

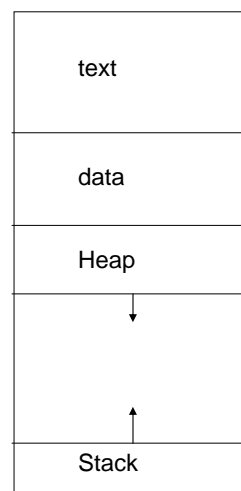
SE-292: High Performance Computing Memory Organization

R. Govindarajan
govind@serc

Use of Main Memory by a Program

- Instructions (code, text)
- Data used in different ways
 - Stack allocated
 - Heap allocated
 - Statically allocated

Use of memory addresses



Stack Allocated Variables

- Space allocated on function call, reclaimed on return
 - Addresses calculated and used by compiler, relative to the top of stack, or some other base register associated with the stack
 - Growth of stack area is thus managed by the program, as generated by the compiler
-

3

Heap Allocated Variables

- Managed by a memory allocation library
 - Functions like malloc, realloc, free
 - Get 'linked' (joined) to your program if they are called
 - Executed just like other program functions
 - What about growth of the heap area?
 - Managed by the library functions
-

4

Memory Management: Protection

- There could be many programs running on the same machine concurrently
- Sharing the resources of the computer
 - Processor time
 - Main memory
- Must be protected from each other: one program should not be able to access the variables of another
- Typically done through [Address Translation](#)
- [Read N4, N5](#)

5

Idea of Address Translation

- Each executing program uses addresses in the range 0 .. MaxAddress
- These addresses are not real, but only Virtual Addresses
- They are translated into real main memory addresses
- Many programs in execution can safely share main memory
- Terminology
 - virtual address, physical address

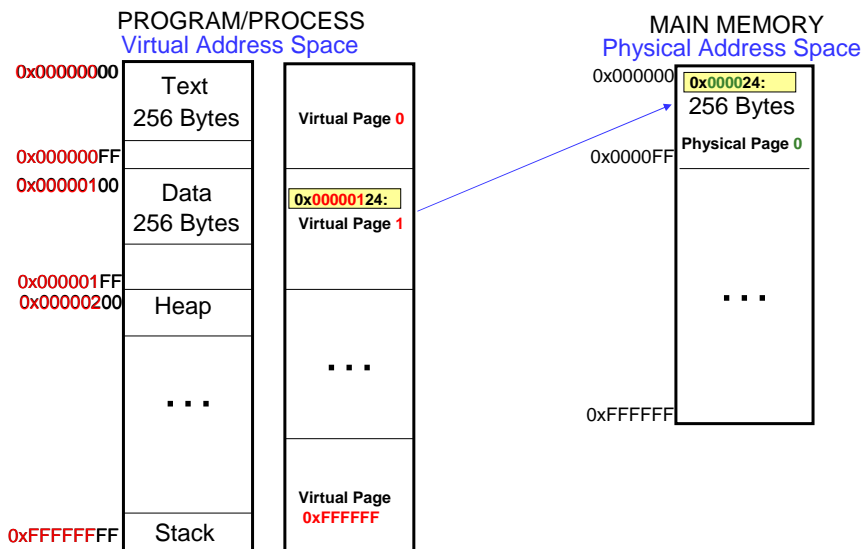
6

Example: Main Memory Management

- Example: **Paged Virtual Memory**
- To translate a virtual address to the corresponding physical address, a table of translation information is needed
- Minimize size of table by not managing translations on byte basis but larger granularity
- **Page**: fixed size unit of memory (contiguous memory locations) for which a single piece of translation information is maintained

