Operating System I

Processes

“Program in execution”
Modern computers allow several at once
“Pseudoparallelism”

Processes

“Program in execution”
• “More” than a program: ls, tcsh
• “Less” than a program: gcc blah.c
  (cpp, cc1, cc2, ln ...)
• “A sequential stream of execution in its own address space”

Process States

Consider:
cat /etc/passwd | grep claypool

Unix Process Creation

System call: fork()
– Creates (nearly) identical copy of process
– Return value different for child/parent
System call: exec()
– Over-write with new process memory
(Hey, you, show demos!)

Design Technique: State Machines

Process states
Move from state to state based on events
– Reactive system
Can be mechanically converted into a program
Other example:
– String parsing, pre-processor

Hey, you, show demos!
Java Process Creation

+ “fork” and “exec” rolled into \texttt{exec()}:
  \begin{verbatim}
  public Process exec(String command)
  \end{verbatim}
  – args separated by whitespace

+ Child Process output: status:
  – \texttt{getOutputStream()} - \texttt{waitFor()}
  – \texttt{getInputStream()} - \texttt{exitValue()}
  – \texttt{getErrorStream()} - \texttt{destroy()}

+ Depends upon underlying OS proc support!

Process Scheduler

<table>
<thead>
<tr>
<th>cat</th>
<th>ls</th>
<th>...</th>
<th>disk</th>
<th>vid</th>
</tr>
</thead>
</table>

+ All services are processes
+ Small scheduler handles interrupts, stopping and starting processes

Process Control Block

+ Each process has a PCB
  – state
  – program counter
  – registers
  – memory management
  – …

+ OS keeps a table of PCB’s, one per process
+ (Hey! Simple Operating System, “system.h“)

Question

+ Usually the PCB is in OS memory only.
+ Assume we put the PCB into a processes address space.
+ What problems might this cause?

Interrupt Handling

+ Stores program counter
+ Loads new program counter
  – jump to interrupt service procedure
+ Save PCB information
+ Set up new stack
+ Set “waiting” process to “ready”
+ Re-schedule (probably awakened process)
+ If new process, called a context-switch

Context Switch

+ Pure overhead
+ Fast, fast, fast
  – typically 1 to 1000 microseconds
+ Sometimes special hardware to speed