SE 292: High Performance Computing [3:0][Aug:2014]

Process Management

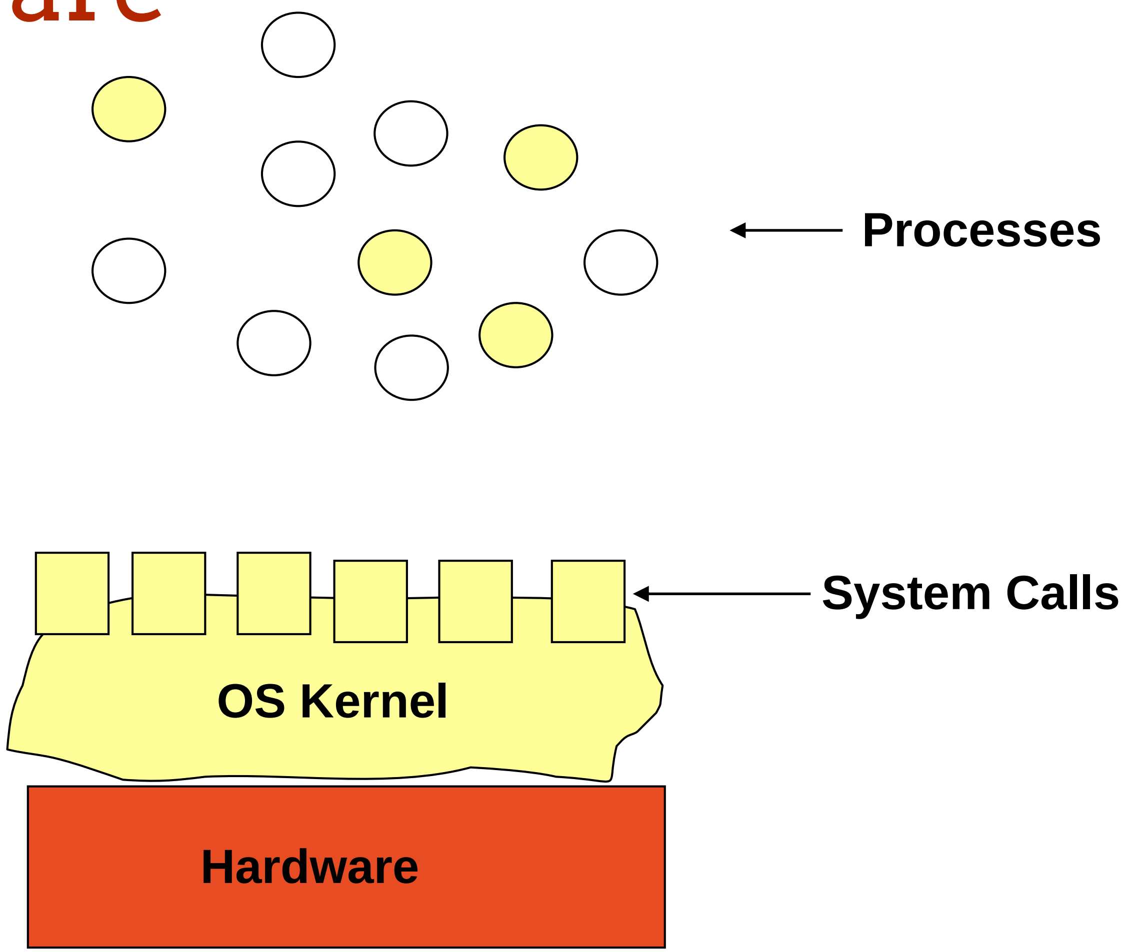
Yogesh Simmhan

Adapted from "Memory Organization and Process Management", Sathish Vadhiyar, SE292 (Aug:2013), "Operating Systems Concepts", Silberschatz, Galvin & Gagne, 2005 & Computer Systems: A Programmer's Perspective", by R.E. Bryant and D. O'Hallaron, 2003

Computer Organization: Software

- Hardware resources of computer system are shared by programs in execution
- Operating System: special program that manages this sharing
  - Ease-of-use, resource allocator, device controllers
- Process: a program in execution
  - **ps** tells you the current status of processes
- Shell: a command interpreter through which you interact with the computer system
  - csh, bash,…

Operating System, Processes, Hardware



Operating System

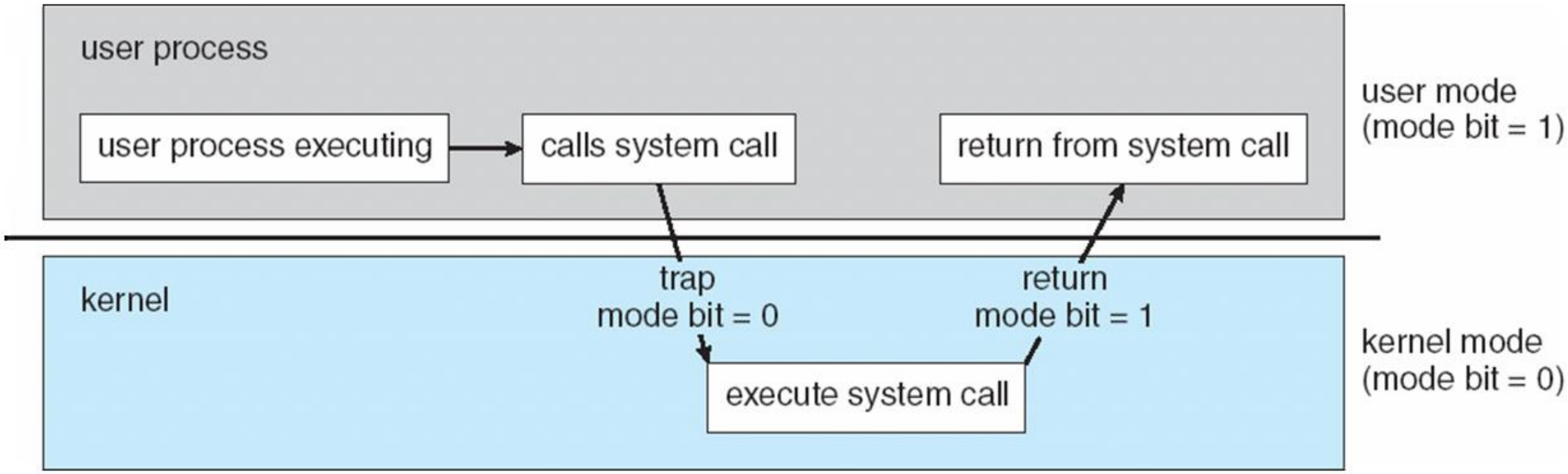
Software that manages the resources of computer system

- CPU time
- Main memory
- I/O devices

- OS functionalities
  - Process management
  - Memory management ✓
  - Storage management

Process Lifetime

- Two modes during execution
  - **User** – when executing on behalf of user application
  - **Kernel mode** – when user application requests some OS service, some privileged instructions
- Implemented using mode bits



Silberschatz – Figure 1.10

Modes

- Can find out the total CPU time used by a process, as well as CPU time in user mode, CPU time in system mode

Shell - What does it do?

```
while (true){
   Prompt the user to type in a command      write
   Read in the command             read
   Understand what the command is asking for
   Get the command executed   fork, exec
}
```

Q: What system calls are involved?

