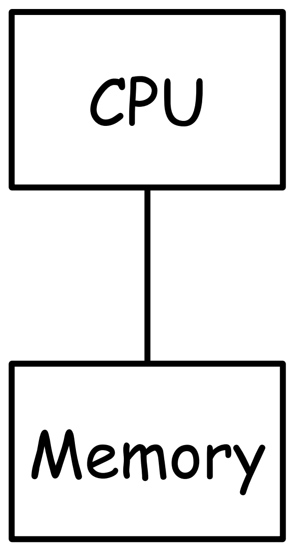
Processes

Introduction: Von Neuman Model

- Both program and data reside in memory
- Execution stages in CPU:
  - Fetch instruction
  - Decode instruction
  - Execute instruction
  - Write back result



- OS is just a program
  - OS code and data reside in memory too
  - Invoke OS functionality through system calls

Execution Mode

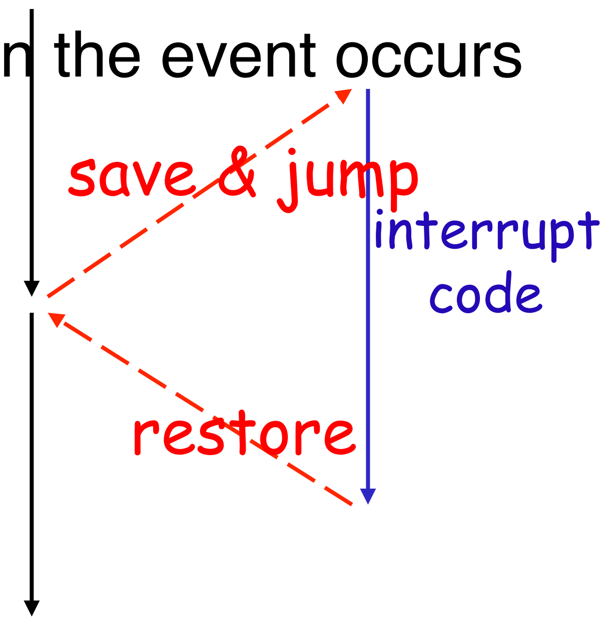
- Two modes of execution for protection reasons
  - Privileged - kernel-mode
  - Non-privileged - user-mode
- Kernel executes in kernel-mode
  - Access hardware resources
  - Protected from interference by user programs
  - Portion of the OS
- User code executes in user-mode
- OS functionality that does not need direct access to hardware may run in user-mode
  - Microkernel design basis

Interrupts and traps

- Interrupt: an asynchronous event
  - External event
    - Independent instruction execution in the processor
    - E.g. DMA completion
  - Can be masked (specifically or not)
- Trap: a synchronous software event
  - Synchronous event
    - Caused by the execution of the current instruction
    - E.g. system calls, floating point error
  - Conditional or unconditional

