11 Lecture: More Scheduling

Outline:

Announcements
    Preemptive scheduling
Minix architecture revisited
Minix 2 Scheduling
Minix 3 Scheduling
    Three-level scheduling
Scheduling Example
Minix IPC
Minix structures
Where do we go from here?

11.1 Announcements

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

11.1.1 Preemptive scheduling

Priority Sheduling Give each job (or class of jobs) a priority (rank, price, etc) and choose the most important one.

Idea: Break priorities into classes: IO Bound first, then compute-bound. Why?

Example: CTSS (Compatible Time Sharing System):

- large quantum for CPU-bound jobs:
- Processes that use the whole quantum move down.
- Processes doing IO move up to the top again.

Other possibilities for granting priority:

- Process already in memory?
- Locks held/needed

- Resource requirements
- Shortest remaining work next (past performance might be a good predictor of future results?)

Guaranteed/Lottery Give each process its due.

Real Time A whole other course

11.2 Minix architecture revisited

The structure of minix is as in Figure 18.

4	init	proc A	proc B	proc C	...		User Processes
3	File Sys.	Network	Info. Srv.	Proc. Mgr.	Reincarnation Srv.	...	Server Processes
2	Disk	Tty	Ethernet	Memory	...		I/O Tasks
1	scheduling, interrupts, communication				Clock Task	System Task	Process Management

Figure 18: Layout of the Minix system

Layer 1 Sheduling, handling traps and interrupts, facilitating message passing.

The bottom is written in assembly(must be), the rest is C.

Clock and System Tasks exist here.

Layer 2 IO Processes, one per device type. Compiled into the kernel, but have separate identities, (and, if supported, privilege levels)

Layer 3 Server processes: File System, Memory Manager, Network Server, etc. These are user-level processes that implement the system calls.

These run with higher priority than user processes and never terminate while the system is running.

Layer 4 User-level processes.

11.3 Minix 2 Scheduling

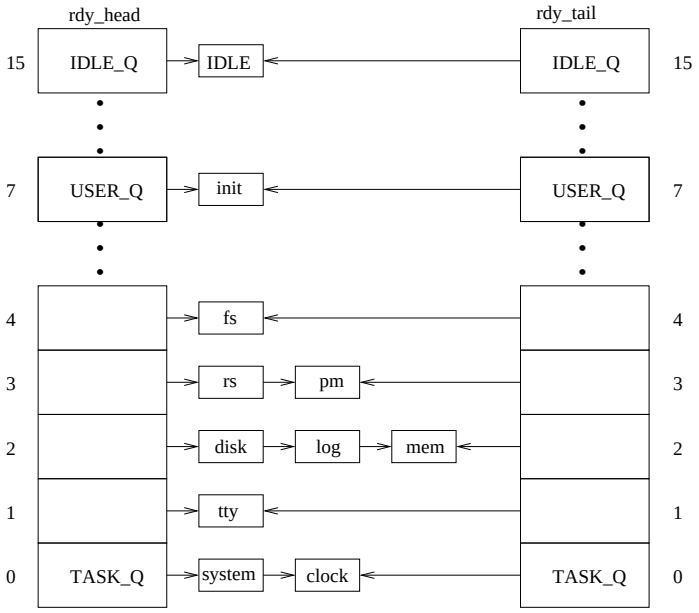
Three-level priority scheduling system:

Level 1	immediate (so no queue)
Level 2	run to completion/block
Level 3	run to completion/block
Level 4	Round-robin. 100ms quantum

Remember the idle process

11.4 Minix 3 Scheduling

Sixteen-level priority scheduling system:



- Variable quantum size set in the process table
- A process that uses its entire quantum will be given another one and sent to the end of the queue.
- Potential change of priority level for processes using complete quanta:
  - If it uses its whole quantum and is its own successor: drops a level.
  - If it was not its own predecessor, rises a level, capped in process table.
- Processes that block with time remaining in their quantum are moved to the head of the queue when they return, but only with the remaining part of their quanta.

11.4.1 Three-level scheduling

For *any* scheduling scheme, it might be worthwhile to treat processes in memory and processes on disk differently.

Processes can be scheduled at three different locations:

admission The system can decide whether or not to admit new jobs, and what to do with them.

memory Constraints are different for jobs in memory and on disk

disk (above)

11.5 Scheduling Example

Perhaps?

11.6 Minix IPC

Three primitives of Minix IPC:

Synchronous

```
send(dest, &message);
receive(source, &message);
send_rec(dest_src, &message);
```

Asynchronous

```
notify(dest);
```

Messages (other than `notify()`) communicated via **rendezvous** semantics: The sender waits until the receiver gets the message. Notify is asynchronous.

Why?

- simplifies buffer management.
- more predictable: there is no question of a program behaving differently given a different buffer size (out of its control)

  (How would one find out the buffer size?)

11.7 Minix structures

To prepare for the next part of the course, read Sections 2.5–2.6 (this time for real):

- Understand the basic structure of the OS
- Be careful and read critically: There is much to learn from observing OS code. There are also places where OS writers do things that ordinary C programmers should not. ("advanced strategy" — G. Fink)

  Remember: design considerations for an OS are different from "mortal programs". (e.g. consistency above all else `panic()`)
- Read the code; you will eventually anyhow.

11.8 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.