CS 33

Final Words
Thread Scheduling

• The OS multiplexes threads on the available processors/cores
  – share the processors equally
    » time slicing: each thread gets a fixed amount of time before it’s forced to yield the processor to another thread (if there is one)
  – some threads are more important than others
    » priorities: higher-priority threads get the processor in preference to lower-priority threads
A Scheduling Issue

• You and four friends each contribute $1000 towards a server
  – you, rightfully, feel you own 20% of it
• Your friends are into threads, you’re not
  – they run 5-threaded programs
  – you run a 1-threaded program
• The scheduler treats all threads equally
• Their programs each get 5/21 of the processor
• Your programs get 1/21 of the processor
  – (you should have paid more attention to the fractal threads lab)
Lottery Scheduling

• 25 lottery tickets are distributed equally to you and your four friends
  – you give 5 tickets to your one thread
  – they give one ticket each to their threads

• A lottery is held for every scheduling decision
  – your thread is 5 times more likely to win than the others
Metered Processors
Algorithm

- Each thread has a meter, which runs only when the thread is running on the processor.
- At every clock tick:
  - give processor to thread that’s had the least processor time as shown on its meter.
  - in case of tie, thread with lowest ID wins.
Issue

• Some threads may be more important than others
Metered Processors (RI Variation)
Details ...

- Each thread pays a bribe
  - the greater the bribe, the slower the meter runs

- to simplify bribing, you buy “tickets”
  » one ticket is required to get a fair meter
  » two tickets get a meter running at half speed
  » three tickets get a meter running at 1/3 speed
  » etc.
New Algorithm

• Each thread has a *(possibly crooked)* meter, which runs only when the thread is running on the processor

• At every clock tick
  – give processor to thread that’s had the least processor time as shown on its meter
  – in case of tie, thread with lowest ID wins
Example

- **Time (quanta)**
- **Meter value (quanta)**

The graph shows a linear relationship between time and meter value, where the meter value increases with each time quanta.
Distributed File Systems
Failures in a Local File System

On-Disk File System

Cache

Open-File State

Server

Client

Client

Client

Client
Distributed Failure
Distribution of Components

NFS client

NFS server

local file system
NFS in Action

```c
char buffer[100];
int fd = open("/home/twd/dir/fileX", O_RDWR);
read(fd, buffer, 100);
...
lseek(fd, 0, SEEK_SET);
write(fd, buffer, 100);
```
Open-File Data Structures (Client)

- File descriptor
- User address space
- File descriptor table
- Kernel address space
- ref count
- access mode
- file location
- file handle + comm. handle

refers to file on server
However ...

```c
int fd = creat("/home/twd/dir/tempfile", 0600);
char buf[1024];
unlink("/home/twd/dir/tempfile");
...
write(fd, buf, 1024);
...
lseek(fd, 0, SEEK_SET);
read(fd, buf, 1024);
close(fd);
```
Locks on Files

• Your point of view
  – you take an exclusive lock on a file
  – you expect to have exclusive access to the file until you unlock it

• Server’s point of view
  – client A locks file
  – client A appears to have crashed
  – client B wants to lock file
  – server lets client B have the lock
How Does Server Know Client Crashed?

• **Timeout**
  – after period of unresponsiveness, server assumes client must have crashed

• **Reboot**
  – client crashes
  – reboots and notifies server that it is rebooting (and thus must have crashed)

• **Neither is perfect**
  – reboot used in CS department NFS
  – timeout used in many modern systems
You’ll Soon Finish CS 33 ...

- You might
  - celebrate
  - take another systems course
    - 32
    - 138
    - 166
    - 167

- become a 33 TA
Systems Courses Next Semester

• **CS 32 (Intro to Software Engineering)**
  – you’ve mastered low-level systems programming
  – now do things at a higher level
  – learn software-engineering techniques using Java, XML, etc.

• **CS 138 (Distributed Systems)**
  – you now know how things work on one computer
  – what if you’ve got lots of computers?
  – some may have crashed, others may have been taken over by your worst (and smartest) enemy

• **CS 166 (Computer Systems Security)**
  – liked buffer?
  – you’ll really like 166

• **CS 167/169 (Operating Systems)**
  – still mystified about what the OS does?
  – write your own!
The End

Well, not quite …
Database is due on 12/14.
The TAs and I will hold hours all next week.

Happy coding and happy holidays!