- Shell – command interpreter
- Shell interacts with the user and invokes system call
- Its functionality is to obtain and execute next user command
- Most of the commands deal with file operations – copy, list, execute, delete etc.
- It loads the commands in the memory and executes

BASH Shellshock

- Vulnerability in BASH command shell
  - Detected in Sep 24, 2014
  - Impact Subscore: **10.0**, Access Complexity: **Low**
  - http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271
- Causes special values in env variables to be executed as command in call to child BASH shell

```
env x='() { :;}; echo vulnerable' bash -c
“echo this is a test”
```

http://en.wikipedia.org/wiki/Shellshock_%28software_bug%29

System Calls

- How a process get the operating system to do something for it; an Application Program Interface (API) for interaction with the operating system
- Examples
  - File manipulation: **open, close, read, write,…**
  - Process management: **fork, exec, exit,…**
  - Memory management: **sbrk,…**
  - device manipulation – **ioctl, read, write**
  - information maintenance – **date, getpid**
  - communications – **pipe, shmget, mmap**
  - protection – **chmod, chown**
- When a process is executing in a system call, it is actually executing Operating System code
- System calls allow transition between modes

Mechanics of System Calls

- Process must be allowed to do sensitive operations while it is executing system call
- Requires hardware support
- Processor hardware is designed to operate in at least 2 modes of execution
  - **Ordinary**, user mode
  - **Privileged**, system mode
- System call entered using a special machine instruction (e.g. MIPS 1 **syscall**) that switches processor mode to system before control transfer
- System calls are used all the time
  - Accepting user's input from keyboard, printing to console, opening files, reading from and writing to files

System Call Implementation

- Implemented as a trap to a specific location in the interrupt vector (interrupting instructions contains specific requested service, additional information contained in registers)
- Trap executed by **syscall** instruction
- Control passes to a specific service routine
- System calls are usually not called directly - There is a mapping between a API function and a system call
- System call interface intercepts calls in API, looks up a table of system call numbers, and invokes the system calls

