1B11 Operating Systems - 3

Memory Management and Protection

Prof. Steve R Wilbur
s.wilbur@cs.ucl.ac.uk

1B11-3 1999 Slide 1

Lecture Objectives

- What does the OS have to do to manage memory?
- How does it provide required sharing when needed and yet inhibit inappropriate access?

1B11-3 1999 Slide 2

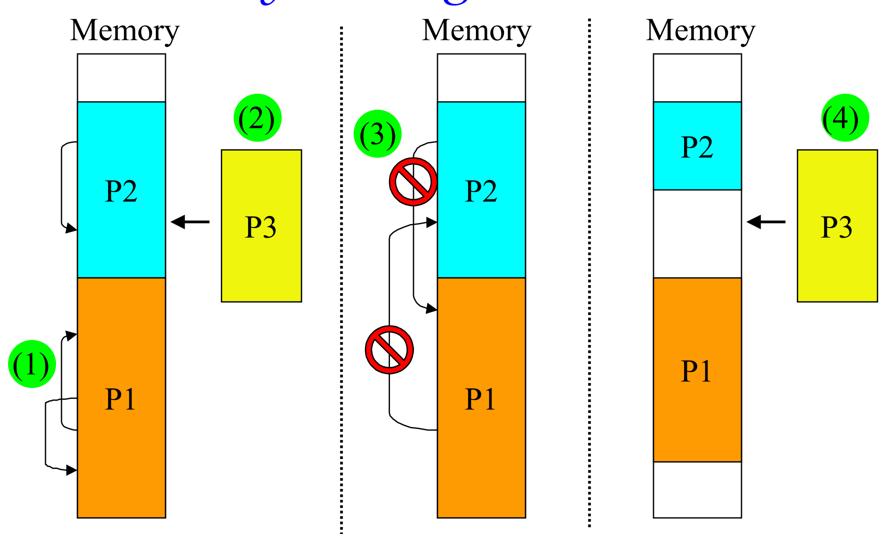
Process Life Cycle

Recap

- To start a program, the OS must:
  - Find the binary/executable on disk
  - Allocate storage for the code and data
  - Allocate swap space on disk
  - Map the pages in memory and swap space
  - Copy the code into the physical pages
  - Start execution by saving current context (see later) and jumping (loading the PC) to the start address of the code!

1B11-3 1999 Slide 3

Memory Management Problems



1B11-3 1999 Slide 4

Memory Management Problems - 2

- (1) Programs may be placed in different locations each time they are loaded - references to ***absolute addresses*** will be invalid
- (2) When memory full and need to load program need ***replacement policy*** to decide which (part of) process to swap out
- (3) Need to protect against processes damaging others - ***containment*** or ***protection***
- (4) May be enough room to load program, but no “hole” is big enough - ***fragmentation***

1B11-3 1999 Slide 5

Memory Management Solutions

- Relative addressing deals with (1), e.g.:
  - Jump ●+50 instead of Jump 456
  - Load ●-90 instead of Load 20
  - where ● = address of current instruction
- Base and Limit registers deal with (1, 3):
  - B and L registers set to address and size of process memory when it becomes current process
  - Program compiled as though starts at address 0
  - Processor adds B to addresses and checks result not less than zero or >L for every memory fetch

1B11-3 1999 Slide 6

Memory Management Solutions - 2

- Paged memory deals with (4):
  - Expensive to "shuffle up" processes
  - Page Table (multiple base registers with fixed limit) makes all holes same size
  - Page table allows fragments to be *physically separate* but have *logically contiguous* addresses

1B11-3 1999 Slide 7

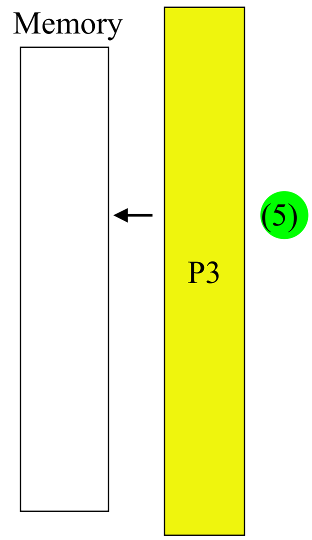
Memory Management Solutions - 3

- ***Replacement Policy*** with some help from hardware deals with (2)
  - *"dirty bit"* in page table indicates which pages have been written to and are thus more expensive to swap out
  - *"use bit"* in page table set when page accessed and cleared every few milliseconds indicates those pages which were recently used - ***Principle of Locality*** suggests they are a bad choice to swap out

