13 Lecture: Minix IO, cont.

Outline:

- Announcements
- Managing Multiple Resources
  - Resource Trajectories
- Managing Multiple Resources, cont.
  - Multi-way bankers'
- Wrapping up deadlock avoidance
- Back to I/O
- Devices
  - Low level considerations: timing, interleaving, etc.
  - Accessing a device: Device Controllers
- Reading from a device: DMA vs. Programmed IO

13.1 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. . .)

- Cleaning up after yourself.
  - `ps -u` *username*
  - `killall -u` *username* `-r pn-cs453/lib`

- Style guide:
  - Clean build
  - Magic numbers
  - Long lines (`~pnico/bin/longlines.pl`)
  - **not checking error returns**
  - A brace is the last thing on a line...

  These will *always* cost points unnecessarily.

13.2 Managing Multiple Resources

But multiple types of resources are a problem...

13.2.1 Resource Trajectories

The example from Tanenbaum, p.249 is shown in Figure 22.

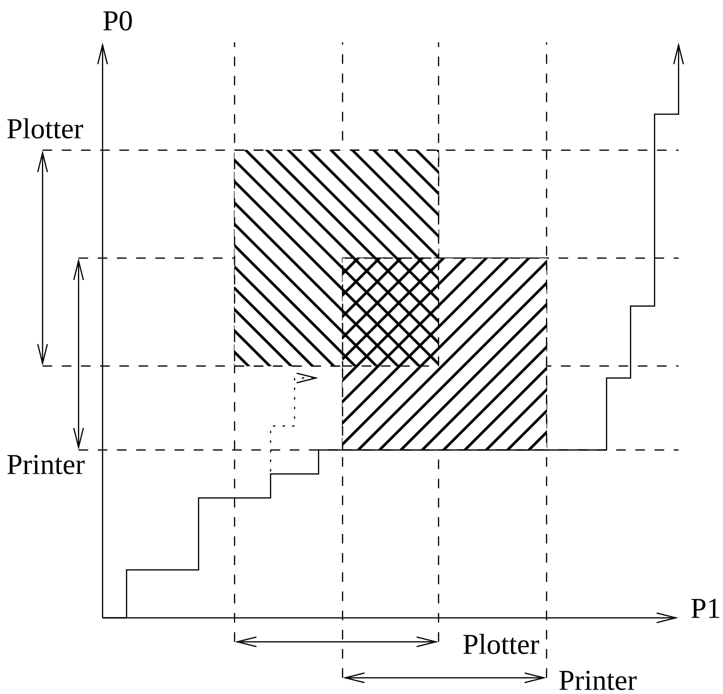

Figure 22: Resource trajectory example from Tanenbaum

Yuck.

13.3 Managing Multiple Resources, cont.

13.3.1 Multi-way bankers'

Here we simply do a vectorized version of the Banker's Algorithm:

- Maintain a table of held resources
- Maintain a table of maximum requests
- Maintain a table of remaining requests
- Maintain a vectors of allocated, free, total resources

If there does not exist a row less than the available vector, the system will deadlock, else:

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.

See the example in Figure 23. Manipulations:

1. Initial situation

2. B requests an instance of R_2 (printer?): (safe: D, then A or E, then...)

3. E requests the last R_2 printer: **unsafe**

13.4 Wrapping up deadlock avoidance

Are any of these any good?

- Processes rarely know their resource requirements in advance.

- Processes come and go.

This is *hard*.

If you have a good idea, you can be famous like Dijkstra. :)

What does Unix do?

Not a thing. The Unix way (as with many other operating systems) is to hope it doesn't happen, and if it does, it expects some higher being (super user) to fix it. Think *Deus extra machina.*

13.5 Back to I/O

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.

13.6 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?

13.6.1 Low level considerations: timing, interleaving, etc.

Isn't it nice the controller can take care of it?

(under each level is another nice level of abstraction)

	R_0	R_1	R_2	R_3
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Assigned Resources

	R_0	R_1	R_2	R_3
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources Still Needed

Exists:	(6	3	4	2)
Alloc.:	(5	3	2	2)
Free:	(1	0	2	0)

(a) Initial Configuration

	R_0	R_1	R_2	R_3
A	3	0	1	1
B	0	1	*1*	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Assigned Resources

	R_0	R_1	R_2	R_3
A	1	1	0	0
B	0	1	*0*	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources Still Needed

Exists:	(6	3	4	2)
Alloc.:	(5	3	3	2)
Free:	(1	0	1	0)

(b) After B requests an instance of R_2
(Safe: $D \to A \to E \to \dots$)

	R_0	R_1	R_2	R_3
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	*1*	0

Assigned Resources

	R_0	R_1	R_2	R_3
A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	*0*	0

Resources Still Needed

Exists:	(6	3	4	2)
Alloc.:	(5	3	4	2)
Free:	(1	0	0	0)

(b) After E requests an instance of R_2
(Unsafe: No process can ba satisfied)

Figure 23: Multi-dimensional Bankers Algorithm

13.6.2 Accessing a device: Device Controllers

Mercifully the OS talks to device controllers not the actual devices. Isn't abstraction great?

Device controllers abstract away much of the complexity. Accessed via:

Memory-Mapped IO Device control registers are mapped into memory. (creates "holes" in memory)

IO Ports Device control registers must be accessed through special instructions. (convenient, but complicates the CPU).

Either way, we set a value in some register, then the controller does the thing, then it sets a register to tell us that it's done it.

13.7 Reading from a device: DMA vs. Programmed IO

Standard ("Programmed IO") Controller interrupts, CPU copies data from controller to memory.

DMA Controller copies data to memory, then interrupts.

At least we have interrupts (Think about the world if we didn't. We'd just have to check again and again, called "polling").