More Interrupts and Traps

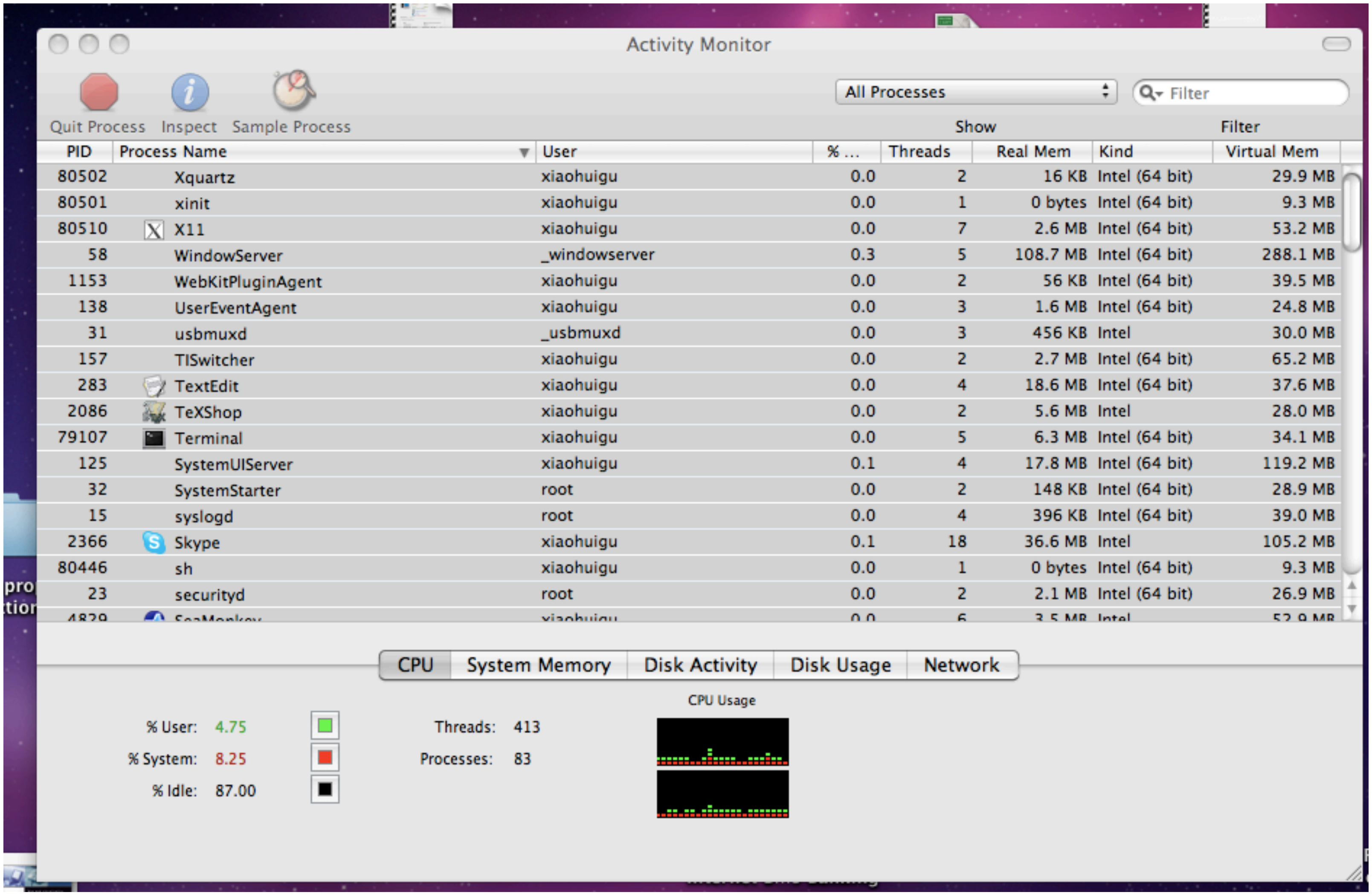
- Interrupt and trap events
  - Statically defined (typically as integers)
  - Each interrupt and trap has an associated interrupt vector
  - Interrupt vector specifies handler
    - Code that should be called when the event occurs
- At interrupt or trap, processor
  - Saves current state of execution
  - Jumps to the handler



Process

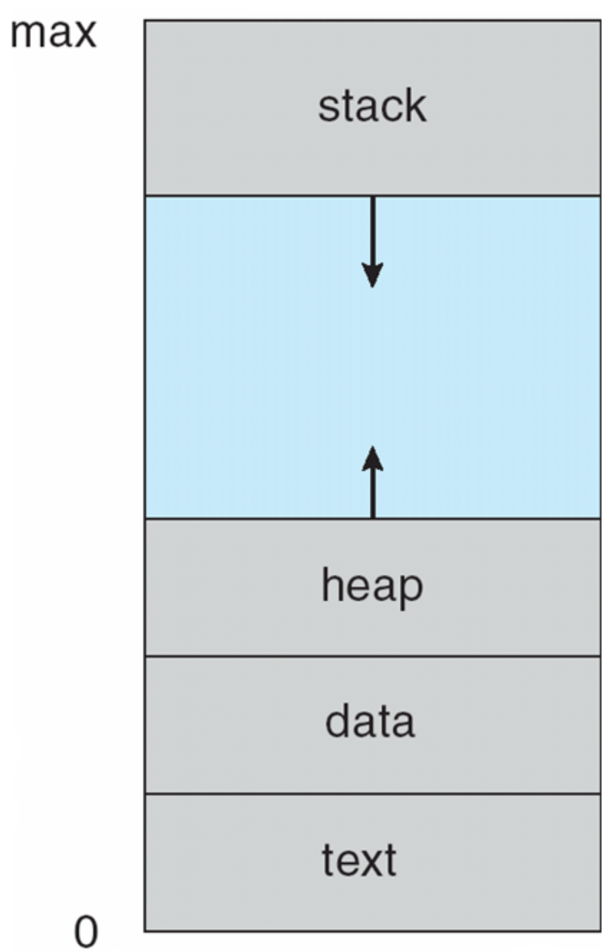
- Process – a program in execution; an "instantiation" of a program
- A process includes:
  - program counter
  - stack
  - data section

