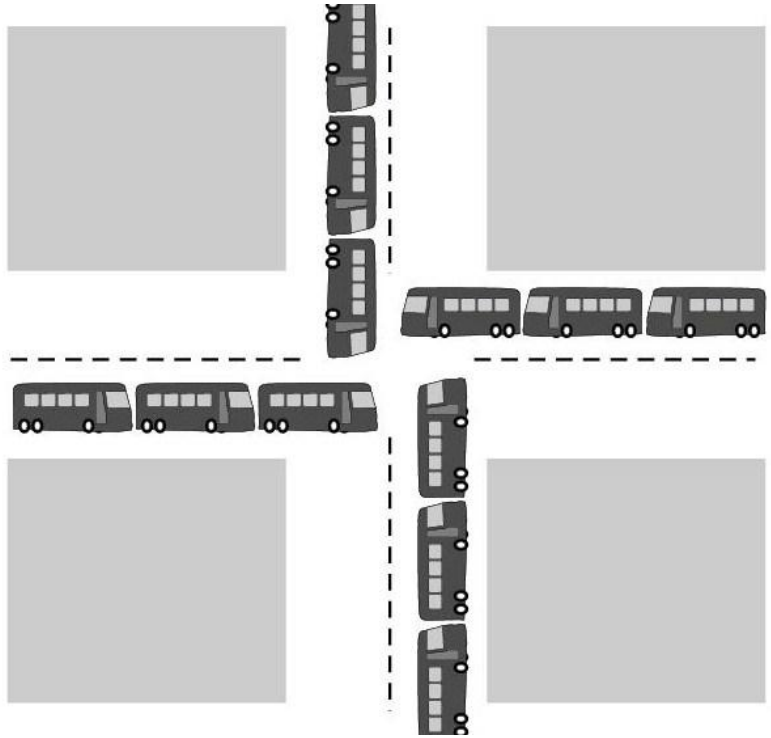
Deadlock

A process in a multiprogramming system is said to be in dead lock if it is waiting for a particular event that will never occur.

Deadlock Example

- All automobiles trying to cross.

- Traffic Completely stopped.
- Not possible without backing some.



A Traffic Deadlock

Resource Deadlock

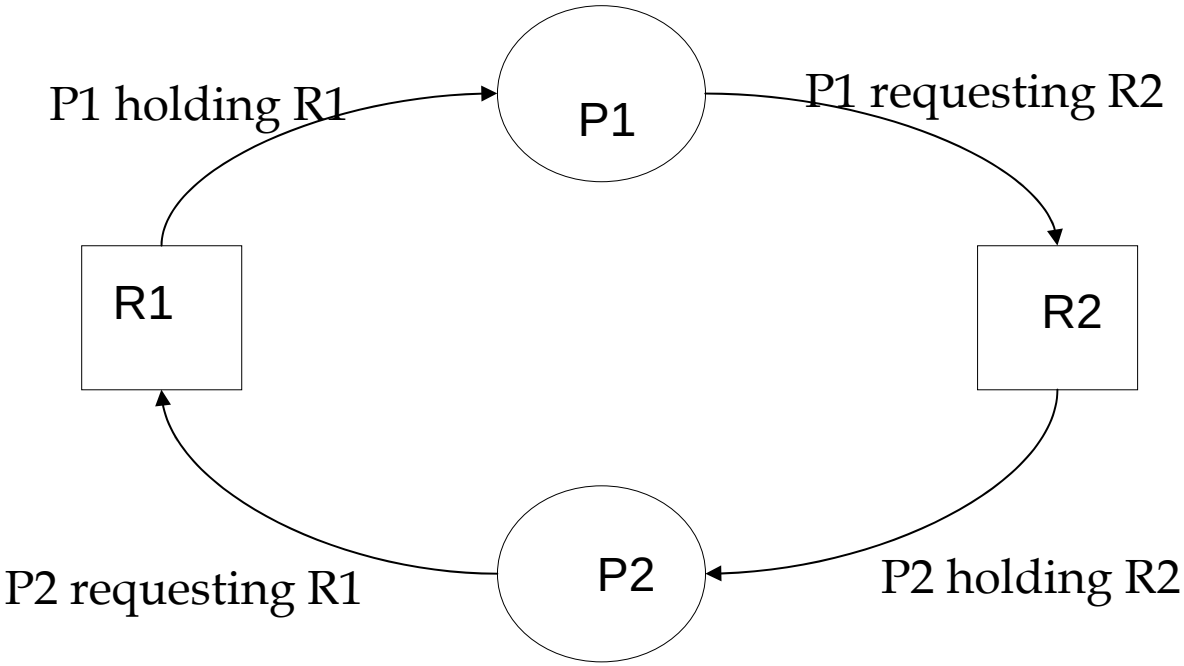
A process request a resource before using it, and release after using it.

