

CS241 System Programming Process

Klara Nahrstedt

Lecture 4

1/27/06

Overview

- Process State
- Process Information
- Context Switching
- Summary

Administrative

- Read T: Chapter 2.1
- Read R&R: Chapter 3

Processes

- A **process** is an abstraction for sequence of operations that implement a computation. A process may be manipulated, suspended, scheduled and terminated
- Process is a unit of work in a modern computer
- Process Types:
 - OS processes executing system code
 - User processes executing user code
- Processes are executed concurrently with CPU multiplexing among them

Process: An Object with State

- OS manipulated a process as if it is an object with a state.
- The operations that OS performs on the process change the state of the process.

Process States

- The process has states during its lifetime: running, waiting, ready, suspended, ended
- State of a process is defined by the current process activity
- Operations on the process may depend on its state---

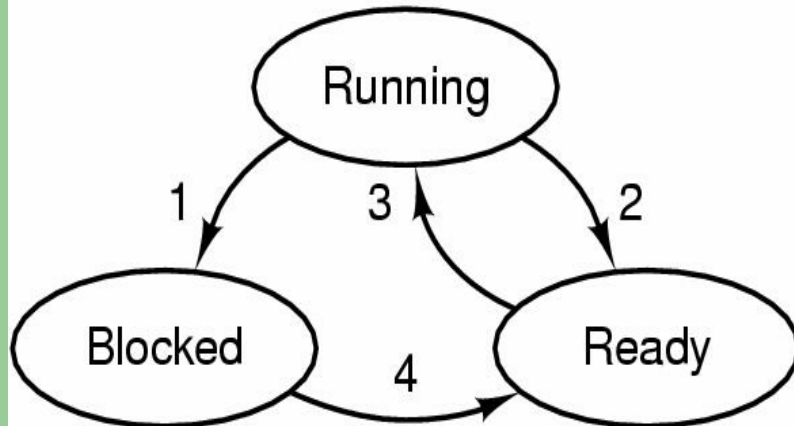
Major Process States

- Active
 - **Running:** On CPU - instructions in the program code are being executed;
- Waiting or halted - process is waiting for some event to occur;
 - **Start (new):** before execution - process is being created;

Major Process States

- **Ready:** waiting for CPU - process is waiting to be assigned to the CPU;
- **Blocked (waiting):** waiting for an event
- **Suspended:** waiting to be swapped in
- **Dead (terminated):** finished or aborted