Processes



Activity Monitor
Quit Process
Inspect
Sample Process
All Processes
Filter
Show
Filter
PID
Process Name
User
% ...
Threads
Real Mem
Kind
Virtual Mem
80502 Xquartz xiaohuigu 0.0 2 16 KB Intel (64 bit) 29.9 MB
80501 xinit xiaohuigu 0.0 1 0 bytes Intel (64 bit) 9.3 MB
80510 X11 xiaohuigu 0.0 7 2.6 MB Intel (64 bit) 53.2 MB
58 WindowServer _windowserver 0.3 5 108.7 MB Intel (64 bit) 288.1 MB
1153 WebKitPluginAgent xiaohuigu 0.0 2 56 KB Intel (64 bit) 39.5 MB
138 UserEventAgent xiaohuigu 0.0 3 1.6 MB Intel (64 bit) 24.8 MB
31 usbmuxd _usbmuxd 0.0 3 456 KB Intel 30.0 MB
157 TISwitcher xiaohuigu 0.0 2 2.7 MB Intel (64 bit) 65.2 MB
283 TextEdit xiaohuigu 0.0 4 18.6 MB Intel (64 bit) 37.6 MB
2086 TeXShop xiaohuigu 0.0 2 5.6 MB Intel 28.0 MB
79107 Terminal xiaohuigu 0.0 5 6.3 MB Intel (64 bit) 34.1 MB
125 SystemUIServer xiaohuigu 0.1 4 17.8 MB Intel (64 bit) 119.2 MB
32 SystemStarter root 0.0 2 148 KB Intel (64 bit) 28.9 MB
15 syslogd root 0.0 4 396 KB Intel (64 bit) 39.0 MB
2366 Skype xiaohuigu 0.1 18 36.6 MB Intel 105.2 MB
80446 sh xiaohuigu 0.0 1 0 bytes Intel (64 bit) 9.3 MB
23 securityd root 0.0 2 2.1 MB Intel (64 bit) 26.9 MB
CPU
System Memory
Disk Activity
Disk Usage
Network
CPU Usage
% User: 4.75
% System: 8.25
% Idle: 87.00
Threads: 413
Processes: 83