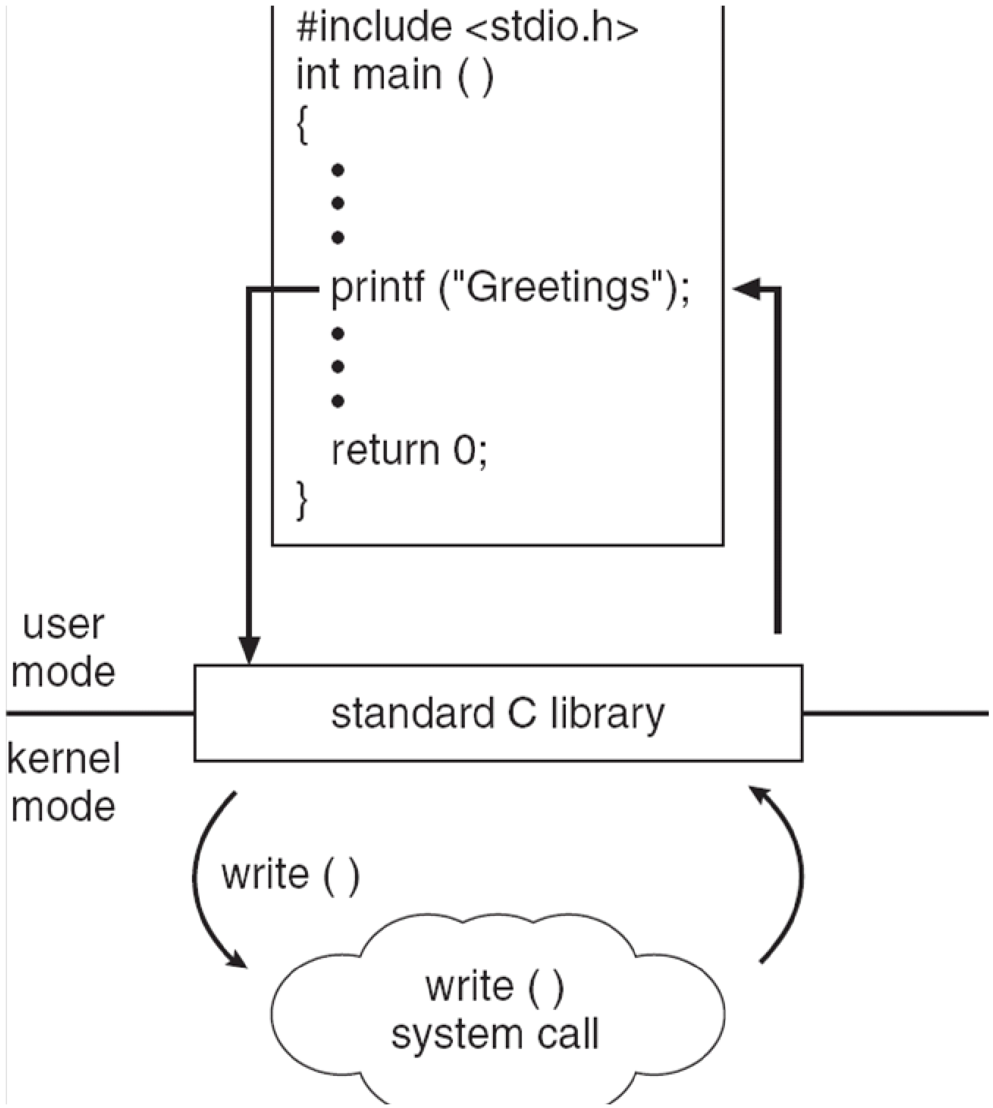
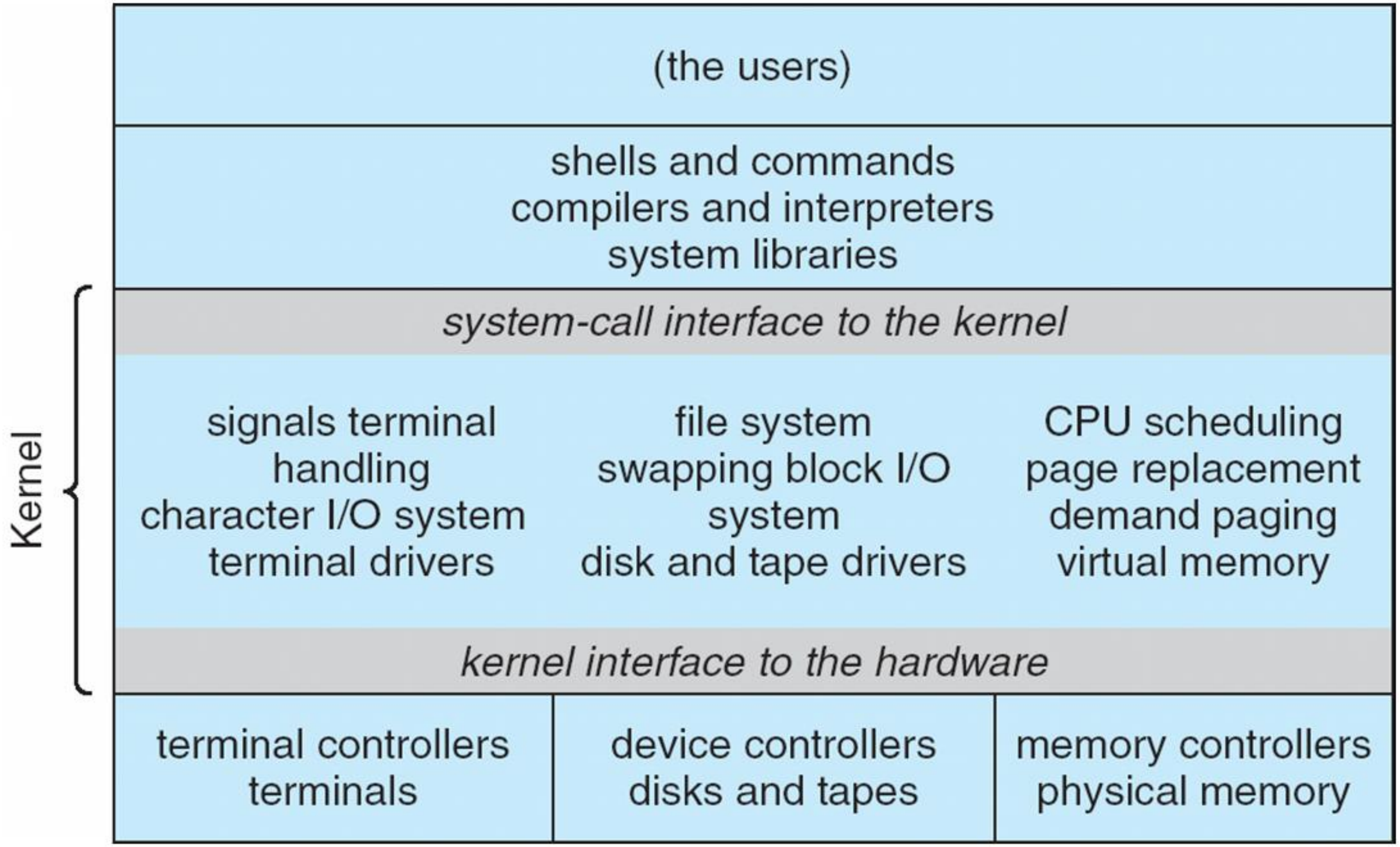


Figure 2.6 (Silberschatz)

Traditional UNIX System Structure



System Boot

- Bootstrap loader – a program that locates the kernel, loads it into memory, and starts execution
- When CPU is booted, instruction register is loaded with the bootstrap program from a pre-defined memory location
- Bootstrap in **ROM (firmware)**
- Bootstrap – initializes various things (mouse, device), starts OS from **boot block** in disk
- Practical:
  - BIOS – boot firmware located in (EP)ROM
  - Loads bootstrap program from Master Boot record (MBR) in the hard disk
  - MBR contains GRUB; GRUB loads OS*
- OS then runs init and waits