1. Request the resource.
2. Use the resource.
3. Release the resource.

If the resource is not available when it is requested, the requesting process is force to wait.

Most deadlocks in OS developed because of the normal contention for dedicated resources.

Resource Deadlock



- Process P1 holds resource R1 and needs resource R2 to continue; Process P2 holds resource R2 and needs resource R1 to continue – deadlock.

Key: Circular wait - deadlock

Conditions for Deadlock

1. *Mutual Exclusion:* Process claims exclusive control of resources they require.
2. *Hold and Wait:* Processes hold resources already allocated to them while waiting for additional resources.
3. *No preemption:* Resources previously granted can not be forcibly taken away from the process.
4. *Circular wait:* Each process holds one or more resources that are requested by next process in the chain.

A deadlock situation can arise if all four conditions hold simultaneously in the system.

Handling Deadlocks

- Deadlock handling strategies:
- We can use a protocol to *prevent* or *avoid* deadlocks, ensuring that the system never enter a deadlock state.
- We can allow the system to enter a deadlock state, *detect* it, and *recover*.

- We can ignore the problem all to gather, and pretended that deadlock never occur in the system.

Deadlock prevention

Fact: *If any one of the four necessary conditions is denied, a deadlock can not occur.*

Denying Mutual Exclusion:

- Sharable resources do not require mutually exclusive access such as read only shared file.

- Problem: Some resources are strictly nonsharable, mutually exclusive control required.

We can not prevent deadlock by denying the mutual exclusion

Deadlock prevention

Denying Hold and Wait:

Resources grant on *all or none* basis;

If all resources needed for processing are available then granted and allowed to process.

If complete set of resources is not available, the process must wait set available.

While waiting, the process should not hold any resources.

Problem:

Low resource utilization.

Starvation is possible.

Deadlock prevention

Denying No-preemption:

When a process holding resources is denied a request for additional resources, that process must release its held resources and if necessary, request them again together with additional resources.

Problem:

When process releases resources the process may loose all its works to that point .

Indefinite postponement or starvation.

Deadlock prevention

Denying Circular Wait:

All resources are uniquely numbered, and processes must request resources in linear ascending order.

The only ascending order prevents the circular.

Problem:

Difficult to maintain the resource order; dynamic update in addition of new resources. Indefinite postponement or starvation.

Deadlock Avoidance

Avoiding deadlock by careful resource allocation.

Decide whether granting a resource is safe or not, and only make the allocation when it is safe.

Need extra information in advanced, maximum number of resources of each type that a process may need.

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

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

Safe and Unsafe States

A state is said to be safe if it is not deadlocked and there is a some scheduling order in which every process can run to completion.

(a) (b) (c) (d) (e)

State in (a) is safe

Deadlock Avoidance

Deadlock Avoidance

Safe and Unsafe States

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

State in (b) is not safe.

In a safe state, system can guarantee that all processes will finish – no deadlock occur; from an unsafe state, no such guarantee can be given – deadlock may occur.

Banker's Algorithm

Models on the way of banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.

When a process request the set of resources, the system determine whether the allocation of these resources will left the system in safe state, if it will the resources are allocated, otherwise process must wait until some other process release enough resources.

Data structures: Available, Max, Allocation & Need. need = max – allocation.

Deadlock Avoidance

Banker's Algorithm

$finish_i$ = False;

for each process if ($need_i$ <= available &&

$finish_i$ = false)

continue$_i$;

available = available + allocation;

$finish_i$ = True;

for each process if ($finish_i$ = True) system is in safe state.

if ($request_i$ <= $need_i$)

if ($request_i$ <= available) { available = available- $request_i$; $allocation_i$ = $allocation_i$ + $request_i$; $need_i$ = $need_i$–$request_i$;

}

else wait;

Deadlock Avoidance

else | error;

Banker's Algorithm

allo. max allo. max allo. max

Source: www.csitnepal.com

Deadlock Avoidance

A	0	6
B	0	5
C	0	4
D	0	7

Available = 10

A	1	6
B	1	5
C	2	4
D	4	7

Available = 2

A	1	6
B	2	5
C	2	4
D	4	7

Available = 1

Which one is unsafe?

Deadlock Avoidance
Problem with Banker's Algorithm

Algorithms requires fixed number of resources, some processes dynamically changes the number of resources.

Algorithms requires the number of resources in advanced, it is very difficult to predict the resources in advanced.