Process in Memory



Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Example PCB in XINU

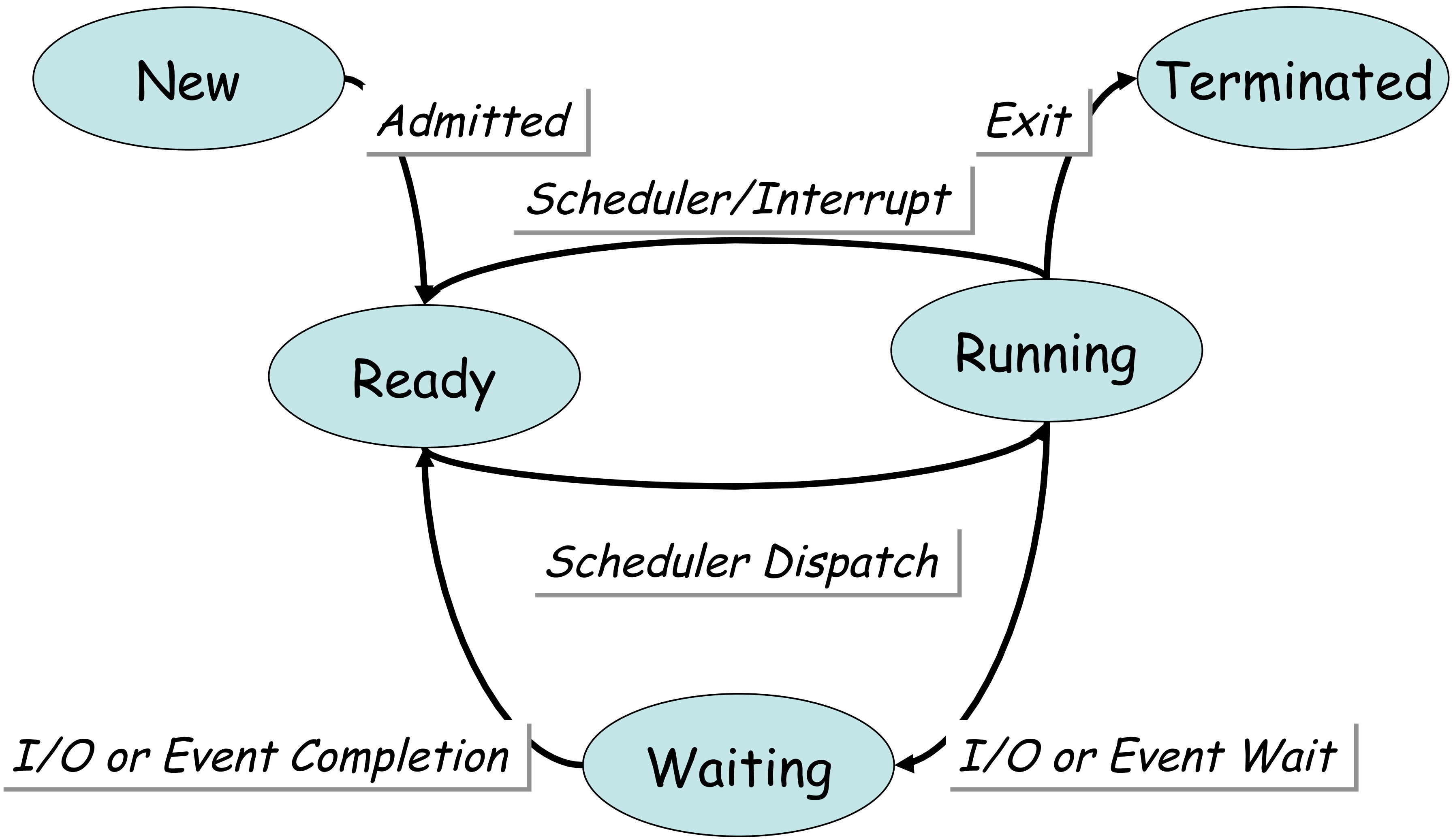
```
/* excerpt from file proc.h */

struct  pentry  {                           /* process table entry       */
        char    pstate;                     /* process state: PRCURR, etc. */
        int     pprio;                      /* process priority          */
        int     pesp;                       /* saved stack pointer       */
        STATWORD pirmask;                   /* saved interrupt mask      */
        int     psem;                       /* semaphore if process waiting */
        WORD    pmsg;                       /* message sent to this process */
        char    phasmsg;                    /* nonzero iff pmsg is valid */
        WORD    pbase;                      /* base of run time stack    */
        int     pstklen;                    /* stack length              */
        WORD    plimit;                     /* lowest extent of stack    */
        char    pname[PNMLEN];              /* process name              */
        int     pargs;                      /* initial number of arguments */
        WORD    paddr;                      /* initial code address      */
        short   pdevs[2];                   /* devices to close upon exit */
        int     fildes[_NFILE];             /* file - device translation */
};
extern  struct  pentry proctab[];
extern  int     numproc;                    /* currently active processes */
extern  int     nextproc;                   /* search point for free slot */
extern  int     currpid;                    /* currently executing process */
```

