RAMA UNIVERSITY

www.ramauniversity.ac.in

FACULTY OF ENGINEERING & TECHNOLOGY

BCA-307 Operating System

Lecturer-15

Manisha Verma
Assistant Professor
Computer Science & Engineering

Deadlocks

- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance

Methods for Handling Deadlocks

•Ensure that the system will *never* enter a deadlock state:

• Deadlock prevention
•Deadlock avoidence
•Deadlock detection and recovery

•Allow the system to enter a deadlock state and then recover it means any state to enter in state then resources will be sharable mode ,so that condition will handle dealock.

•Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

Deadlock Prevention

- Mutual Exclusion – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it.
  - Low resource utilization; starvation possible

No Preemption –

If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released

Preempted resources are added to the list of resources for which the process is waiting

Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

•Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.

•The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

•Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

•When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

•System is in safe state if there exists a safe sequence of all processes.

•Sequence <P1, P2, …, Pn> is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j < i.

- •If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.
- •When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.
- •When Pi terminates, Pi+1 can obtain its needed resources, and so on.

Facts

- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

Deadlock Example

```
•/* thread one runs in this function */
•void *do_work_one(void *param)
{
•   pthread_mutex_lock(&first_mutex);
•   pthread_mutex_lock(&second_mutex);
•   /** * Do some work */
   pthread_mutex_unlock(&second_mutex);
•   pthread_mutex_unlock(&first_mutex);
•   pthread_exit(0);
•}
•/* thread two runs in this function */
•void *do_work_two(void *param)
{
•   pthread_mutex_lock(&second_mutex);
•   pthread_mutex_lock(&first_mutex);
•   /** * Do some work */
   pthread_mutex_unlock(&first_mutex);
•   pthread_mutex_unlock(&second_mutex);
•   pthread_exit(0);
•}
```

Deadlock Example with Lock Ordering

```
•void transaction(Account from, Account to, double amount)
•{
•   mutex lock1, lock2;
•   lock1 = get_lock(from);
•   lock2 = get_lock(to);
•   acquire(lock1);
•      acquire(lock2);
•         withdraw(from, amount);
•         deposit(to, amount);
•      release(lock2);
•   release(lock1);
•}
```

Banker's Algorithm

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

Data Structures for the Banker's Algorithm

- Available: Vector of length m. If available [j] = k, there are k instances of resource type Rj available
- Max: n x m matrix. If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj
- Allocation: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj
- Need: n x m matrix. If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task
  - Need [i,j] = Max[i,j] – Allocation [i,j]

- The content of the matrix Need is defined to be Max – Allocation

	Need		
	A	B	C
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

•The system is in a safe state since the sequence < P1, P3, P4, P2, P0> satisfies safety criteria

MCQ

The request and release of resources are _____.
A. command line statements
B. interrupts
C. system calls
D. special programs

Multithreaded programs are :
A. lesser prone to deadlocks
B. more prone to deadlocks
C. not at all prone to deadlocks
D. None of these

For Mutual exclusion to prevail in the system :

A. at least one resource must be held in a non sharable mode
B. the processor must be a uniprocessor rather than a multiprocessor
C. there must be at least one resource in a sharable mode
D. All of these

For a Hold and wait condition to prevail :

A. A process must be not be holding a resource, but waiting for one to be freed, and then request to acquire it
B. A process must be holding at least one resource and waiting to acquire additional resources that are being held by other processes
C. A process must hold at least one resource and not be waiting to acquire additional resources
D. None of these

For effective operating system, when to check for deadlock?

A. every time a resource request is made
B. at fixed time intervals
C. both (a) and (b)
D. none of the mentioned