneously, in uni-processor or multi-processor environment.
- Semaphore/mutex should not be used in interrupt context (ISR), because ISR should never sleep.
- Semaphore is internally a counter and mutex is a lock. If multiple processes try to use the Semaphore/mutex simultaneously, there may be race condition for Semaphore/mutex itself.

Solution 1

SUNBEAM INFOTECH

- When Semaphore count is incremented (V) or decremented (P), the processor interrupts can be disabled. This will ensure that no other process will preempt P and V operation, and thus no race condition for Semaphore.
- Semaphore P operation:
    - step 1: disable interrupts.
    - step 2: P operation (decrement and block if negative).
    - step 3: enable interrupts.
- Semaphore V operation:
    - step 1: disable interrupts.
    - step 2: V operation (increment and unblock if any process sleeping).
    - step 3: enable interrupts.
- This solution is applicable for uni-processor system. In multi-processor system, if interrupts are disabled, it will disable interrupts of current processor only. The process running on another processor can still access the Semaphore.
- Disabling (masking) interrupts also increases interrupt latencies.

Solution 2

- When Semaphore count is incremented (V) or decremented (P), some hardware level synchronization mechanism should be used to access Semaphore by only one process.
- Spinlock is hardware level synchronization mechanism. Spinlocks are implemented using bus-holding instructions -- test_and_set() kind i.e. only one task can access the bus at a time. The test and set operations are done in same bus cycle i.e. bus remains locked until both operations are completed.
    - Example: ARM7 -- SWP, ARM Cortex -- LDREX, STREX.
- https://developer.arm.com/documentation/den0013/d/Multi-core-processors/Exclusive-accesses

Spinlock working

- Spinlock is a variable -- 0 (unlocked/available) or 1 (locked/busy).
- Spinlock initialization. It is unlocked.

```
lock = 0;
```

- To lock a spinlock: If spinlock is busy, wait (busy wait loop); otherwise lock.

SUNBEAM INFOTECH

```
while(lock == 1)
    ;
lock = 1;
```

- To unlock a spinlock, clear it.

```
lock = 0;
```

ARM7 SWP instruction

- Spinlock implementation

```
spin:
    MOV r1, =lock
    MOV r2, #1

    SWP r3, r2, [r1] ; hold the bus until complete

    CMP r3, #1
    BEQ spin
```

- SWP instruction
  - SWP r3, r2, [r1]
    - r3 = *r1 and *r1 = r2;
- Refer slos book

ARM Cortex LDREX/STREX

SUNBEAM INFOTECH

- Refer: Joseph Yiu

```
spin_lock:                   ; an assembly function to get the lock
    LDR        r0, =Lock_Var
    MOVS       r2, #1            ; use for locking STREX
lock_loop:
    LDREX      r1, [r0]
    CMP        r1, #0
    BNE        lock_loop         ; It is locked, retry again
    STREX      r1, r2, [r0]      ; Try set Lock_Var to 1 using STREX
    CMP        r1, #0            ; Check return status of STREX
    BNE        lock_loop         ; STREX was not successful, retry
    DMB                          ; Data Memory Barrier
    BX         LR                ; Return
```

```
spin_unlock:                ; an assembly function to free the lock
    LDR r0, =Lock_Var
    MOVS r1, #0
    DMB                      ; Data Memory Barrier
    STR r1, [r0]             ; Clear lock
    BX LR                    ; Return
```

Semaphore vs Mutex vs Spinlock

- Spinlock are busy wait (not sleep).
- Can be used in interrupt context.
- Available only in kernel space.

SUNBEAM INFOTECH