Process States



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

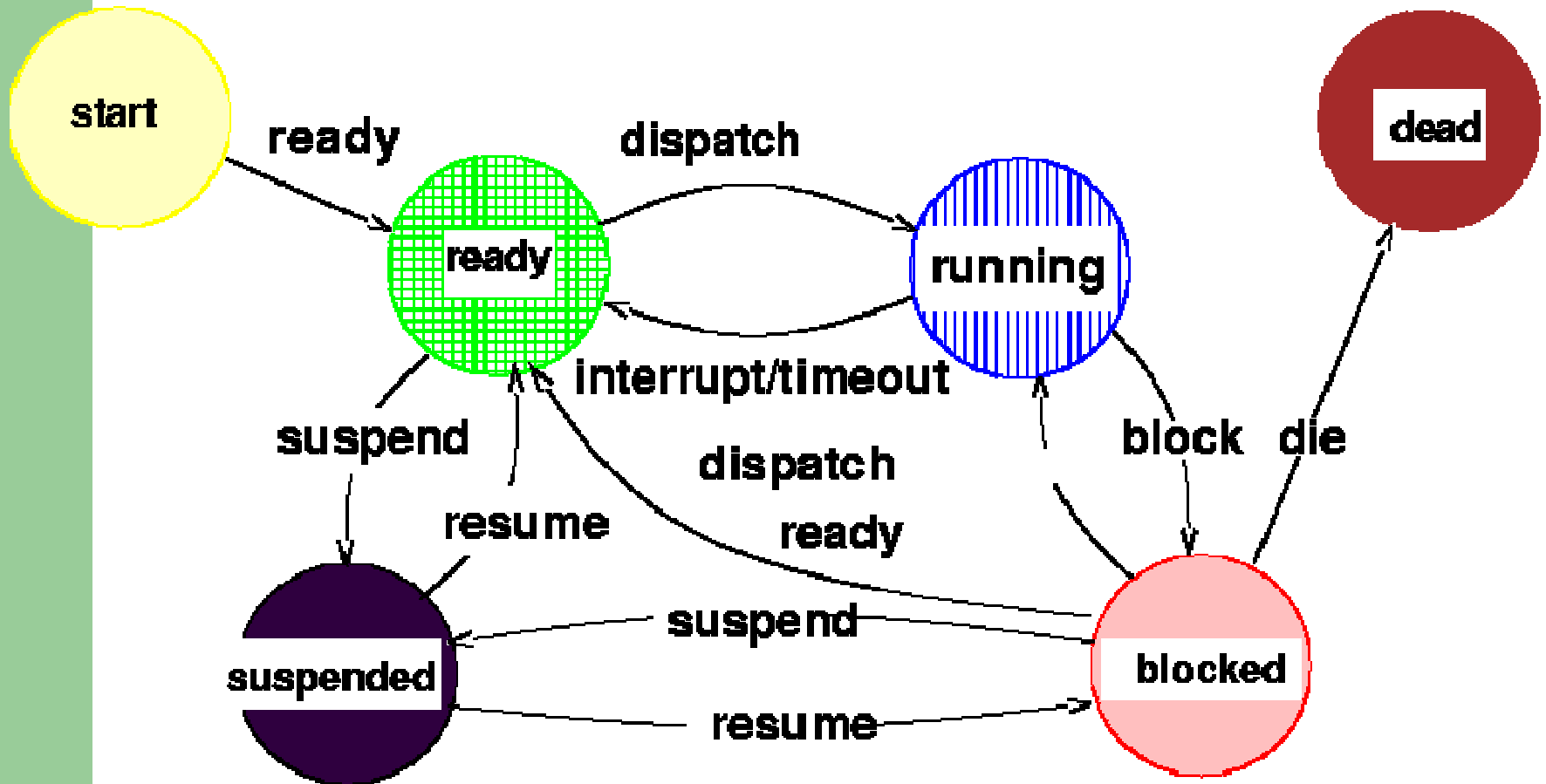
- Possible process states

- Running (occupy CPU)
- Blocked
- Ready (does not occupy CPU)
- Other states: suspended, terminated

- Transitions between states

- Question: in a single processor machine, how many processes can be in running state?

Process States Finite State Diagram



State Transitions

Admitted	New	to	Ready
Dispatch	Ready	to	Running
Preemptive Dispatch (I/O or Sync)	Blocked	to	Running
Relinquish Processor	Running	to	Ready
<i>Time Slice Run Out</i>	Running	to	Ready
Block (I/O Request or Sync)	Running	to	Blocked
Ready (I/O Complete or Sync)	Blocked	to	Ready
Suspend	Running	to	Suspended
	Ready	to	Suspended
	Blocked	to	Suspended
Resume	Suspended	to	Ready
	Suspended	to	Blocked
Halt	Blocked	to	Dead

So What Is A Process?

- It's one executing instance of a "program"
- It's separate from other instances
- It can start ("launch") other processes
- It can be launched by them

So What's In A Process? And Why?

- Process State
 - new, ready, running, waiting, halted;
- Program Counter
 - the address of the next instruction to be executed for this process;
- CPU Registers
 - index registers, stack pointers, general purpose registers;
- CPU Scheduling Information
 - process priority and pointer;
- Memory Management Information
 - base/limit information;
- Accounting Information
 - time limits, process number; owner
- I/O Status Information
 - list of I/O devices allocated to the process;

What Does This Program Do?

```
int myval;
int main(int argc, char *argv[])
{
    myval = atoi(argv[1]);
    while (1)
        printf("myval is %d, loc 0x%lx\n",
               myval, (long) &myval);
}
```