* http://www.gnu.org/software/grub/manual/multiboot/multiboot.html

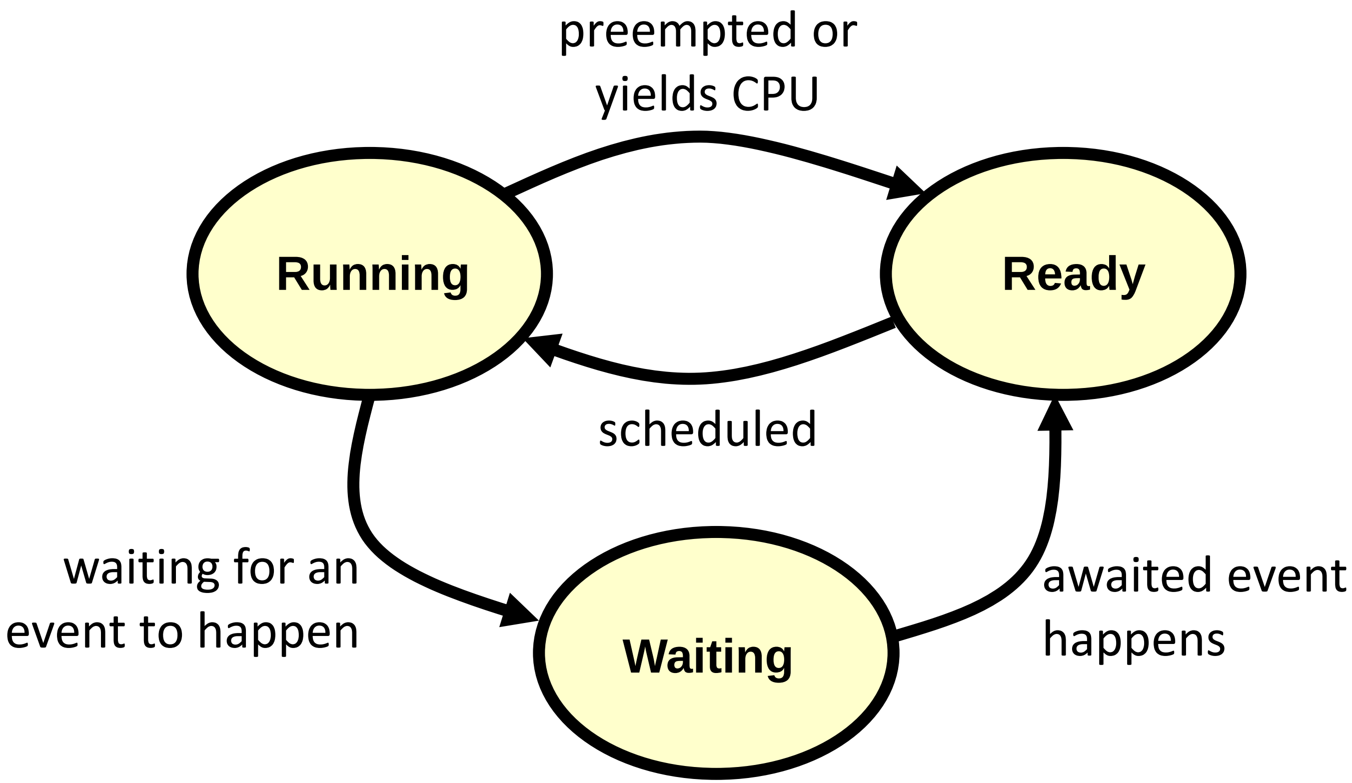
Process Management

- What is a Process?
  - Unit of work in “time sharing” systems
    - **Job** is unit of work in “Batch Processing” systems
  - A program or an application *in execution*
  - But some programs run as multiple processes
  - And instance of same program can be run by multiple processes at same time

Process vs Program

- Program: static, passive, dead
- Process: dynamic, active, living
- Process changes state with time
- Possible states a process could be in?
  - Running (Executing on CPU)
  - Ready (to execute on CPU)
  - Waiting (for something to happen)

Process State Transition Diagram



Process States

- Ready – waiting to be assigned to a processor
- Waiting – waiting for an event

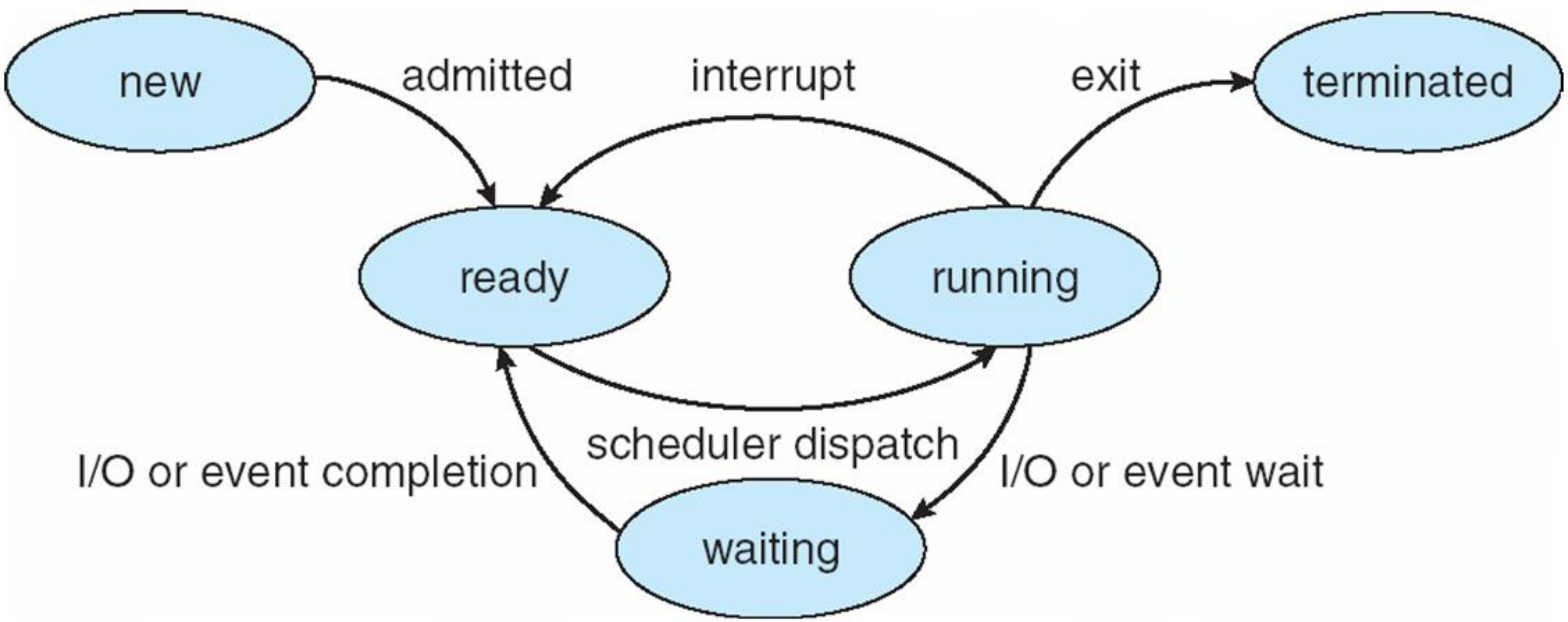
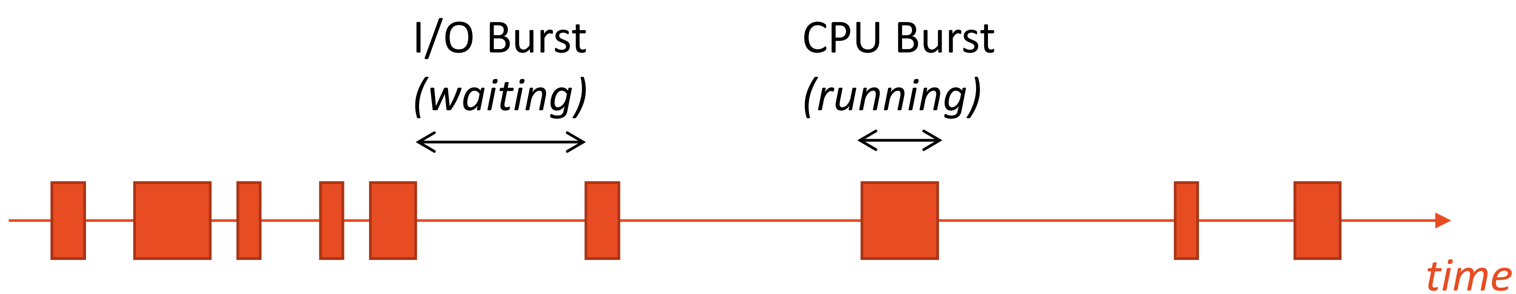


