CS 370: OPERATING SYSTEMS
[MEMORY MANAGEMENT]

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT
COLORADO STATE UNIVERSITY

2

Frequently asked questions from the previous class survey

- Could you depict the lifecycle of a program through its becoming a process, being subject to scheduling, memory management, etc?
- When swapping processes, is the PCB for that process also put in the swap space?
- Are direct mapping strategies (to physical addresses) performed by the OS?
- Traps?

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MEMORY MANAGEMENT L20.3

3

Logical address spaces in action

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
  printf("location of code : %p\n", main);
  printf("location of heap : %p\n", malloc(100e6));
  int x = 3;
  printf("location of stack: %p\n", &x);
  return x;
}
```

Output when run on a 64-bit Mac

```
location of code : 0x1095afe50
location of heap : 0x1096008c0
location of stack: 0x7fff691aea64
```


Topics covered in this lecture

- Contiguous memory allocations
- Fragmentations
  - External and Internal
- Segmentation
- Paging

Each process is contained in a single continuous section of memory

CONTIGUOUS MEMORY ALLOCATION

6

Partitioning of memory

- Main memory needs to **accommodate** the OS and user processes
- Divided into two partitions
  - Resident OS
    - Usually low memory
  - User processes

7

Memory Mapping and Protection

- Base register (also referred to as a *relocation* register)
  - Smallest physical address
- Limit register
  - Range of logical addresses

8

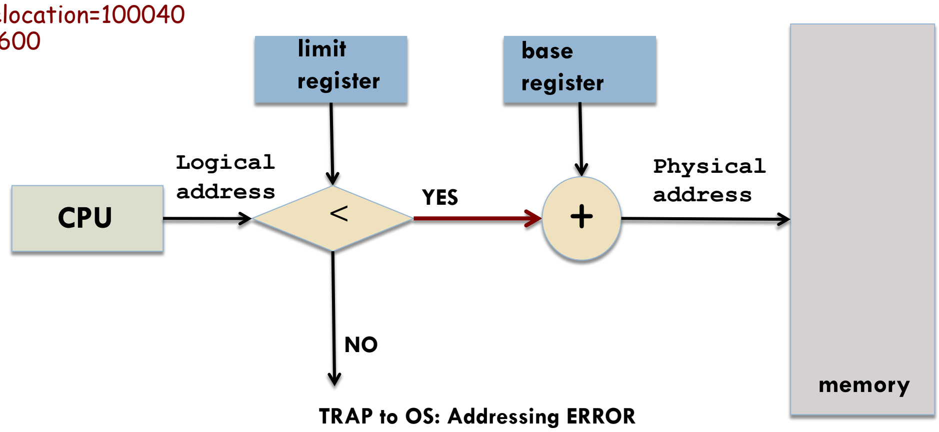
Memory Mapping and Protection

- When CPU scheduler selects a process for execution
  - Base and limit registers reloaded as part of context switch
- Every address generated by the CPU
  - Checked against the relocation(base)/limit registers

9

Memory Mapping and Protection

E.g.: base/relocation=100040
and limit=74600



