12 Lecture: MINIX IO Architecture

Outline:

12.1 Announcements

- Coming attractions:

Event	Subject	Due Date			Notes
asgn3	dine	Wed	Feb 4	23:59	
lab03	problem set	Mon	Feb 9	23:59	
midterm	stuff	Wed	Feb 11		
lab04	scavenger hunt II	Wed	Feb 18	23:59	
asgn4	/dev/secret	Wed	Feb 25	23:59	
lab05	problem set	Mon	Mar 9	23:59	
asgn5	minget and minls	Wed	Mar 11	23:59	
asgn6	Yes, really	Fri	Mar 13	23:59	
final (sec01)		Fri	Mar 20	10:10	
final (sec03)		Fri	Mar 20	13:10	

  Use your own discretion with respect to timing/due dates.

- Old exams on the web site (warning. . .)

- Just a reminder that you should be reading T&W. Remember that chapter 2 reading is slow going.

12.2 Where do we go from here?

Three major operational areas for any operating system: (specializations of magic?)

IO How does information get in and out of the system and get where it's going.

Memory Management Making sure that things are where programs expect them to be.

Filesystem Making sure things stay where put and that everything operates efficiently.

12.3 Input/Output

Without IO, there's no real point in doing the computation. It's also complicated.

As always, it's all about abstraction: keep the dirty machine details hidden.

12.4 Devices

The Unix/Minix modes, devices are classified as:

- block devices (e.g. disks)
- character devices (e.g. ttys)

Not always clear: networks? printers? tapes? clocks? To model deadlocks, we need to consder abstract **resources** of two types:

preemptable resource one that can be taken away from the process owning it with no ill effects. (e.g. the CPU, memory)

nonpreemptable resoure one that can not be taken away from the process holding it without causing an error. (e.g. a printer once it's started to print.)

To use a non-preemptable resource:

1. request the resource
2. use the resource
3. release the resource.

12.5 Deadlock and its avoidance

Since we're talking about sharing, we have to worry about deadlock.

Occasionally a process must be given exclusive access to some resource, e.g.:

- a scanner,
- cdrom,
- tape drive,
- mouse,
- printer(lpd is a user process),
- terminal(usually unnoticed), etc.

Imagine two processes:

- copy tape to cdrom
- copy cdrom to tape

oops. (see Fig. 19 for a resource graph description)

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.

Conditions required for deadlock: (Coffman, *et al.*, 1971)

1. **Mutual exclusion** processes don't share
2. **Hold and wait** processes can hold a resource and make other requests
3. **No pre-emption** processes can't be forced to surrender a resource
4. **Circular wait** the circle must exist.

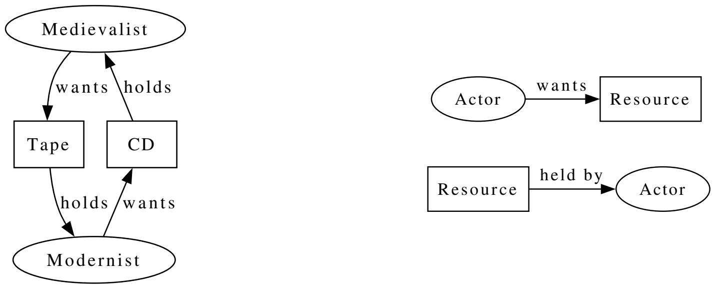

Figure 19: How a deadlock forms

12.5.1 Deadlock Avoidance Methods

- **Don't** the ostrich approach
- **Detection and recovery** discover deadlocks and kill jobs until it's gone. (carefully, or heuristically)
- **Prevention** place restrictions on accesses so that deadlock can never occur. (e.g. order resources)
- **Avoidance** Be very careful in resource allocation so it doesn't happen.

12.6 Banker's Algorithm (Dijkstra, 1965)

Keep track of the current state of resource allocation and only approve resources that could not make any one customer's requests unsatisfiable. (approve if there is a path that gets all customers to their credit limits.)

Each process must publish its maximum requirements up front.

12.6.1 Single Resource

Analogous to a line of credit: Bankers keep on hand much less than the sum of the maxima. The OS can, too.

A state is considered **safe** if there exists a sequence of states that leads to all customers getting their maximum loans. (resources)

The process:

1. Choose a process whose resource requirements can be satisfied. (It doesn't matter which, because it always increases the resource pool.)
2. Assume its resources are released (because it's finished)
3. Repeat until all processes terminate or there are no more satisfiable processes.

If all processes can terminate, the state **safe**. If not, it is **unsafe** and the resource request must be delayed.

Consider the following example in Figure 20 (from Tanenbaum).

Initial condition:

Name	**Used**	**Max.**	**Needed**
A	0	6	6
B	0	5	5
C	0	4	4
D	0	7	7

Available: 10

Safe: any order.

In progress:

Name	**Used**	**Max.**	**Needed**
A	1	6	5
B	1	5	4
C	2	4	2
D	4	7	3

Available: 2

Safe: $C \to D \to B \to A$

After B requests one more unit:

Name	**Used**	**Max.**	**Needed**
A	1	6	5
B	2	5	3
C	2	4	2
D	4	7	3

Available: 1

Unsafe: Nobody

Figure 20: One dimensional example of Dijkstra's Bankers Algorithm

Initial condition:

Name	Used	Max.	Needed
A	0	10	10
B	0	2	2
C	0	4	4
D	0	9	9
E	0	3	3

Available: 11

Safe: any order.

In progress:

Name	Used	Max.	Needed
A	2	10	8
B	1	2	1
C	2	4	2
D	2	9	7
E	2	3	1

Available: 2

Safe: $C(4) \rightarrow B(5) \rightarrow E(7) \rightarrow D(9) \rightarrow A(11)$

After A requests one more unit:

Name	Used	Max.	Needed
A	3	10	7
B	1	2	1
C	2	4	2
D	2	9	7
E	2	3	1

Available: 1

Unsafe: $B(2) \rightarrow C(4) \rightarrow E(6) \rightarrow$ Nobody

Figure 21: Another example, but one that doesn't wedge immediately.