LSN 8
Synchronization

ECT362 Operating Systems
LSN 8 – Concurrency

• Concurrency Can Occur For . . .
  – Multiple applications
  – Structured applications
  – Operating system structure

• Requires The Ability To Enforce Mutual Exclusion
  – Mutual exclusion is the ability to exclude all other processes from a course of action while one process is granted that ability
LSN 8 – Principles of Concurrency

• Concurrent operational issues
  – Sharing of global resources places constraints on the sequence of events with respect to the shared resource
    • Race conditions
  – Deadlock can be caused by resource allocation strategies
  – Programming errors are hard to debug because results may not be deterministic and are often difficult to reproduce
LSN 8 – Concurrency Issues of OSs

• Operating systems must be able to account for all active processes
• Operating systems must allocate/deallocate resources to processes
• Operating system must protect the data and physical resources of each process against unintended interference by other processes
• Results of a process must be independent of the speed in which it is being executed relative to the speed of execution for other concurrent processes
LSN 8 – Concurrency With Resources

• Competition For Resources
  – Mutual exclusion
    • Two or more processes require access to a non-sharable resource (critical resource)
    • Solutions can cause
      – Deadlock
      – Starvation

• Sharing of Resources
  – Multiple processes “sharing” a resource without knowledge of each other
  – Data coherence
LSN 8 – Mutual Exclusion Requirements

- Enforcement mechanism
- Processes halting in non-critical section must not hinder operations of other processes (deadlock)
- Must not allow processes waiting for critical section to be delayed indefinitely (deadlock/starvation)
- All process must be allowed to access its critical section without delay if there is no contention for it
- No requirement on relative process speeds
- A process remains in its critical section for a finite time
LSN 8 – Software Mutual Exclusion

• Assume that some mechanism already exists for exclusive use of memory
• Dekker’s Algorithm
  – Try 1:
    • Use a global variable as a flag to indicate which process can access the critical section
    • Must wait (polling of flag)
    • Inefficient use of processor’s time
    • Processes are responsible for updating the value of the global flag
LSN 8 – Software Mutual Exclusion

– Try 2:

• Create a Boolean array where each process has its own element
• Each process monitors the other processes’ flag values to see if it is in critical section
• Sets/clears its own flag depending as it enters/leaves the critical section
• Relieves problem of waiting for other process to give “permission” to go into critical area
LSN 8 – Software Mutual Exclusion

– Try 3:
  • Have process set its own flag prior to reading the flag of the other process
    – If other process’s flag is set, wait
    – If other process’s flag is not set, start executing in critical section

– Try 4:
  • Allow processes in “Try 3” to back-off from their desire to gain access to the critical section
  • Process may reset its flag if it does not gain access immediately
  • Problem could occur again if two or more processes have the same back-off philosophy (livelock)
LSN 8 – Software Mutual Exclusion

– Try 5:
  • Combine “Try 1” and “Try 4”
  • Use a global variable to dictate who has the ability to insist on entering the critical area
LSN 8 – Hardware Mutual Exclusion

• Interrupt disabling
  – A process will continue to run until it invokes an operating system service or until it is interrupted
    • Use a kernel primitive to perform the disabling/enabling of interrupts around critical section

• Special machine instructions
  – Ability to perform 2 actions within a single instruction
    • Read/write from/to a memory location
  – Since all operations occur within same machine instruction, there is no concern for interference by other processes during operation
LSN 8 – Semaphores

- OS mechanism to achieve mutual exclusive operation
- Two or more processes can cooperate by means of simple signals, such that a process can be forced to stop at a specified place until it has received a specific signal
- Signaling using a special variable called a semaphore
  - signal(s)
    - Primitive to transmit a signal using semaphores
  - wait(s)
    - Primitive to receive a signal using semaphores
LSN 8 – Semaphores

• Semaphores are integer variables
  – Initialized to non-zero values
  – wait() decrements the semaphore value
  – signal() increments the semaphore value

• Processes waiting on a semaphore are stored in queues
  – Strong semaphores use FIFO queues
  – Weak semaphores have no order to the removal of processes from the queues
Creating & Using Semaphores

```c
struct semaphore {
    int count;
    queueType queue;
}

void semWait(semaphore s) {
    s.count--;
    if (s.count < 0) {
        place this process in s.queue;
        block this process
    }
}

void semSignal(semaphore s) {
    s.count++;
    if (s.count <= 0) {
        remove a process P from s.queue;
        place process P on ready list;
    }
}

/* program mutual exclusion */
const int n = /* number of processes */;
semaphore s = 1;
void P(int i) {
    while (true) {
        semWait(s);
        /* critical section */;
        semSignal(s);
        /* remainder */;
    }
}

void main() {
    parbegin (P(1), P(2), . . . , P(n));
}
```
LSN 8 – Using Semaphores

Note that normal execution can proceed in parallel but that critical regions are serialized.
LSN 8 – Producer – Consumer Problem

• One or more producers are generating data and placing it in a buffer
• A single consumer is taking items out of the buffer one at time until buffer is empty
• Only one producer or consumer may access the buffer at any one time
LSN 8 – Producer – Consumer Problem

- Assume infinite buffer

Note: shaded area indicates portion of buffer that is occupied
LSN 8 – Homework

• Reading
  – Chapter 5.1 – 5.2

• Assignment:
  – Work producer/consumer problem assuming a finite length buffer