Chapter 2

Part 5: Synchronization

Multiple Programs on Single CPU

- Problem: resource sharing
- Examples
  - Updating 64-bit global variable in 32-bit machine
  - Updating global data structure
    - Linked list of free buffers and associated count
  - Printer sharing
  - Reserving airline tickets
- Problem: operations not atomic ("all or nothing")
- Issue of data corruption
- Solution: exclusive access until operation is complete
  - Mutual exclusion, MUTEX

Mutual Exclusion (Mutex)

- Mutex
  - Simple form of synchronization
- Same synchronization problem can occur on (shared memory) multiprocessor systems

Mutual Exclusion (Mutex)

- Only one program running on CPU
  - Problem can occur with interrupts
- Disable interrupt for MUTEX

```
Disable interrupt;
Access shared resource (critical section);
Enable interrupt;
```

- Disabling interrupt is a general solution for Mutex
  - Process switch on timer interrupts
  - Not a good solution because all unrelated programs are also affected

Implementing Mutex

- Minimize the impact of Interrupt Disable

```
Disable interrupt;
if('Access Variable' is 0) {
        Set variable to 1;
        Re-enable interrupts;
        Access the resource (critical section);
        Disable interrupts;
        Set the 'Access Variable' back to 0;
        Re-enable interrupts;
} else {
        Re-enable interrupts;
        /* Try back later; */
}
```

Implementing Mutex

- Better: provide an CPU instruction for synchronization
  - Why is a machine instruction atomic?
- Example: test-and-set (TAS) instruction

```
        // "x" initialized to zero (free to use)
TestAndSet(integer: x) {
    integer y = lock;
    x = 1;
    return y;       // atomic read-and-modify
}     //  two simultaneous TAS ordered by hardware
```

- Mutual exclusion with test-and-set

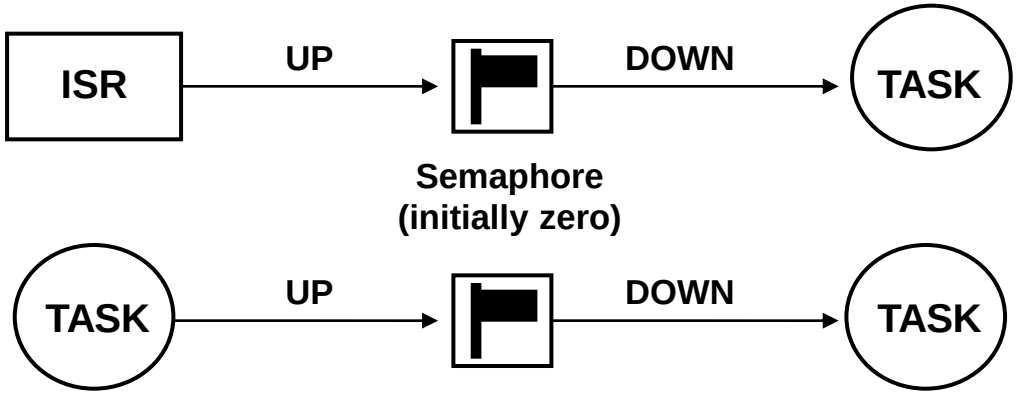
```
while (TestAndSet(lock) == 1);       // spin lock
do critical section;
lock = 0;                            // release lock
```

(Binary) Semaphore: Higher Level

- Dijkstra: what if TAS return failure, how to busy-wait?
  - Associate semaphore for shared resource
- INIT: sem_printer = 1 (i.e., printer is free)
- DOWN (or wait)
  - If available, sem_printer=0 and give exclusive access
    - Entire operation must be atomic (cf. test-and-set)
  - If unavailable, put task in waiting list
- UP (or signal)
  - If no waiting, sem_printer = 1
  - If no, waiting, give exclusive access to waiting task

Inter-Task Communications

- Semaphore (binary and counting)
  - Mutex plus additional data structures and service
- Barrier synchronization
- Producer and consumer: synchronization with semaphore



- Synchronization plus information exchange
  - Place to hold a pointer

Data Exchange Models (IPC)

- Signal
  - Asynchronous interrupt via OS
- Pipe
  - Byte stream, via a file
- Message queue
  - Notion of message
- Shared memory
  - Just share physical memory block

Section 2.10

Parallelism and Instructions: Synchronization

Synchronization

- Two processors sharing an area of memory
  - P1 writes, then P2 reads
  - Data race if P1 and P2 don’t synchronize
    - Result depends of order of accesses
- Hardware support required
  - Atomic read/write memory operation
  - No other access to the location allowed between the read and write

Synchronization

- System programmers build user-level library routines (synchronization library, lock and unlock)
  - Architect supply synchronization instructions
    - A number of alternative formulations
    - Atomically read and modify a memory location
- Hardware primitive can be a single instruction
  - Challenge to put both memory read and write in a single uninterruptable instruction
- An alternative is to use a pair of instructions

Synchronization in MIPS

- Load linked: `ll rt, offset(rs)`
- Store conditional: `sc rt, offset(rs)`
  - Succeeds if location not changed since the `ll`
    - Returns 1 in rt
  - Fails if location is changed
    - Returns 0 in rt
- Example: atomic swap (to test/set lock variable)

```
try: addi $t0,$zero,1  ;copy locked value
     ll  $t1,0($s1)    ;load linked
     sc  $t0,0($s1)    ;store conditional
     beq $t0,$zero,try ;branch store fails
     add $s4,$zero,$t1 ;put load value in $s4
```

Concurrent Programming Example