Process State

- As a process executes, it changes ***state***
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

Process Lifecycle

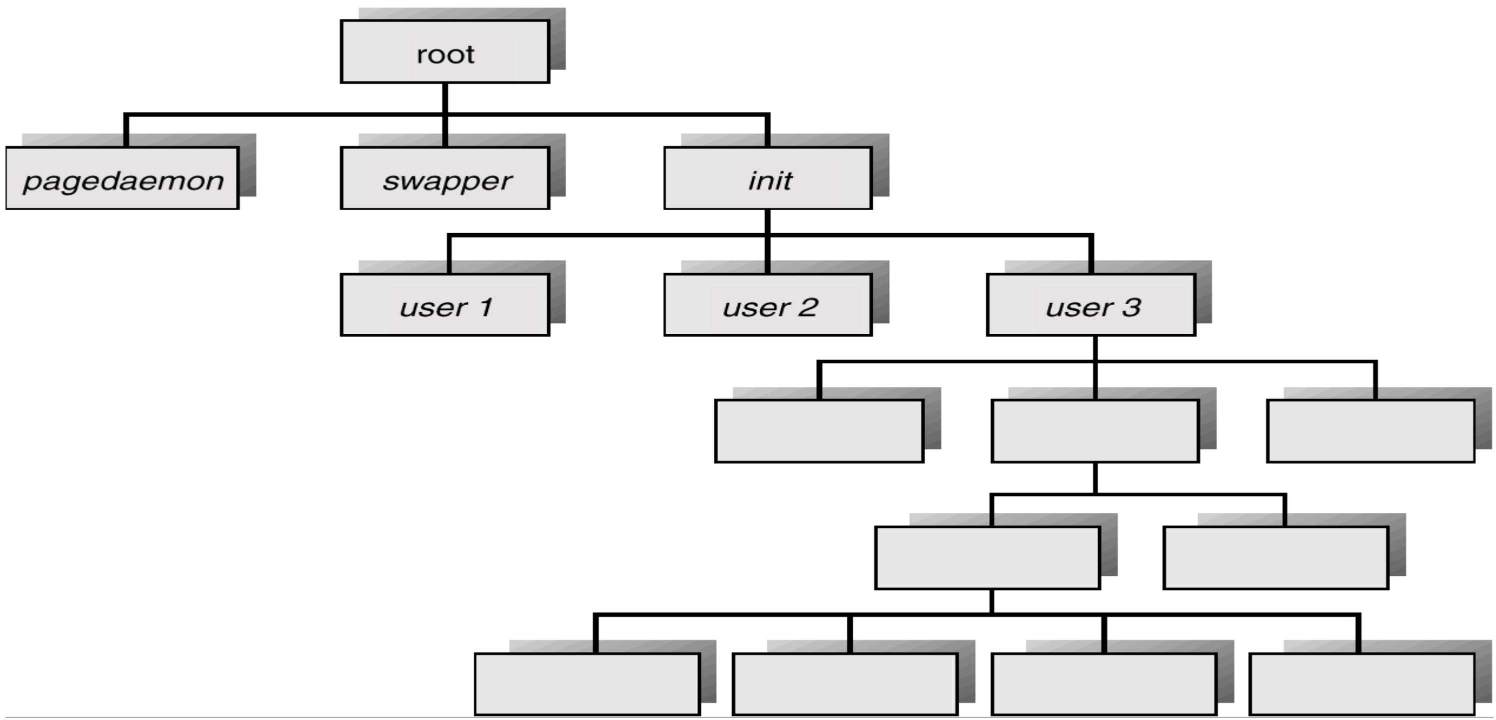


Process Manipulation

- Performed by OS routines
- Example operations
  - Creation
  - Termination
  - Suspension
  - Resumption
- State variable in process table records activity

Process Creation

- Parent process creates children processes,
  - Which, in turn create other processes,
  - Forming a tree of processes