Figure 3.2 (Silberschatz)

CPU and I/O Bursts

- Processes alternate between two states of CPU burst and I/O burst.
- There are a large number of short CPU bursts and small number of long I/O bursts



Process Control Block

- Process represented by Process Control Block (PCB). Contains:
- **Process state**
  - text, data, stack, heap
  - Hardware – PC value, CPU registers
- Other information maintained by OS:
- **Identification** – process id, parent id, user id
- **CPU scheduling information** – priority
- **Memory-management information** – page tables etc.
- **Accounting information** – CPU times spent
- **I/O status information**
- Process can be viewed as a data structure with operations like fork, exit, etc. and the above data

PCB and Context Switch

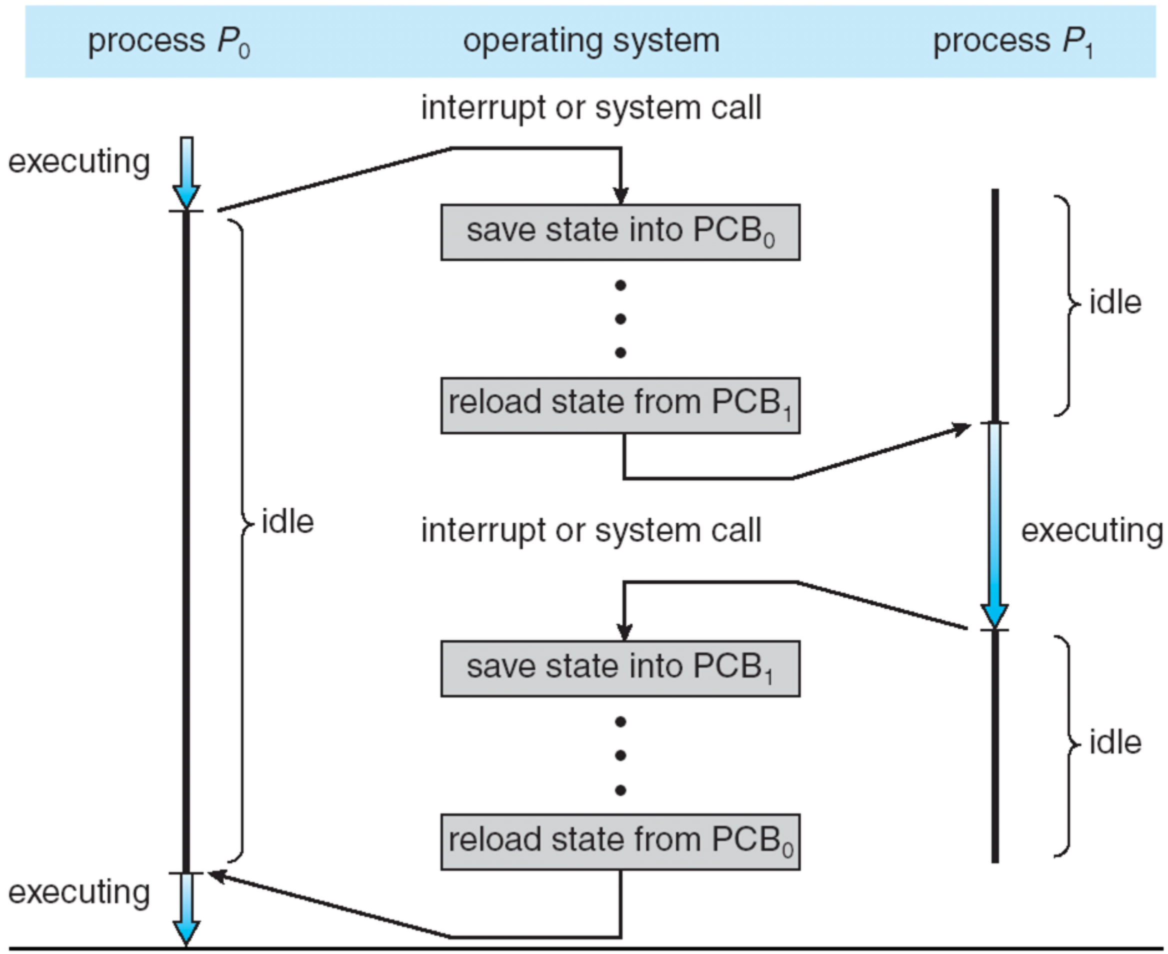


Fig. 3.4
(Silberschatz)