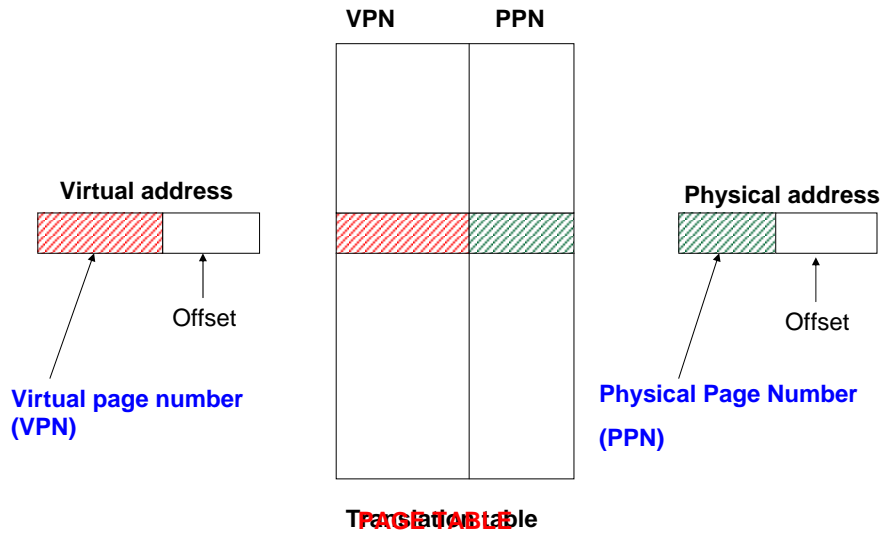
7

Paged Virtual Memory



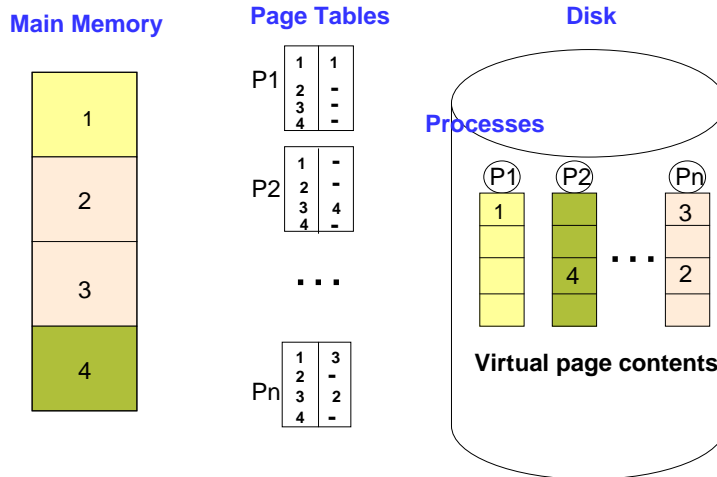
8

Address Translation



9

What's happening...



10

Page Fault

- Situation where virtual address generated by processor is not available in main memory
- Detected on attempt to translate address
 - Page Table entry is invalid
- Must be `handled' by operating system
 - Identify slot in main memory to be used
 - Get page contents from **secondary memory**
 - Part of disk can be used for this purpose
 - Update page table entry
- Data can now be provided to the processor

11

Page Replacement Policies

- Question: How does the page fault handler decide which main memory page to replace when there is a page fault?
- **Principle of Locality of Reference**
 - A commonly seen program property
 - If memory address **A** is referenced at time **t**, then **it** and its neighbouring memory locations are likely to be referenced in the near future
 - temporal
 - spatial
 - Suggests that a Least Recently Used (LRU) replacement policy would be advantageous

12

Locality of Reference

- Based on your experience, why do you expect that programs will display locality of reference?

	Same address (temporal)	Neighbours (spatial)
Program	Loop Function	Sequential code Loop
Data	Local Loop index	Stepping through array

13

Page Replacement Policies

- LRU might be too expensive to implement
- Alternatives
 - Approximation to LRU
 - First in First Out (FIFO)
 - Random

14

Implementation of Address Translation

- Easy
 - But can't be done by OS, which is software
- Memory Management Unit (MMU): part of CPU; hardware that does address translation
- Big Problem: Page Table size
 - Example: 32b virtual address, 16KB page size
 - 256K virtual pages => MB page table size per process
 - Has to be stored in memory, probably virtual address space
- Translation Lookaside Buffer (TLB): memory in MMU that contains some page table entries that are likely to be needed soon
- TLB Miss: required entry not available

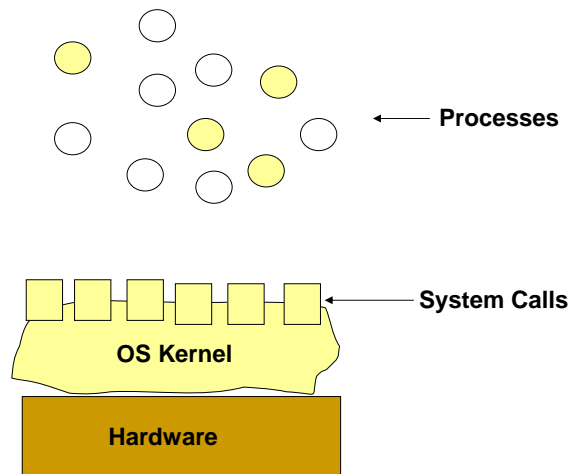
15

Computer Organization: Software

- Hardware resources of computer system are shared by programs in execution
- Operating System: special program that manages this sharing
- 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,...
 - Just another program?

16

Operating System, Processes, Hardware



17

Operating System

Software that manages the resources of a computer system

- ❑ CPU time
- ❑ Main memory
- ❑ I/O devices
- ❑ Software resources

18

Process Lifetime

- At any given point in time, a running process is executing either in user mode or in system mode
- Can find out the total CPU time used by a process, as well as CPU time in user mode, CPU time in system mode
- **HW: Read the manual entry for `time`**
- Two programs we have talked about
 - Operating system
 - Shell
- **HW: Look at the manual entry for your shell.**

19

What does a Shell 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?

