*25-September-2017*

**DEADLOCK** BAB 6

* **Resource** is anything that must be acquired, used, and released over the course of time.
* **Resource Acquisition**
  + For some kinds of resources, such as records in a database system, it is up to the user processes rather than the system to manage resource usage themselves. One way of allowing this is to associate a semaphore with each resource. These semaphores are all initialized to 1. Mutexes can be used equally well.
* **Deadlock**

can be defined formally as follows:

“ *A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.”*

=========================**deadlock** is called a **resource deadlock**==========================

* For many applications, a process needs exclusive access to not one resource, but several. Suppose, for example, two processes each want to record a scanned document on a Blu-ray disc. Process A requests permission to use the scanner and is granted it. Process B is programmed differently and requests the Blu-ray recorder first and is also granted it. Now A asks for the Blu-ray recorder, but the request is suspended until B releases it. Unfortunately, instead of releasing the Bluray recorder, B asks for the scanner. At this point both processes are blocked and will remain so forever. This situation is called a **DEADLOCK.**
  + **Preemptable and Nonpreemptable Resources**
    - **Preemptable resource** is one that can be taken away from the process owning it with no ill effects. Memory is an example of a preemptable resource. Consider, for example, a system with 1 GB of user memory, one printer, and two 1-GB processes that each want to print something.
    - **Nonpreemptable resource** is one that cannot be taken away from its current owner without potentially causing failure. If a process has begun to burn a Blu-ray, suddenly taking the Blu-ray recorder away from it and giving it to another process will result in a garbled Blu-ray
* **Conditions for Resource Deadlocks**
  + Coffman et al. (1971) showed that four conditions must hold for there to be a (resource) deadlock:
    - 1. Mutual exclusion condition. Each resource is either currently assigned to exactly one process or is available.
    - 2. Hold-and-wait condition. Processes currently holding resources that were granted earlier can request new resources.
    - 3. No-preemption condition. Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.
    - 4. Circular wait condition. There must be a circular list of two or more processes, each of which is waiting for a resource held by the next member of the chain
* **Strategies Are Used For Dealing With Deadlocks**

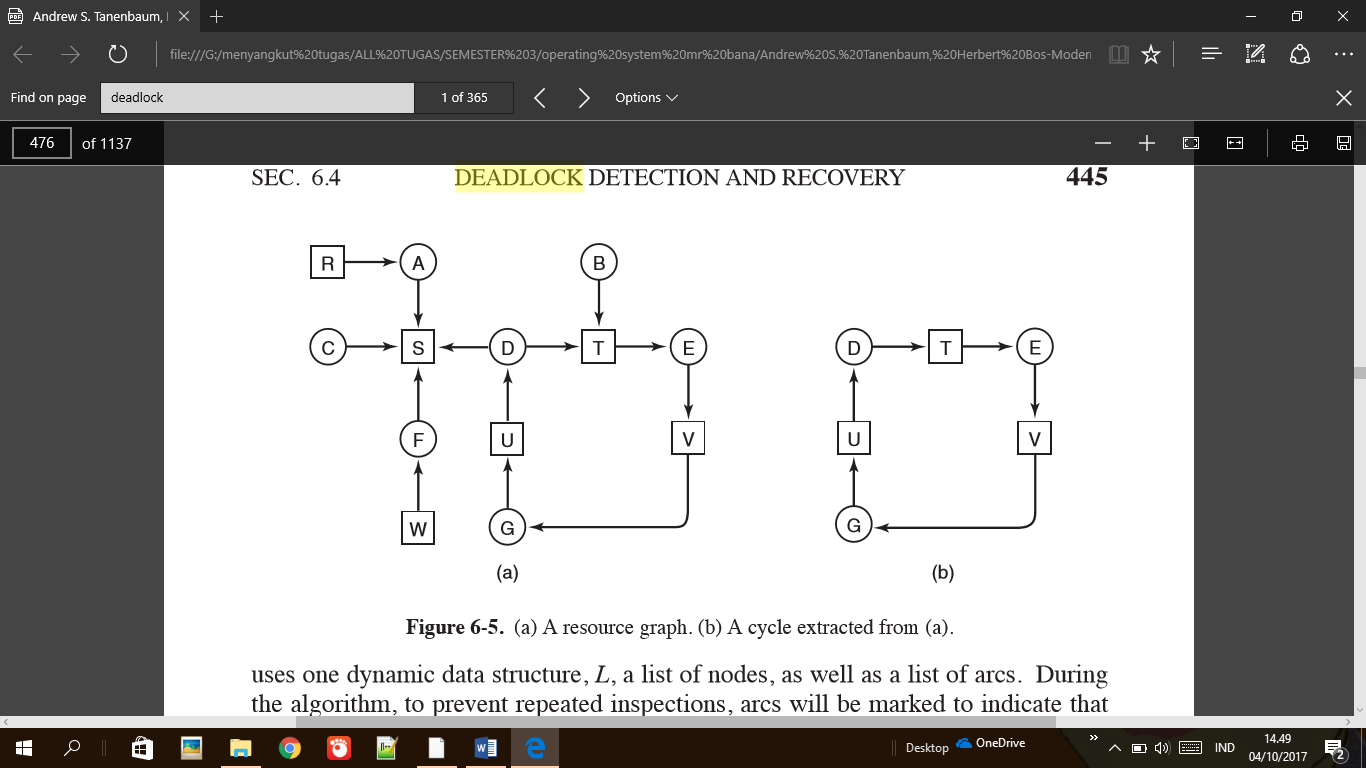
1. Just ignore the problem. Maybe if you ignore it, it will ignore you.

