OPERATING SYSTEMS

Deadlocks

Chandravva Hebbi
Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Deadlock Prevention, Deadlock Example

Chandravva Hebbi
Department of Computer Science

Methods for Handling Deadlocks

- Generally speaking there are three ways of handling deadlocks:
  - Deadlock prevention or avoidance - Do not allow the system to get into a deadlocked state.
  - Allow the system to enter into deadlocked state, detect it, and recover.
  - Ignore the problem all together and pretend that deadlocks never occur in the system.

Deadlock Prevention

4 Necessary conditions for deadlock to occur

- **Mutual Exclusion**
  - At least one resource must be non sharable
    - Ex. printers and tape drives, mutex locks
  - Sharable resources do not require mutual exclusion
    - Ex . Read-only files
- **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

Deadlock Prevention (Cont.)

- **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** –
  - Each resource will be assigned with a numerical number.
  - A process can request the resources increasing/decreasing. order of numbering.

Circular Wait

- Ex., if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.
- Resources={R1,R2,…..Rm}
- Resources are assigned unique number
- Each process request resource in increasing order enumeration
- we define a one-to-one function F: R→N, where N is the set of natural numbers
- Protocol 1: Process makes request for Ri and then for Rj. Resources Rj request is allowed if and only if F(Rj)>F(Ri).
- Protocol 2: Process requesting an instance of resource type Rj, must have released any resource Ri , such that F(Ri)>=F(Rj).
- If these protocols are used then circular wait will not exist.

Circular Wait

If these two protocols are used, then the circular-wait condition cannot hold.

Pro0f by contradiction:

- We can demonstrate this fact that by assuming that circular wait condition cannot hold
- Consider set of processes P={P0,P1,……Pn}
- Let us consider process P0 is waiting for resource held by P1, P1 waiting for P2,……..Pn-1 is waiting resource held by Pn, Pn is waiting for resources held by P0.
- Generalizing this, Process Pi is waiting for resources Ri, Ri is held by Pi+1 and it is making request for Ri+1
- We must have F(Ri)<F(Ri+1)
- But this condition means that $F(R0) < F(R1) < \ldots < F(Rn) < F(R0)$.
- By transitivity, $F(R0) < F(R0)$, which is impossible
- Therefore no circular wait.

Circular Wait

Example

- F(tape drive)=1
- F(disk drive)=5
- F(printer)=12
- F(tape drive)<F(printer)

Circular Wait

- Ex., if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.
- Invalidating the circular wait condition is most common.
- Assign each resource (i.e., mutex locks) a unique number.
- Resources must be acquired in order.
- If:
  first_mutex = 1
  second_mutex = 5

  code for **thread_two** could not be written as follows:

OPERATING SYSTEMS

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);
}
```

Transactions 1 and 2 execute concurrently. Transaction 1 transfers $25 from account A to account B, and Transaction 2 transfers $50 from account B to account A

THANK YOU

Chandravva Hebbi
Department of Computer Science Engineering

chandravvahebbi@pes.edu