Memory Allocation: Fixed Partition method

- **Divide** memory into several **fixed-size** partitions
  - Each partition contains exactly one process
- Degree of multiprogramming
  - Bound by the number of partitions

Memory allocation: Variable-partition method [1/2]

- Used in batch environments
- OS maintains table tracking memory utilization
  - What is available?
  - Which ones are occupied?
- Initially all memory is available
  - Considered a large **memory gap**
  - Eventually *many* memory gaps will exist

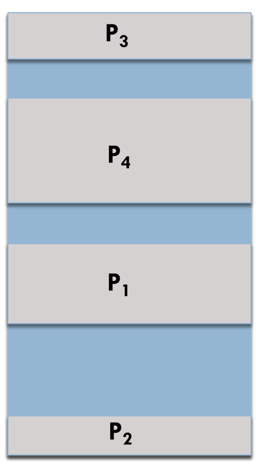
Memory allocation: Variable-partition method [2/2]

- OS orders processes according to the scheduling algorithm
- Memory allocated to processes until requirements of the next process cannot be met
  - *Wait* till a larger block is available
  - Check if smaller requirements of other processes can be met

Variable-partition method: Reclaiming spaces

- When process arrives, if space is too large
  - Split into two
- When process terminates?
  - If released memory is adjacent to other *memory gaps*
    - **Fuse** to form a larger space

Splitting and Fusing Memory spaces



Dynamic Storage Allocation Problem

- Satisfying a request of size ***n*** from the set of available spaces
  - First fit
  - Best fit
  - Worst fit

First fit

- Scan list of segments until you find a memory-gap that is big enough
- Gap is broken up into two pieces
  - One for the process
  - The other is unused memory

Best Fit

- Scan the entire list from beginning to the end
- Pick the smallest memory-gap that is adequate to host the process

Comparing Best Fit and First Fit

- Best fit is **slower** than first fit
- Surprisingly, best fit also results in more **wasted memory** than first fit
  - Tends to fill up memory with tiny, useless gaps

Worst fit

- How about going to the other extreme?
  - Always take the largest available memory-gap
  - Perhaps, the new memory-gap would be useful