2. Detection and recovery. Let them occur, detect them, and take action.

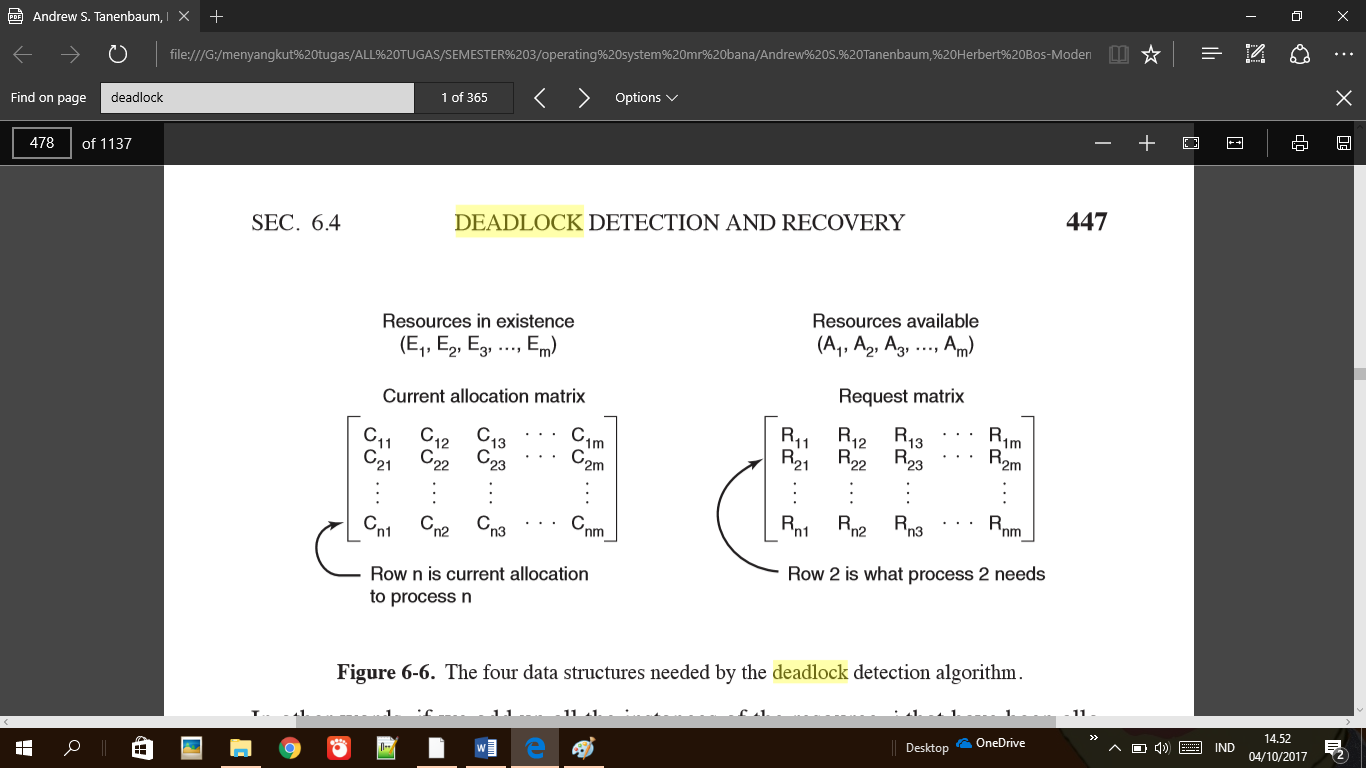
3. Dynamic avoidance by careful resource allocation.

4. Prevention, by structurally negating one of the four conditions

* **Deadlock Detection and Recovery** 
  + When deadlock occured, we can’t solve it. Instead it happen we can try to detect what is the truly happen, in here we just can see the fact and then the recover the effect of deadlock that had happened.
* **According Each Type :**
  + Deadlock detection with one resource



* + Deadlock detection with multiple resource



* **Deadlock Avoidance**
  + Deadlock avoidance mean an algorithm for avoid the happening of deadlock. So let pick one example : process A request device A and granted, process B request device B when it finish the process within device A. Unfortunately process A also request device B and granted too, then process B can’t continue it process because of process A. After we know the process of that deadlock, we can create algorithm for avoid that deadlock happen again by suspend the process B first and start it after process A finish it job.