CS 3305

Process Synchronization - III

Lecture 14
Process Synchronization

- Race Condition
- Critical Section
- Mutual Exclusion
- Peterson’s Solution
- Disabling Interrupts
- Test and Lock Instruction (TSL)
- Semaphores
- Deadlock
Mutual Exclusion in Critical Sections

A enters critical region

A leaves critical region

B attempts to enter critical region

B blocked

B enters critical region

B leaves critical region

Process A

Process B

T1

T2

T3

T4

Time
Semaphores

- Semaphore is a process synchronization technique supported by the OS

- Semaphores are used by programmers to
  - ensure mutual exclusion
  - send signals from one process to another

- Today there are libraries that provide application programmers with semaphores
What is a semaphore?

- A semaphore allows multiple processes to cooperate by using signal. Semaphore is an integer variable with the following three operations:
  - **Initialize**: Semaphore (S) is initialized to a positive value
  - **Decrement**: (down operation) decrements the semaphore
  - **Increment**: (up operation) increments the semaphore value

- If S is positive then only a process enters into critical section
- Two types of semaphore: Binary and Counting
What is a Semaphore?

- Use **down** before entering a critical section
- Use **up** after finishing with a critical section
- Example: Assume $S$ is initialized to 1.

```plaintext
S = 1;
while (true)
{
    down (S);
    critical section
    up(S);
    up(S);
    remainder section;
}
```
Semaphores Example

Process $P_0$
$S = 1$
down(S);
   critical section
up(S);

Process $P_1$
$S = 1$
down(S);
   critical section
up(S);

- Initialize the semaphore variable, $S$, to 1
- Now what would happen if $P_0$ executes the down operation?
  - The semaphore $S$ is currently 1.
  - $S$ becomes 0 and $P_0$ enters the critical section
Semaphores Example

Process $P_0$

- $S = 0$
- `down(S);`
- `critical section`
- `up(S);`

Process $P_1$

- $S = 0$
- `down(S);`
- `critical section`
- `up(S);`

Now what would happen if $P_1$ executes the `down` operation?
- The semaphore $S$ is currently 0, $P_1$ is blocked
Semaphores Example

\[ S = 0 \rightarrow 1 \]

Process \( P_0 \)

\[ \text{down}(S); \]
\[ \text{critical section} \]
\[ \text{up}(S); \]

Process \( P_1 \)

\[ \text{down}(S); \]
\[ \text{critical section} \]
\[ \text{up}(S); \]

❑ Now what would happen if \( P_0 \) is done with critical section?

❑ \( P_0 \) calls the \textit{up} function
  ❑ \( S \) becomes 1
  ❑ \( P_1 \) is unblocked and \( P_1 \) enters into Critical Section
Semaphore Types

- **Binary Semaphore:**
  - Allows only ONE process to be in critical section at a time
  - Initialized to 1
  - Often referred to as a mutex

- **Counting Semaphore:**
  - Allows multiple processes to be in critical section at a time
  - Initialized to N where N is the max processes that can be in critical section simultaneously
Deadlock

- **Deadlock** - Two or more processes are waiting indefinitely for an event that can only be caused by one of the waiting processes.

- **Deadlock conditions:**
  - Resource allocation cycle (circular wait)
  - Mutual exclusion
  - Hold and Wait
  - Non-preemptive resource
Deadlock

- Avoidance Approaches
  - Avoid cycle in the resource allocation graph

- Avoid Mutual Exclusion

- Avoid Hold & Wait
  - Release resources that may not be needed immediately
  - Do not request resource ahead of time

- Block a process that requesting large number of resources
Deadlock

- Deadlock Recovery
  - Abort all deadlock processes
  - Back up all deadlock processes to the previous safe state and then restart
  - Selectively abort processes until deadlock broken