- Simulations have shown that worst fit is not a good idea either

SEGMENTATION

Base and limits translation lacks many of the features needed to support modern programs

- Base and limits translation supports only **coarse-grained** protection at the level of the *entire* process
  - It is not possible to prevent a program from overwriting its own code, for example
  - It is also **difficult to share** regions of memory between two processes
  - Since the memory for a process needs to be contiguous …
    - Supporting dynamic memory regions, such as for heaps, thread stacks, or memory mapped files, becomes difficult to impossible

In our discussions so far …

- Logical/virtual memory is **one-dimensional**
  - Logical addresses go from 0 to some `max` value

- Many problems can benefit from having two or more **separate** logical address spaces

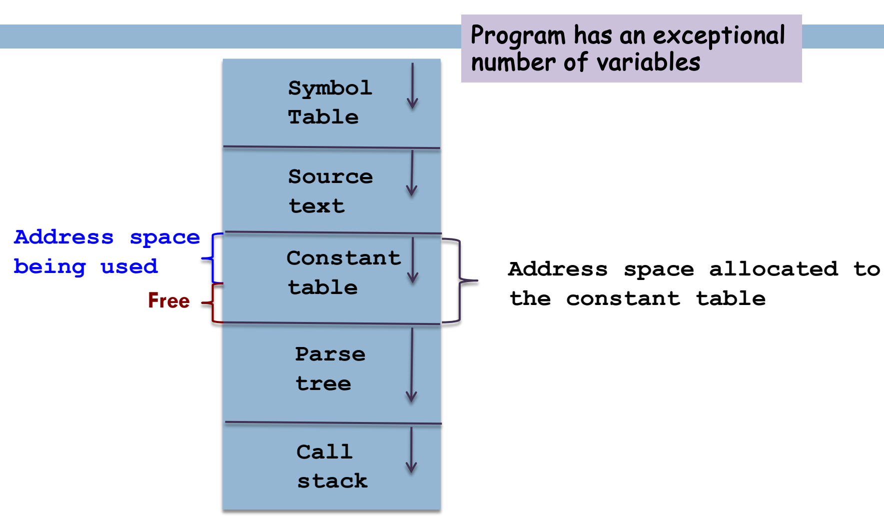
A compiler has many tables that are built up as compilation proceeds

- Source Text
- Symbol table
  - Names and attributes of variables
- Constants Table
  - Integer and floating point constants
- Parse tree
  - Syntactic analysis of program
- Stack
  - Procedure calls within the compiler

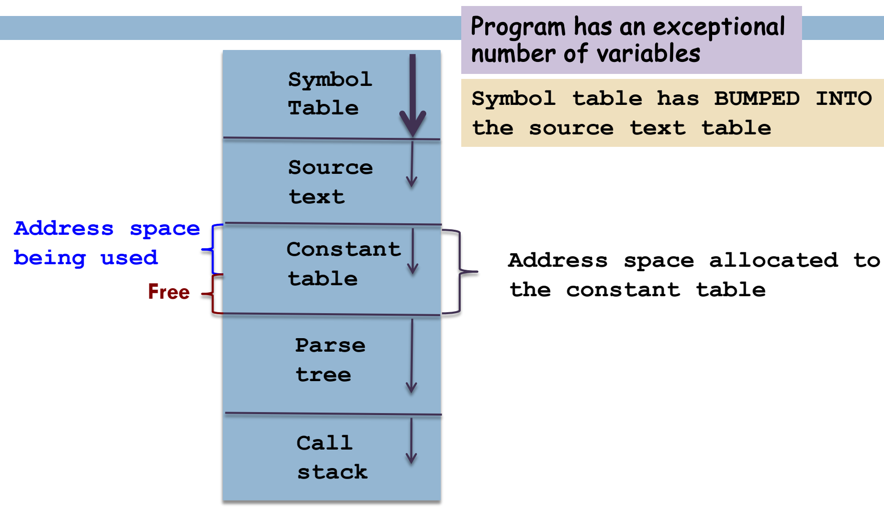
Grows continuously as compilation proceeds (Source Text, Symbol table, Constants Table, Parse tree)

Grows and shrinks in unpredictable ways during compilation (Stack)

One dimensional address space with growing tables



One dimensional address space with growing tables



Options available to the compiler

- Say that compilation cannot continue
  - Not cool
- Play Robin Hood
  - Take space from tables with room
  - Give to tables with little room

What would be really cool …

- Free programmer from having to manage expansion and contraction of tables

But how?

- Provide <u>many</u> completely **independent address spaces**
  - Segments
- Each segment has linear sequence of addresses
  - 0 to max

