19 April

Three Classes to Go!
Questions?
VM and Making Programs Go
Operating System

The OS is JUST A PROGRAM

but it runs in SUPERVISOR state
access to PHYSICAL addresses
access to special registers (like page table register)
all IO devices, etc.

whereas ordinary programs run in USER state
only access to VIRTUAL addresses through page tables
normally no access to IO devices

Programs ask the OS for services (syscall)
give me more memory
read/write data from/to disk
put pixel on screen
give me the next character from the keyboard

The OS may choose to “map” devices such as the screen into USER space
Shell

You normally interact with a SHELL
It provides the command prompt, or GUI (graphical user interface)
It is JUST A PROGRAM
It runs in USER state just like your programs
It interprets your mouse clicks or typed commands and asks the OS to implement your requests

Suppose you “double-click” on a program icon
What happens?
Program Startup in SHELL

First the SHELL finds the file (using FILE SYSTEM in OS) indicated by the icon
It checks some permissions and such
Finally it calls the EXEC system call with the file name and possibly some arguments
Now the OS takes over
OS Exec

The OS keeps a PROCESS TABLE of all running programs
  disk location of executable
  memory location of page tables
  priority
  current status (running, waiting ready, waiting on an event, etc.)
  PID (process ID) a number assigned to the process

A PROCESS is an independent program running in its own
memory space

The OS allocates a new entry in the PROCESS TABLE
And sets up the PAGE TABLE for the new process
Program Startup

Now everything is ready

- The PROCESS TABLE entry has been set up
- The PAGE TABLE for the process has been initialized
- The TEXT SEGMENT is out on disk
- The DATA SEGMENT is in memory
- The STACK SEGMENT has been allocated 1 PAGE

The OS is ready to take the leap of faith

ONLY ONE program runs at a time

When your program is running the OS is not

To run your program and maintain control the OS must trust that is will eventually get control again
  - when the program asks for a service
  - when the program does something illegal
  - when a timer goes off
Fault in the Text

When we branch to the beginning of “main” we get a page fault
So the OS copies the first page of the TEXT of main to a free page in memory
Fault in the Text

page table

0x00000000 1 text segment
0x00001000 0 text segment
0x00002000 1 data segment
0x00003000
0x00004000
0x00005000

memory

0xfffffe000
0xffffff000 1 stack

disk

foo
swap
foo
Allocate a block of memory

Now suppose the first thing our program needs to do is get 6k of memory for an array
The program uses “new” to make an array
Down inside “new” it calls “malloc”
Down inside “malloc” it uses a system call to ask the OS for memory
The OS will have to find 2 pages to hold 6k
Allocate a block of memory

![Diagram of page table and memory allocation]

- **Page Table**
  - 0x00000000: 1 text segment
  - 0x00001000: 0 text segment
  - 0x00002000: 1 data segment
  - 0x00003000: 1 heap
  - 0x00004000: 1 heap
  - 0x00005000
  - 0xfffffe000
  - 0xffffff000: 1 stack

- **Memory Allocation**
  - Disk
    - foo
    - swap
    - foo
Fault in the other page of TEXT

Page Table

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>text segment</td>
</tr>
<tr>
<td>0x00001000</td>
<td>text segment</td>
</tr>
<tr>
<td>0x00002000</td>
<td>data segment</td>
</tr>
<tr>
<td>0x00003000</td>
<td>heap</td>
</tr>
<tr>
<td>0x00004000</td>
<td>heap</td>
</tr>
<tr>
<td>0x00005000</td>
<td></td>
</tr>
<tr>
<td>0xfffffe000</td>
<td></td>
</tr>
<tr>
<td>0xffffffff000</td>
<td>stack</td>
</tr>
</tbody>
</table>

Memory

Disk

foo

swap

foo
Grow the stack

Now our program needs more stack space
Perhaps it has to call a recursive function to traverse a complex data structure
Or perhaps the user declares an “automatic” array like double work[1000];
which needs 8000 bytes of memory
Grow the stack

memory

page table

0x00000000 1 text segment
0x00000100 1 text segment
0x00000200 1 data segment
0x00000300 1 heap
0x00000400 1 heap
0x00000500
...

0xffffd000 1
0xffffe000 1
0xffffff000 1 stack

disk

foo
swap
foo
Get partially paged out

Sometime later, some other program running on the system needs more memory
It asks the OS
The OS realizes that not enough physical memory remains available
So the OS chooses to PAGE OUT one page from our program
It would choose one that hasn’t been used for a while like possibly one of the heap segments
Partially Paged Out

page table

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>text segment</td>
</tr>
<tr>
<td>0x00000100</td>
<td>text segment</td>
</tr>
<tr>
<td>0x00000200</td>
<td>data segment</td>
</tr>
<tr>
<td>0x00000300</td>
<td>heap</td>
</tr>
<tr>
<td>0x00000400</td>
<td>heap</td>
</tr>
<tr>
<td>0x00000500</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0xfffffd00</td>
<td></td>
</tr>
<tr>
<td>0xfffffe00</td>
<td></td>
</tr>
<tr>
<td>0xffffffff</td>
<td>stack</td>
</tr>
</tbody>
</table>

memory

disk

Comp 411– Spring 2007

4/19/07

L22 – 05 16
Later we need that page
Exit

Finally our program exits
It calls the “exit” system call to notify the OS that it is done
The OS puts the memory back on the free list
Cleans up the PAGE TABLE and PROCESS TABLE
And goes on about its business...

What does the OS do when no programs are ready?