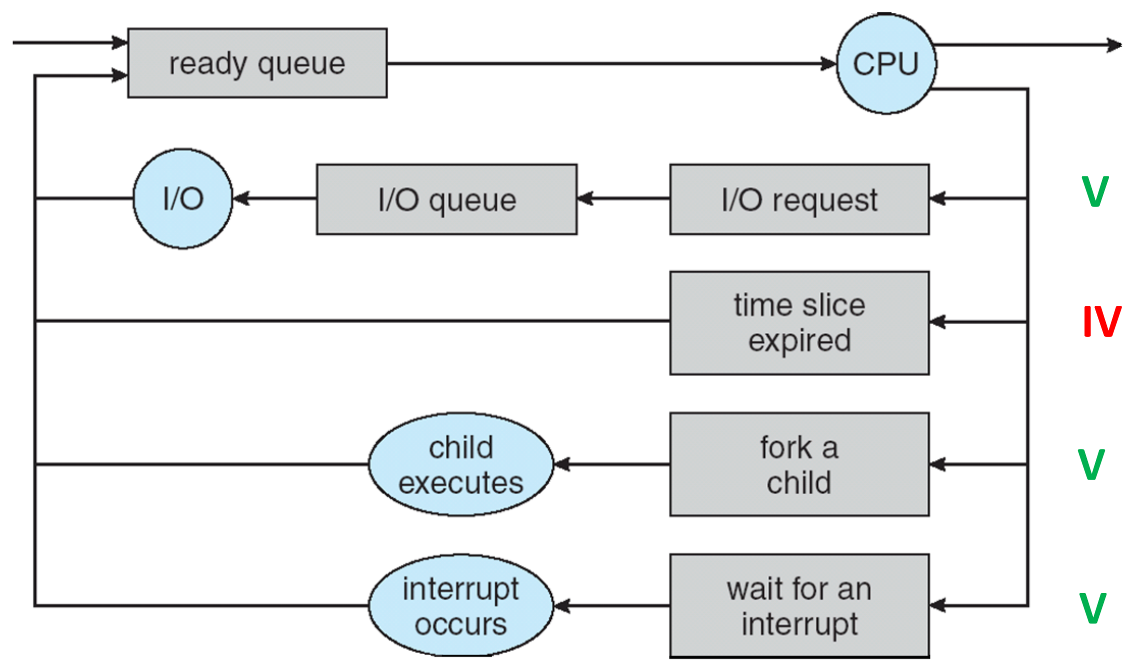
Process Management

- *What should OS do when a process does something that will take a long time?*
  - e.g., file read/write operation, page fault, …
  - Devices may themselves be in demand
- Objective: Maximize utilization of CPU
  - Change status of process to `Waiting' and make another ready process `Running'
- Which process?
- Objectives:
  - Minimize average program execution time
  - Ensure Fairness

Process Scheduling

- Selecting a process from many available ready processes for execution
- A process residing in memory and waiting for execution is placed in a ready queue
- Can be implemented as a linked list
- Other devices (e.g. disk) can have their own queues

[V] Voluntarily give up CPU
[IV] Involuntarily have CPU taken away



Queue diagram

Fig. 3.7 (Silberschatz)

Scheduling Criteria

- CPU utilization
- Throughput
- Turnaround time
- Waiting time
- Response time
- Fairness

Scheduling Policies

- Preemptive vs Non-preemptive
  - Preemptive policy: one where OS `preempts' the running process from the CPU even though it is not waiting for something…*Involuntary*
  - Idea: give a process some maximum amount of CPU time before preempting it, for the benefit of the other processes
  - CPU time slice: amount of CPU time allotted
  - In a non-preemptive process scheduling policy, process would yield CPU either due to waiting for something or *voluntarily*

Process Scheduling Policies

- Non-preemptive
  - First Come First Served (FCFS)
  - Shortest Process Next
- Preemptive
  - Round robin
  - Preemptive Shortest Process Next *(shortest remaining time first)*
  - Priority based
    - Process that has not run for more time could get higher priority
    - May even have larger time slices for some processes

Recommended Reading

- Process Management
  - Chapter 2: System Structures, Silberschatz 7th Ed.
  - Chapter 3: Processes, Silberschatz 7th Ed.

Multilevel Feedback

- Used in some kinds of UNIX
- Multilevel: Priority based (preemptive)
  - OS maintains one ready Q per priority level
  - Schedules from front of highest priority non-empty queue
- Feedback: Priorities are not fixed
  - Process moved to lower/higher priority queue for fairness

Linux Kernel: Scheduling

- Linux assigns dynamic priorities for non real-time processes
- Long running processes have their priorities decreased
- Waiting processes have priorities increased dynamically
- Compute-bound versus I/O bound
  - Linux favours I/O bound processes over compute *(why?)*
- Another classification:
  - Interactive processes. Shells, text editors, GUI apps
  - Batch processes. Compilers, DB indexers, number-crunching
  - Real-time processes. A/V apps, sensors, robot controllers

http://cs.boisestate.edu/~amit/teaching/597/scheduling.pdf
http://oreilly.com/catalog/linuxkernel/chapter/ch10.html

Linux Kernel: Scheduling

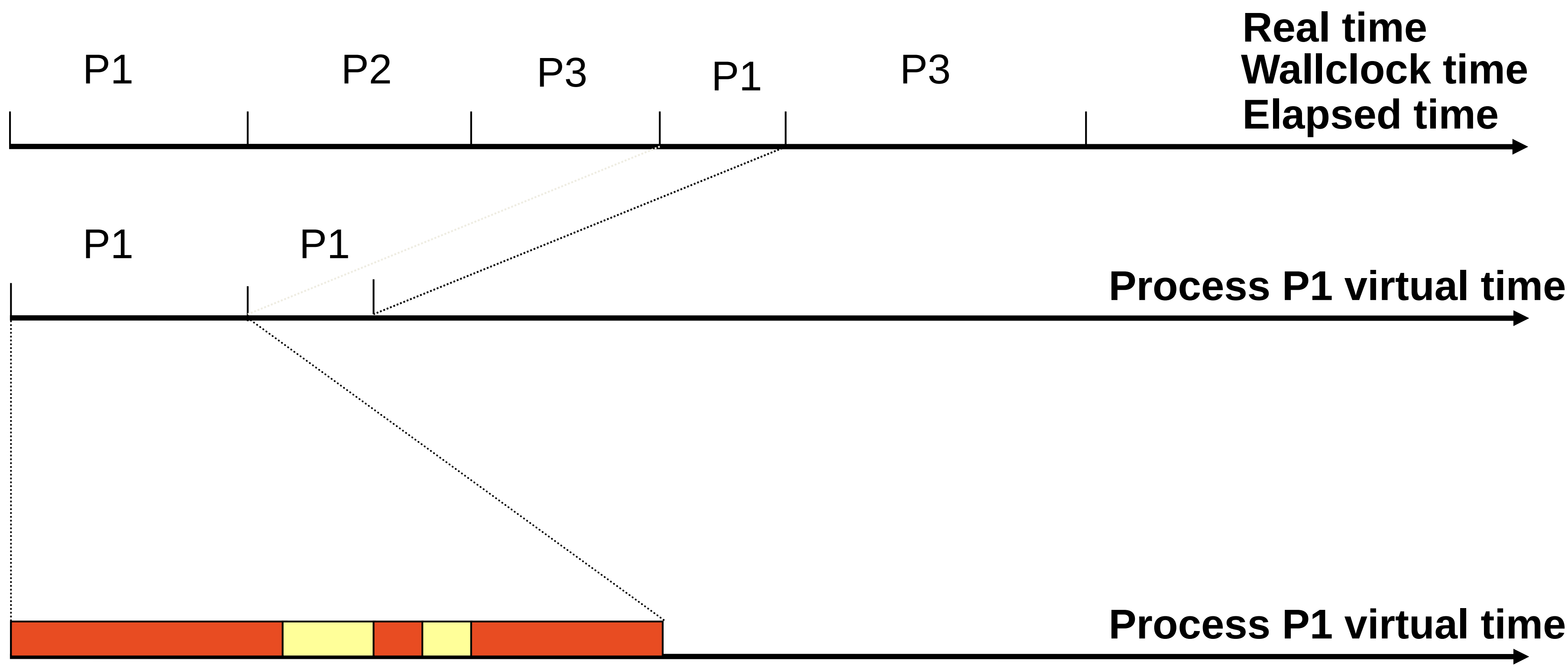
- Algorithm divides CPU time into *epochs*
- Each process has a specified time quantum computed when the epoch begins
- A process can be selected several times by the scheduler in the same epoch
  - As long as its quantum is not exhausted
- *Base time quantum*: Default assigned to a process that's exhausted its previous quantum. E.g. 210 ms
- Users can change the base time quantum using the **nice**( ) and **setpriority**( ) system calls