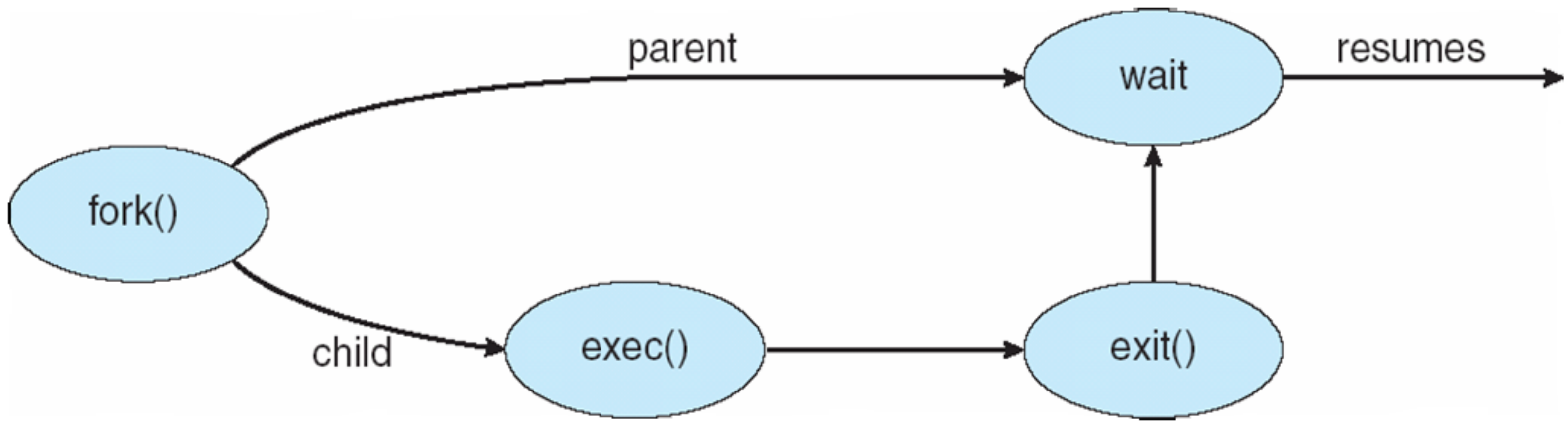
Process Creation

- Policy on resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Policy on execution
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Policy on address space
  - Child duplicate of parent
  - Child has a program loaded into it

Process Creation (Cont.)

- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace new process’ memory space with a new program

Process Creation



C Program Forking Separate Process

```
int main()
{
pid_t  pid;
   /* fork another process */
   pid = fork();
   if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
   }
   else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
   }
   else { /* parent process */
        /* parent will wait for the child to
   complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
   }
}
```

Process Termination

- Possible scenarios for process termination
  - Exit (by itself)
  - Abort (by parent)
  - Kill (by sysadmin)
- Exit
  - Process executes last statement and asks operating system to delete

Process Termination

- Abort
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
    - All children terminated - ***cascading termination***
- Kill
  - Administration purpose

Process Suspension

- Temporarily “stop” a process
  - Prohibit from using the CPU
- Why?
- What should be done?
  - Change its state in PCB
  - Save its machine states for later resumption
    - Process table entry retained
    - Complete state saved

Context Switch

- When CPU switches to another process
- System must
  – Save the state of the old process (suspend) and
  – Load the saved state for the new process (resume)
- Context-switch time is overhead