1B11-3 1999 Slide 8

Memory Management Problems - 3

- (5) Program too large to fit in real memory
- Solution:
  - Use paging scheme where address space greater than memory space
  - Some part of process will always be on disk
  - will be swapped in when needed (after delay)



1B11-3 1999 Slide 9

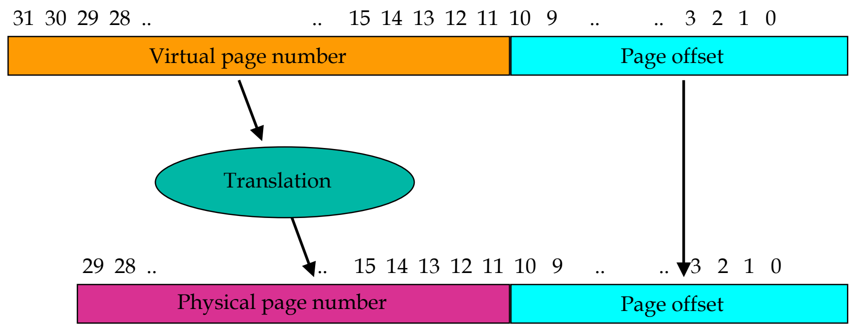
Virtual Memory

- Result is Virtual Memory
- Programs written by human programmers are translated (compiled or interpreted) into machine code + data
- Machine code is executed in a ***virtual address space*** which may provide user with a ***virtual memory*** much larger than real machine, albeit somewhat slower than main memory
- Courtesy of the OS and hardware combined

1B11-3 1999 Slide 10

Mapping Process

- For a 4 GB virtual address space and a 1 GB physical address space



1B11-3 1999 Slide 11

Virtual Memory - 2

- The CPU accesses memory via a memory management unit - this uses a page table to translate addresses the CPU it gives (virtual addresses) into physical addresses.
- The operating system ***("kernel")*** has its own page table that lets it get at everything
- User processes have page tables managed by the operating system
- These page tables ***map*** virtual addresses to physical

1B11-3 1999 Slide 12

Virtual Memory - 3

- So each program is run apparently in the address space 0-max address
- This is mapped in chunks of a page - usually the same size as a disk block
- Mapped to real memory - or flagged if swapped out to disk
- Context switches change the page tables….