http://oreilly.com/catalog/linuxkernel/chapter/ch10.html, Linux 2.4 kernel scheduler

Context Switch

- When OS changes process that is currently running on CPU
- Takes some time, as it involves replacing hardware state of previously running process with that of newly scheduled process
  - Saving HW state of previously running process
  - Restoring HW state of scheduled process
- Amount of time would help in deciding what a reasonable CPU timeslice value would be

Time: Process virtual and Elapsed



: Running in user mode

: Running in system mode

How is a Running Process Pre-empted?

- OS preemption code must run on CPU
  - How does OS get control of CPU from running process to run its preemption code?
- Hardware timer interrupt
  - Hardware generated periodic event
  - When it occurs, hardware automatically transfers control to OS code (timer interrupt handler)
  - Interrupt is an example of a more general phenomenon called an exception

Exceptions

- Certain exceptional events during program execution that are handled by processor HW
- Two kinds of exceptions
  - Traps: Synchronous, software generated
    - Page fault, Divide by zero, System call
  - Interrupts: Asynchronous, hardware generated
    - Timer, keyboard, disk

What Happens on an Exception

1. Hardware
   - Saves processor state
   - Transfers control to corresponding piece of OS code, called the **exception handler**
2. Software (exception handler)
   - Takes care of the situation as appropriate
   - Ends with **return from exception** instruction
3. Hardware (execution of RFE instruction)
   - Restores the saved processor state
   - Transfers control back to saved PC value

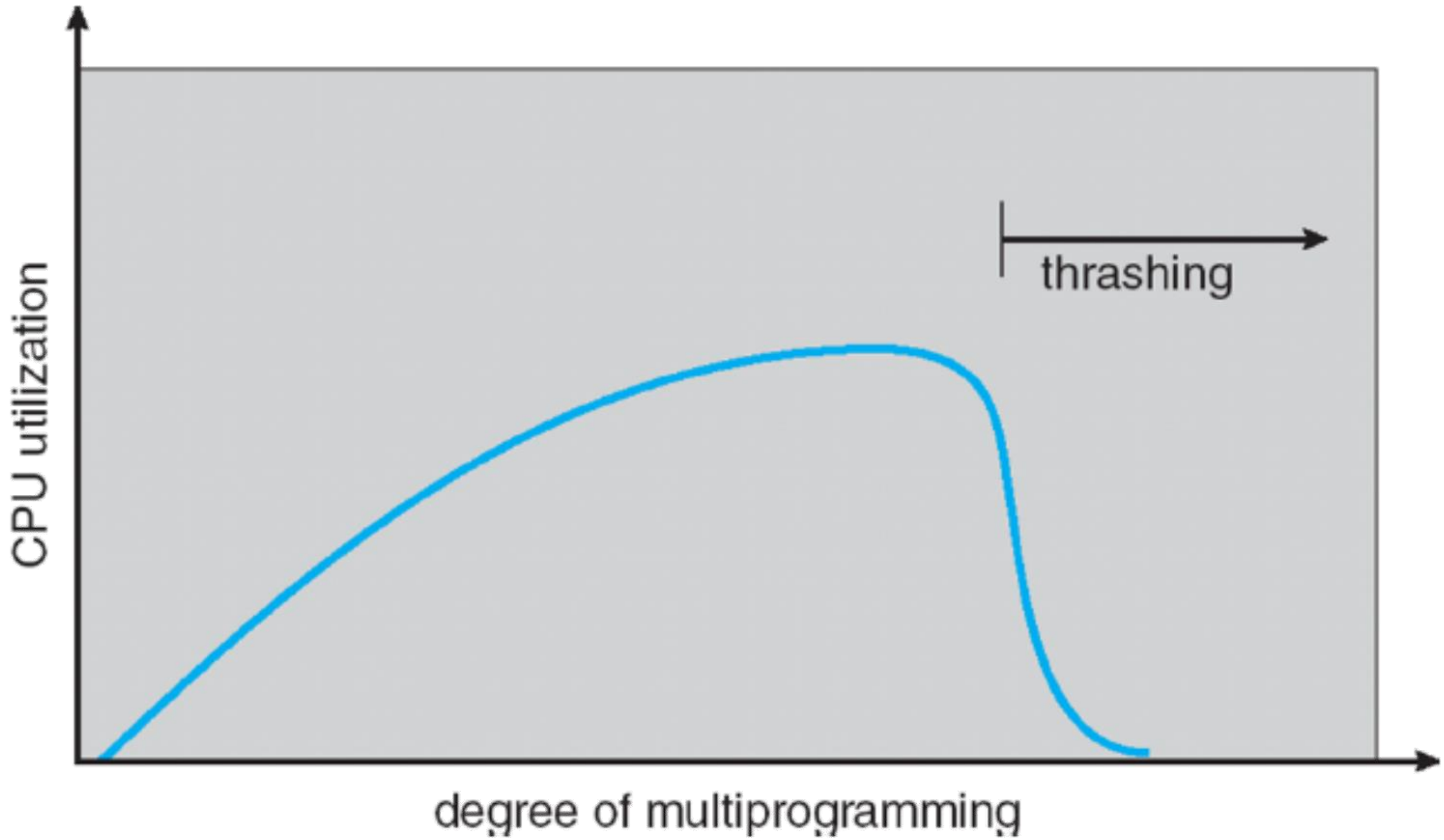
Re-look at Process Lifetime

- Which process has the exception handling time accounted against it?
  - Process running at time of exception
- All interrupt handling time while process is in running state is accounted against it
  - Part of `running in system mode'