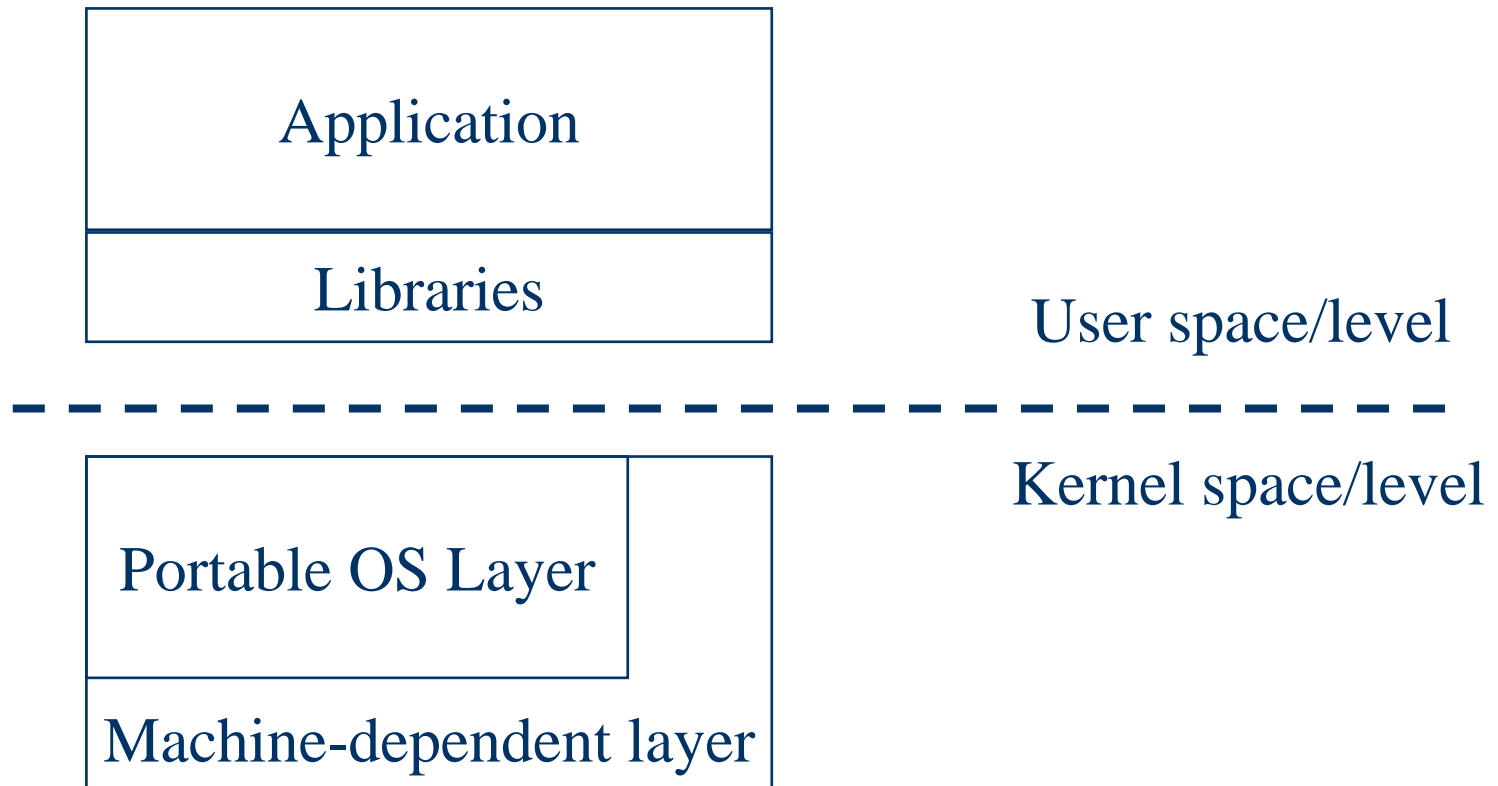
Now simultaneously start two instances of this program

- Myval 5
- Myval 6
- **What will the outputs be?**

Instances Of Programs

- The address was always the same
- The values were different
 - Implies that the programs aren't seeing each other
 - But they think they're using the same address
- Conclusion: addresses are not absolute
- Implication: memory mapping
- What's the benefit?

Remember This?



Address Space

- One (common) approach
 - Kernel is high memory
 - User is low memory
- What restrictions apply?
- `read(f, buf, nbytes)`



More Address Space

- Program segments
 - Text
 - Data
 - Stack
 - Heap
- Lots of flexibility
 - Allows stack growth
 - Allows heap growth
 - No predetermined division



Process Control Block (PCB)

Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment Pointer to data segment Pointer to stack segment	Root directory Working directory File descriptors User ID Group ID

Fields of a process table entry

Process Scheduling

- **Objective of multiprogramming** – maximal CPU utilization, i.e., have always a process running
- **Objective of time-sharing** – switch CPU among processes frequently enough so that users can interact with a program which is running
- **Need Context Switching**