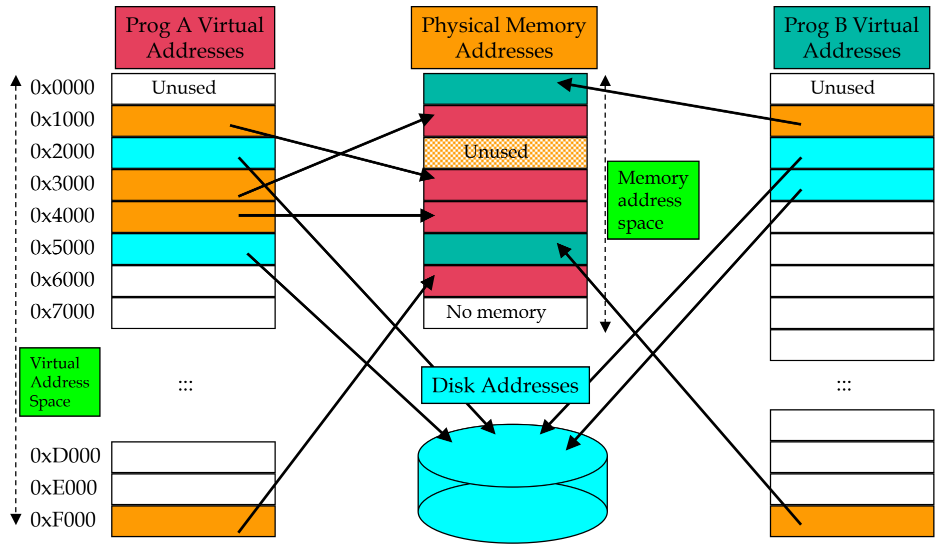
1B11-3 1999 Slide 13

Virtual Memory - 4

- *Q. If you run more programs than will fit in real memory (easy to do) what happens?*
- A. Paging. The most recently run programs are in memory - the oldest one has the blocks (pages) of physical memory written out to disk, and the page table (in memory) changed to say that the pages are on disk - in the ***swap area***
- This of course takes time!
- The page table (kept by the OS) keeps a map of the swap area too

1B11-3 1999 Slide 14

Virtual Address Mapping



1B11-3 1999 Slide 15

Thrashing

- If there are a lot of processes, and they do a lot of I/O, and they are constantly being de-scheduled and re-scheduled, then there *may* be a lot of paging (copying of blocks to and from disk/memory)
- This makes the system *run like a dog* (purely technically speaking, you understand)
- Its called ***thrashing*** - very little progress is made under such circumstances….

1B11-3 1999 Slide 16

Protection

- Operating System provides various degrees of ***protection*** for the user and programs...
  - Safety - Ps should not be able to damage others accidentally or deliberately
  - Fairness - Ps should get fair treatment based on policies in force
  - Integrity - basic assumptions should not be violated
  - Authenticity - ensure objects are what they purport to be
- OS enhances hardware support, to give a smarter virtual service

1B11-3 1999 Slide 17

MMU Protection

- Memory ***maps*** (page tables) provide basic protection against damage, because P can ***only*** "see" real memory that has been mapped to it
- Each page can be marked as:
  - read
  - write
  - execute
  - or a combination

1B11-3 1999 Slide 18

MMU Protection - 2

- For example:
  - code would normally be execute-only to prevent copies being taken or being overwritten accidentally
  - constants, including strings, would be read-only so they cannot be changed
    - Consider x := x+1 if "constant" 1 was changed to 4!!
  - working data and the stack would be read+write

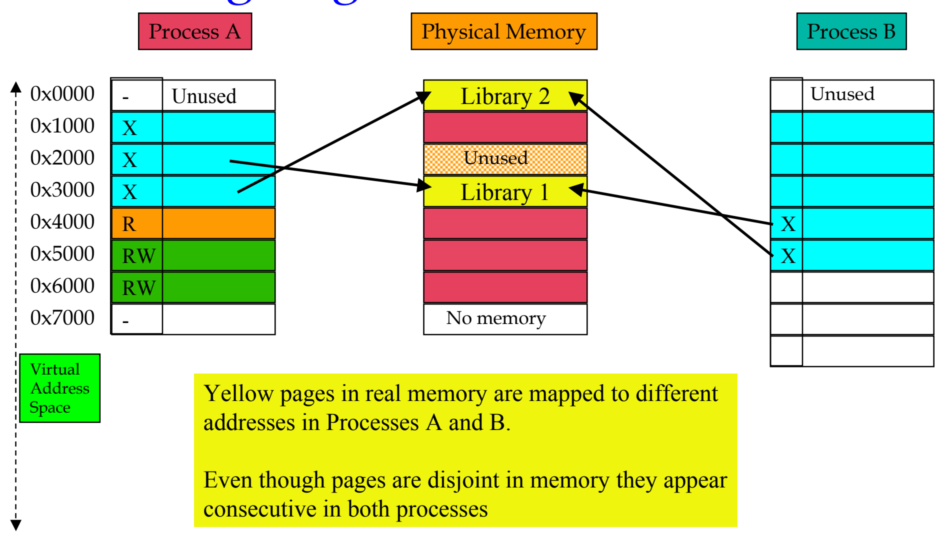
1B11-3 1999 Slide 19

MMU Protection - 3

- Two or more processes can share memory pages (not necessarily at the same address in each P)
  - one might have write access, the other read access - producer-consumer processes e.g.. print server
  - both might have read-write access - application-defined sharing - need to carefully control use (lost updates?)
  - both might have execute access - shared libraries
- Only OS can change the page maps so it controls access