Thrashing

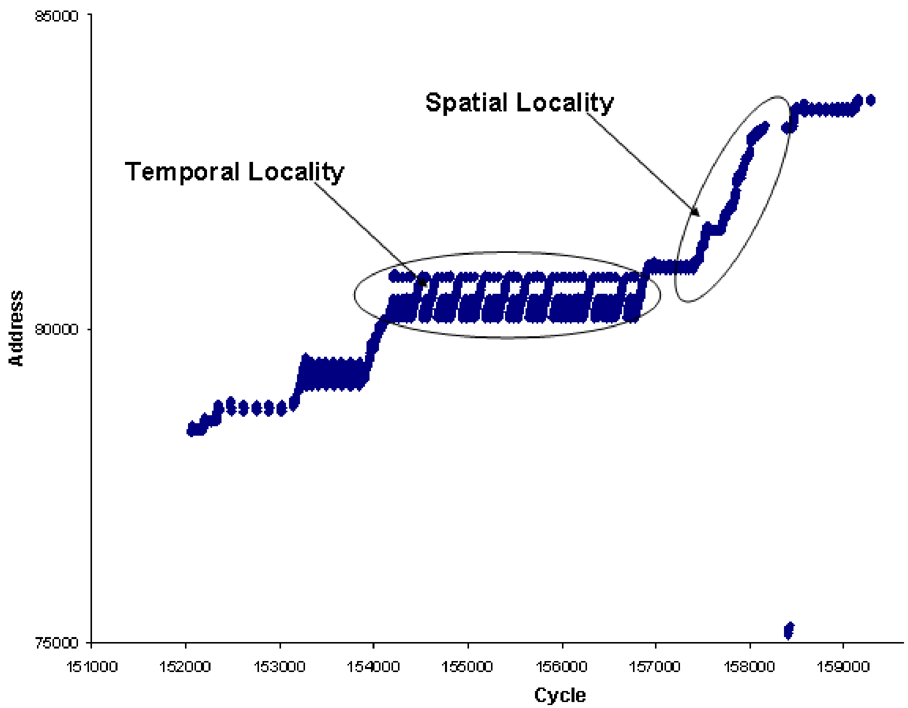
- When CPU utilization decreases, OS increases multiprogramming level by adding more processes
- Beyond a certain multiprogramming level, processes compete for pages leading to page faults
- Page fault causes disk reads by processes leading to lesser CPU utilization
- OS adds more processes, causing more page faults, lesser CPU utilization – cumulative effect

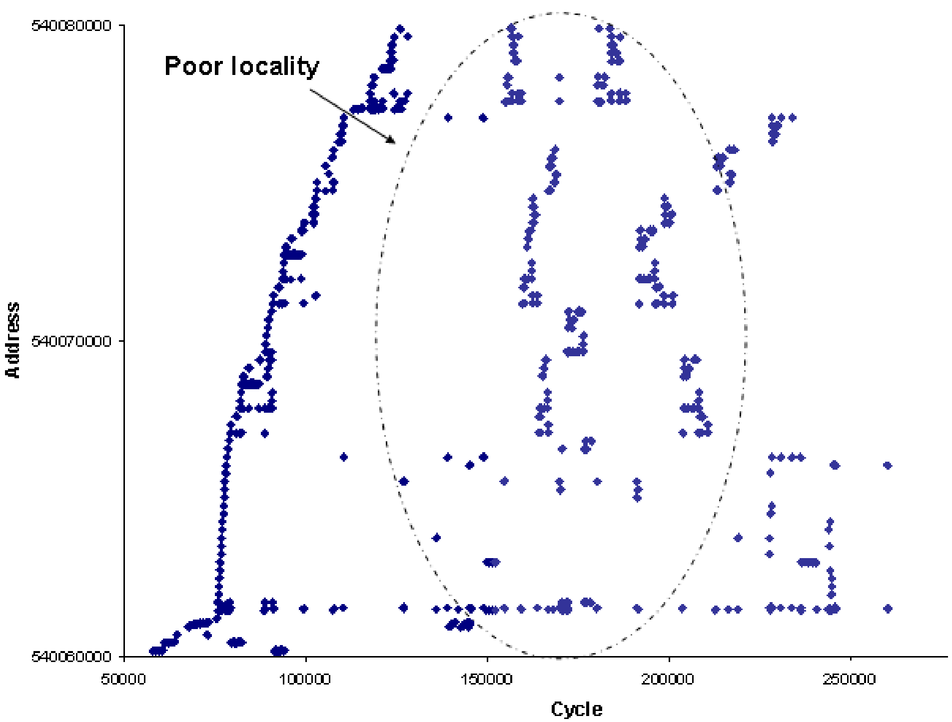
Thrashing



Silberschatz, 7th Ed. – Figure 9.18

Memory Locality





Optimizing for instruction caches,
Amir Kleen, et al, 2007,
http://www.eetimes.com/document.asp?doc_id=1275470

Working Set Model

- Conceptual model to prevent thrashing.
  - Collection of pages a process is using actively,
  - must be memory-resident to prevent it from thrashing.
- If the sum of all working sets of all runnable processes exceeds memory, pause some of the processes.
- Divide processes into two groups: active and inactive:
  - An active process's entire working set must be in memory
  - An inactive process's working set can migrate to disk.
  - Inactive processes are never scheduled for execution.
- Collection of active processes is the *balance set*.
  - Gradually moving processes into and out of the balance set.
  - As working sets change, the balance set must be adjusted.

Working Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\, WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend one of the processes

Silberschatz, 8th Ed.

Midterm II Topics

- Virtual Memory Management
  - Chapter 9: Virtual Memory, Bryant 2nd Ed.
  - Chapter 9: Virtual Memory , Silberschatz 7th Ed.
- Process Management
  - Chapter 2: System Structures, Silberschatz 7th Ed.
  - Chapter 3: Processes, Silberschatz 7th Ed.