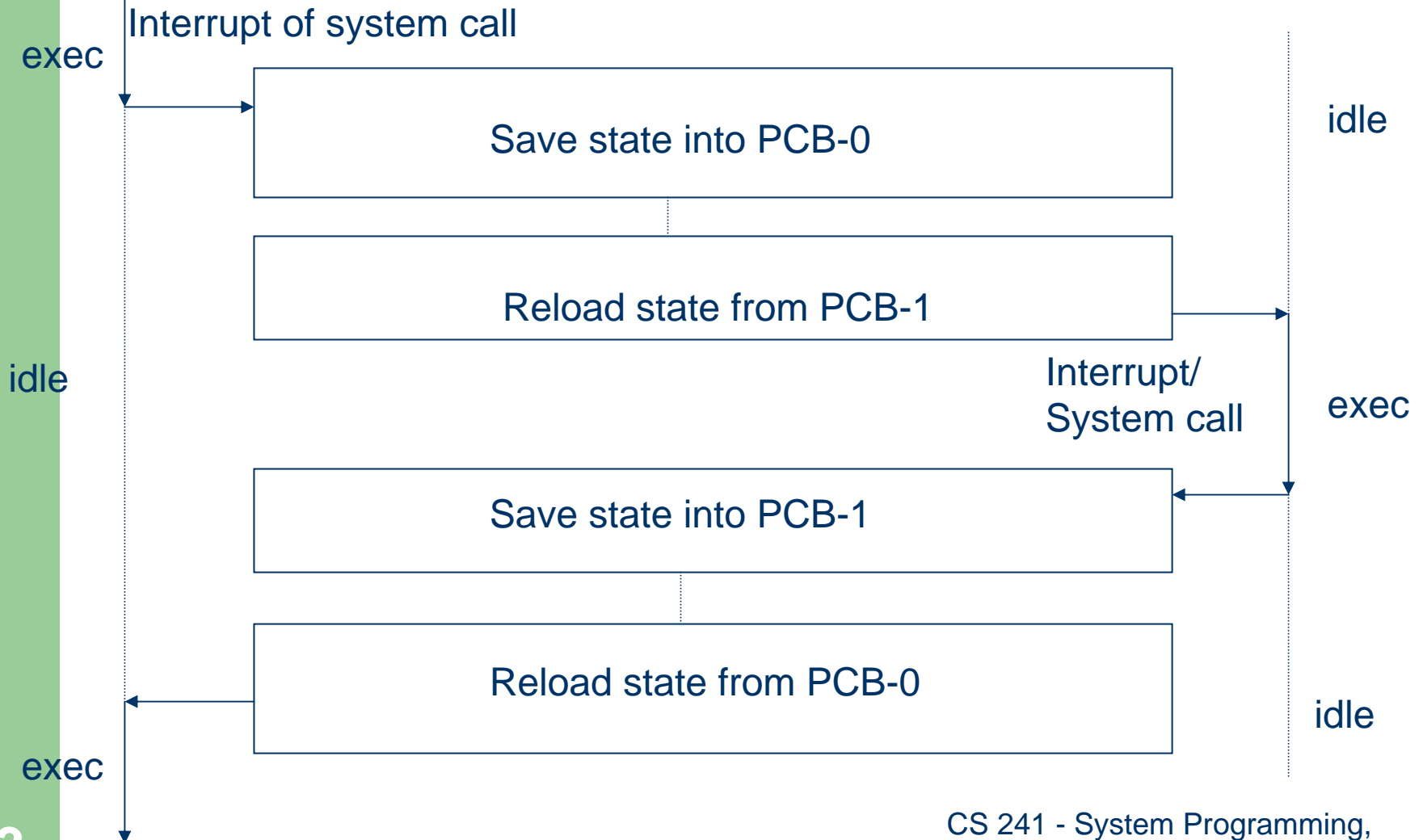
Context Switch

- Switch CPU from one process to another
- Performed by scheduler
- It includes:
 - save PCB state of the old process;
 - load PCB state of the new process;
 - Flush memory cache;
 - Change memory mapping (TLB);
- **Context switch is expensive**(1-1000 microseconds)
 - No useful work is done (pure overhead)
 - Can become a bottleneck
 - **Real life analogy?**
- Need hardware support

Process Context Switch

PCB-0

PCB-1



Interrupts

- **Interrupt** is an electrical signal generated by a peripheral device
- A Device generates interrupt to set a **hardware flag** within the processor
- At each instruction cycle, processor checks hardware flags to see if any peripheral devices need attention
- If interrupt is detected, processor saves the current value of the program counter and loads a new value – address of a special function – **interrupt service routine or interrupt handler**

Interrupts to Implement Time-Sharing

- A device called **timer** can generate interrupt after a specified interval of time.
- Process Management OS starts the timer before setting the program counter – set **quantum**
- When **timer expires**, it generates an interrupt that causes the CPU to execute timer interrupt service routine
- Interrupt service routine writes the address of the OS code into the program counter, and the OS is back in control
- CPU switch occurs, old process loses the CPU and new process starts

Interrupt Processing

- How is the Illusion of multiple sequential processes with one CPU and many I/O devices maintained?
- Each I/O device class is associated with location, called **interrupt vector**
- Interrupt vector includes
 - Address of interrupt service procedure
- When interrupt occurs:
 - Save registers into process table entry for the current process (assembly)
 - Info pushed onto the stack by the interrupt is removed and the stack pointer is set to point to a temporary stack used by process handler (assembly)
 - Call interrupt service (e.g., to read and buffer input), process is done (C-language)
 - Scheduler decides which other process to run next (C-language)
 - Start to run assembly code to load registers, etc. (assembly)

Summary

- What is process
 - Process states and model
 - What information in a process
 - Process address space
 - Context switch
 - Next lecture: thread
-
- Read T: Chapter 2.1
 - Read R&R: Chapter 3