End of Lecture 8

20

System Calls

- How a process gets the operating system to do something for it; an interface or API for interaction with the operating system
- Examples
 - Operations on files: open, close, read, write,...
 - Operations on processes: fork, exec, exit,...
 - Operations related to memory: sbrk,...
 - Operations related to CPU time: wait,
- When a process is executing in a system call, it is actually executing Operating System code

21

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

22

Some Homework

1. Find out how to get a **system call trace** (details of the sequence of system calls made by a program when executed) from Linux.
2. Use it to find out what system calls are used
 - a. by the dynamic memory allocation library
 - b. during file I/O
 - c. during terminal I/O
 - d. as part of program startup
 - e. as part of program termination

23

Some More Homework

Read N6 (Processes)

Write a shell program that can

- Execute a program file whose full pathname is typed as a command
 - File name could be terminated by & to indicate immediate re-prompt.
- Respond to the following shell commands:
 - prompt 'string' / change shell prompt
 - exit / terminate shell
 - !! / re-execute previous command
 - time / print current time

24

Process Management

- What is a Process?
 - Program in execution
 - But some programs run as multiple processes
 - And same program can be running multiply at same time

25

Process as a Data Structure

- Operations?
 - fork, exit, exec, ...
- Data?
 - Text, data, stack, heap
 - Hardware information
 - PC value, register values
 - Other information maintained by operating system
 - Process id, parent id, user id
 - Time of CPU used by process, in user/system
 - Memory management info: Page table
 - File related info: Open files, file pointers

26

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)

27

Process Management

- What should OS do when a process does something that will involve **a long time**?
 - e.g., file read/write operation, page fault, ...
- Objective: Maximize utilization of CPU
 - Change status of process to `Waiting` and make another process `Running`
- Which process?
- Objective?
 - Minimize average program execution time
 - Fairness

28

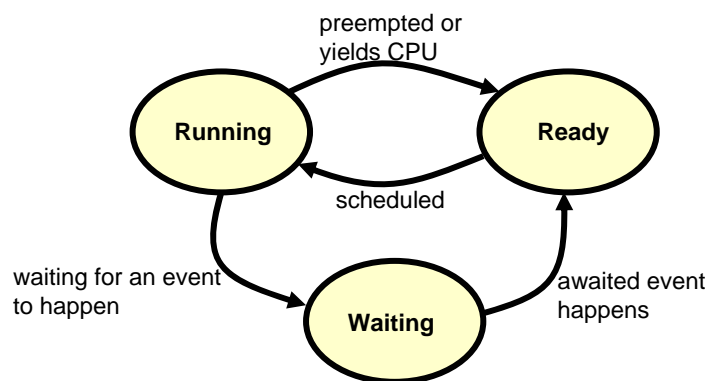
Scheduling Policies

■ Preemptive vs Non-preemptive

- ❑ Preemptive policy: one where OS `preempts' the running process from the CPU even though it is not waiting for something
- ❑ 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

29

Process State Transition Diagram



30

Process Scheduling Policies

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

31

Example: Multilevel Feedback

- Used in some kinds of UNIX
- Multilevel: Priority based (preemptive)
 - OS maintains one readyQ 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

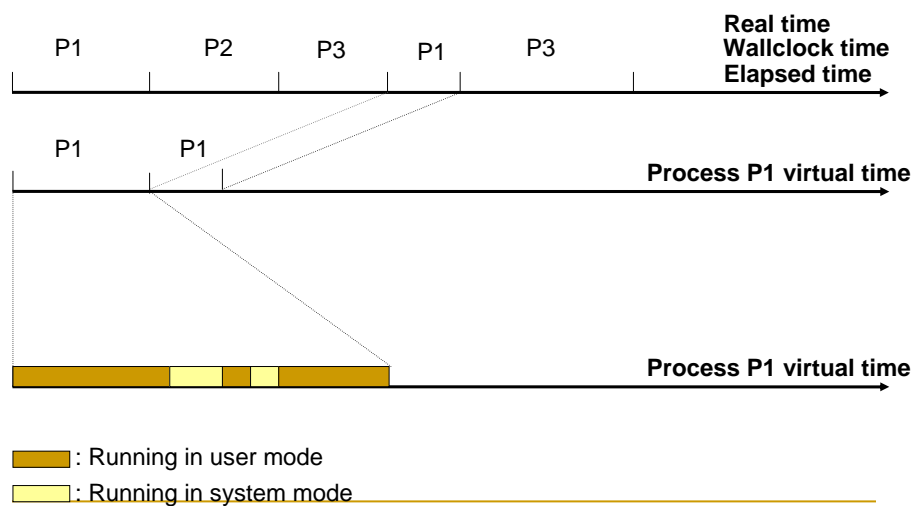
32

Context Switch

- When OS changes which process 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

33

Time: Process virtual and Elapsed



34

How is a Running Process Preempted?

- 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**

35

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

36

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

37

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'

38