1B11-3 1999 Slide 20

Sharing Pages



Kernel and User Mode

- *Q. How does the OS stop user processes from getting at these data structures (e.g. Page Tables, Process tables etc)?*
- A. The processor can be set to ***kernel mode*** or ***user mode***
- (When the machine is ***booted*** it is set to ***kernel mode*** and memory mapping is set to ***Virtual = Real***)
- The processor mode is changed as a ***side effect*** when the processor executes a ***trap*** or ***interrupt***

Kernel and User Mode - 2

- In kernel mode any instruction can be executed (OS runs in this mode)
- In user mode, (normal processes,) I/O, halt and some other instructions cause a mode change to kernel mode and for a routine at a fixed location to be invoked
- This mechanism is known as a ***trap***

1B11-3 1999 Slide 23

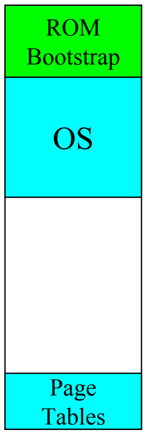
Traps

There are at least 3 types of trap:

- ***System calls*** - Special instructions which can be used to invoke OS functions e.g. create a new process
- ***Exceptions*** - these occur when the processor executes an instruction which makes no sense, e.g.
  - Divide by 0
  - illegal instruction - undefined or not allowed in user mode
  - illegal memory access
- ***Interrupts*** - generated by I/O devices when they are done

1B11-3 1999 Slide 24

Cold Start



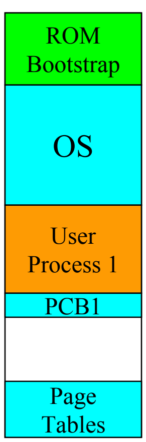
Kernel mode

- Machine starts in kernel mode
- Memory management off - i.e. mapped Virtual = Real
- ROM bootstrap executed to load OS
- Page table set for operating system to "see" whole memory
  - OS R, X, W as appropriate
  - free space R, W
- Memory management switched on

1B11-3 1999 Slide 25

Cold Start - 2

User mode

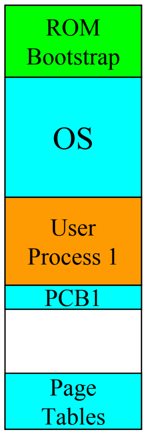


Kernel mode

- Initial user process loaded
  - command processor
  - login process, etc.
- PCB created
- User P's initial PC, SP, page table pre-loaded into PCB
- PCB added to head of *Ready to Run* Q
- Dispatcher invoked

1B11-3 1999 Slide 26

Cold Start - 3

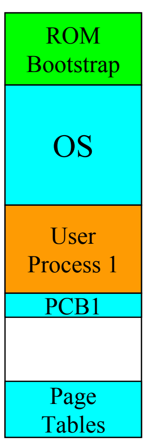


Kernel mode
- Set up user mode page table for P1 from "faked" PCB
- "Return" to P1
  - causes PC to be set
  - causes mode change to ***user mode***

User mode
- P1 executes

1B11-3 1999 Slide 27

Cold Start - 4



User mode
- P1 needs to do I/O - e.g. get user's login name
- ***System call*** for synchronous read

Kernel mode
- Change to kernel mode
- Volatile registers saved in PCB by OS
- OS sets up I/O operation
- P1 status = Blocked
- Idle process (internal in OS) run until I/O complete

- Try extending this for multiple user processes

1B11-3 1999 Slide 28

Summary

- Memory management needed to ensure M used efficiently
- Page tables provide safe mechanism
- Pages can be flagged as read, write or execute or a mixture
- Only OS allowed to change tables
- User mode ensures normal user cannot access I/O devices or act as OS
- Traps provide means of changing mode and communication between user P and OS

1B11-3 1999 Slide 29