  System does no useful work while switching
- Time dependent on hardware support

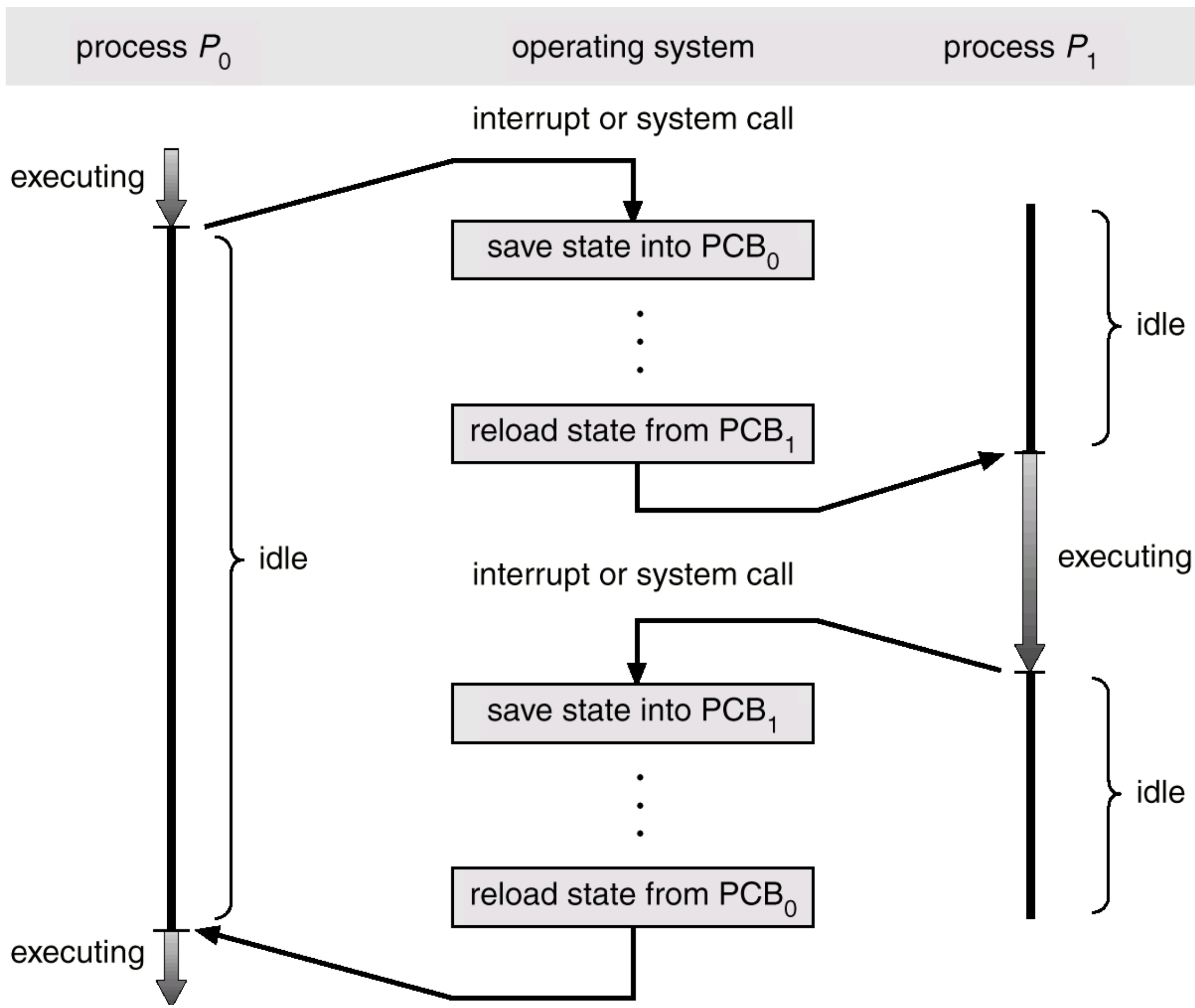
Context Switching

- How to do a context switch?
  - Very carefully!!
- Save state of currently executing process
  - Copy all “live” registers to process control block
  - Need at least 1 scratch register -- points to area of memory in process control block that registers should be saved to
- Restore state of process to run next
  - Copy values of live registers from process control block to registers
- How to get into and out of the context switching code?

Context Switching

- OS is just code stored in memory, so ...
  - Call context switching subroutine
  - The subroutine saves context of current process, restores context of the next process to be executed, and returns
  - The subroutine returns in the context of another (the next) process!
  - Eventually, will switch back to the current process
  - To process, it appears as if the context switching subroutine just took a long while to return

CPU Switch From Process to Process



Xinu Implementation

- Read relevant source code in Xinu
  - Process queue management
    - h/q.h sys/queue.c sys/insert.c, …
  - Proc. creation/suspension/resumption/termination:
    - sys/create.c, sys/suspend.c sys/resume.c, sys/kill.c
  - Process scheduling
    - sys/resched.c
  - Other initialization code

Next Lecture

- Thread