Deadlock Avoidance

Algorithms predict all process returns within finite time, but the system does not guarantee it.

Deadlock Detection and Recovery

Instead of trying to prevent or avoid deadlock, system allow to deadlock to happen, and recover from deadlock when it does occur.

Hence, the mechanism for deadlock detection and recovery from deadlock required.

Deadlock Detection

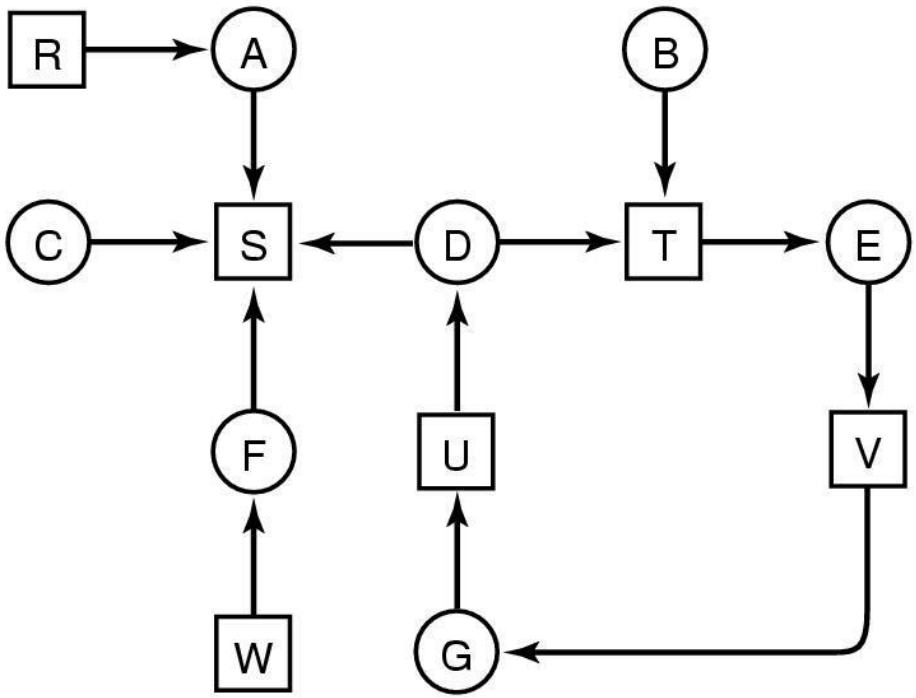
Consider the following scenario:
a system with 7 processes (A – G) , and 6 resources (R – W) are in following state.

1. Process A holds R and wants S.
2. Process B holds nothing but wants T.
3. Process C holds nothing but wants S.
4. Process D holds U and wants S and T.
5. Process E holds T and wants V.

6. Process F holds W and wants S. 7 . Process G holds V and wants U.

Is there any deadlock situation?

Deadlock Detection



How to detect the cycle in directed graph?

Deadlock Detection Algorithm

```
List L;
Boolean cycle = False;
for each node
    L = Φ   /* initially empty list*/ add node to L; for each
    node in L for each arc if (arc[i] = true)
                add corresponding node to L;
                if (it has already in list)
                        cycle = true;
                print all nodes between these nodes;
                exit;
```

```
else if (no such arc) remove
        node from L;
        backtrack;
```

Recovery from Deadlock

What do next if the deadlock detection algorithm succeed? – recover from deadlock.

By Resource Preemption

Preempt some resources temporarily from a processes and give these resources to other processes until the deadlock cycle is broken.

Problem:

Depends on the resources.

Need extra manipulation to suspend the process.

Difficult and sometime impossible.

Recovery from Deadlock

By Process Termination

Eliminating deadlock by killing one or more process in cycle or process not in cycle.

Problem:

If the process was in the midst of updating file, terminating it will leave file in incorrect state.

Key idea: we should terminate those processes the termination of which incur the minimum cost.

Ostrich Algorithm

Fact: there is no good way of dealing with deadlock.

Ignore the problem altogether.

For most operating systems, deadlock is a rare occurrence;

So the problem of deadlock is ignored, like an ostrich sticking its head in the sand and hoping the problem will go away.

Home Works

HW #6

1. Q. 1, 2, 3, 4, 6, 12, 14, 15, 20 & 22 from Textbook (Tanenbaum)
2. Distinguish indefinite postponement and Deadlock.
2. State the conditions necessary for deadlock to exist. Give reason, why all conditions are necessary.
3. A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain your answer.

4. Show that four necessary conditions hold in traffic deadlock example. State a simple rule that will avoid deadlock in traffic system.