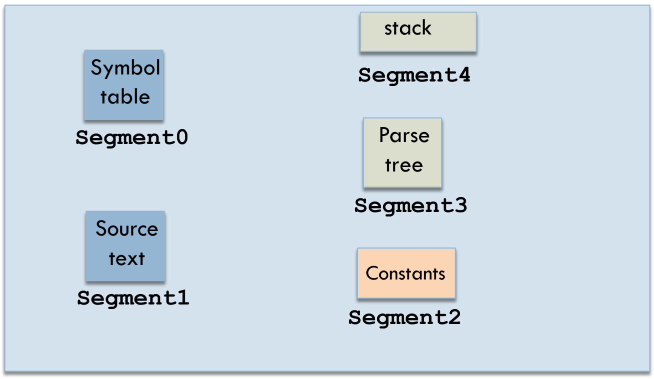
Segments and Base/Limit registers

- The hardware supports an **array** of pairs of base and bounds registers, for each process
  - **Segment Table**
- Each entry in the array controls a portion, or **segment**, of the virtual address space
- The physical memory for **each segment is stored contiguously**, but different segments can be stored at different locations
  - For example, code and data segments are not immediately adjacent to each other in either the virtual or physical address space

Other things about segments

- Different segments can and do have different lengths
- Segments grow and shrink independently without affecting each other; For example, consider a segment for the stack
  - Size increase: something pushed on stack segment
  - Size decrease: something popped off of stack segment

Segmentation allows users to view memory as a collection of variable-sized segments



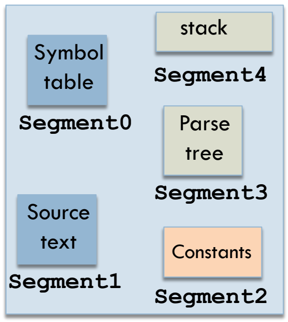
32

Segmentation

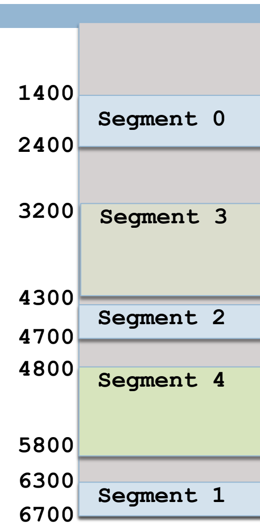
- Logical address space is a collection of segments
- Segments have name and length
- Addresses specify
  - Segment name
  - Offset within the segment
- Tuple: **`<segment-number, offset>`**

33

Segmentation Addressing Example

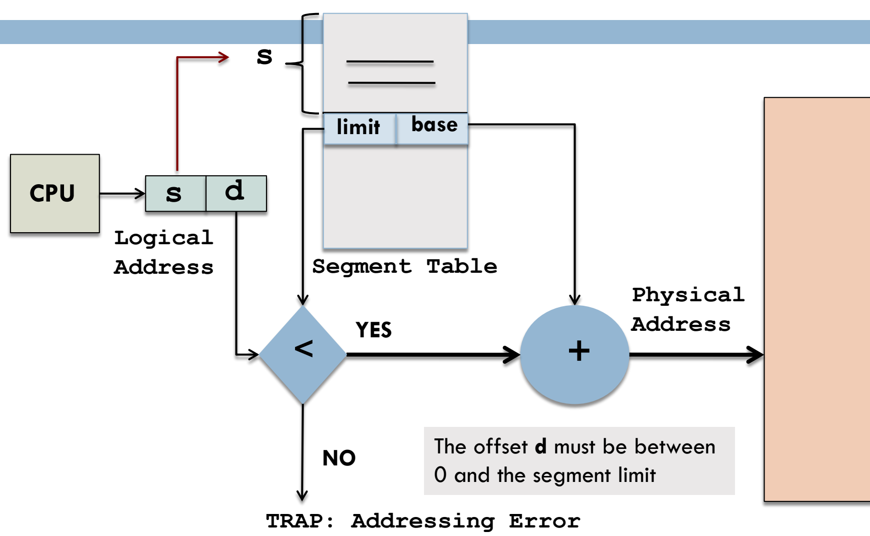


	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1000	3200
4	1000	4800



34

Segmentation Hardware



35

FRAGMENTATION

COMPUTER SCIENCE DEPARTMENT   COLORADO STATE UNIVERSITY

36

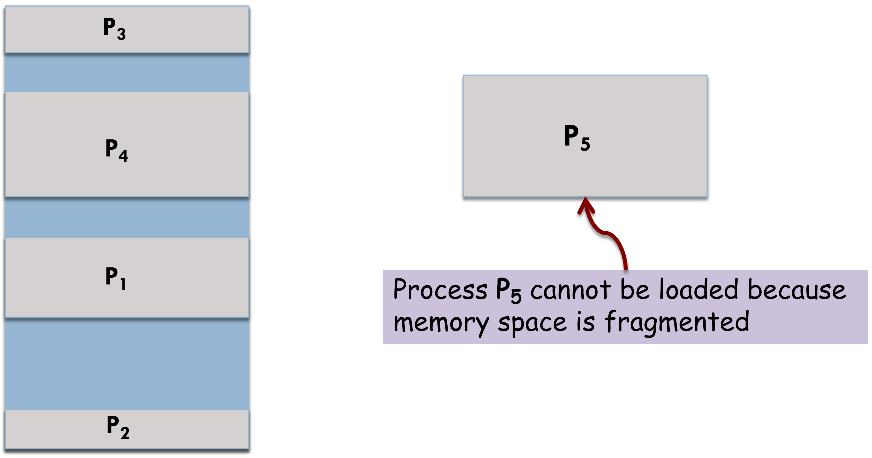
Contiguous Memory Allocation: Fragmentation

- As processes (and segments) are loaded/removed from memory
  - Free memory space is **broken** into small pieces
- **External fragmentation**
  - Enough space to satisfy request; BUT
  - Available spaces are *not contiguous*

COLORADO STATE UNIVERSITY   Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT   MEMORY MANAGEMENT   L20.37

37

Fragmentation: Example



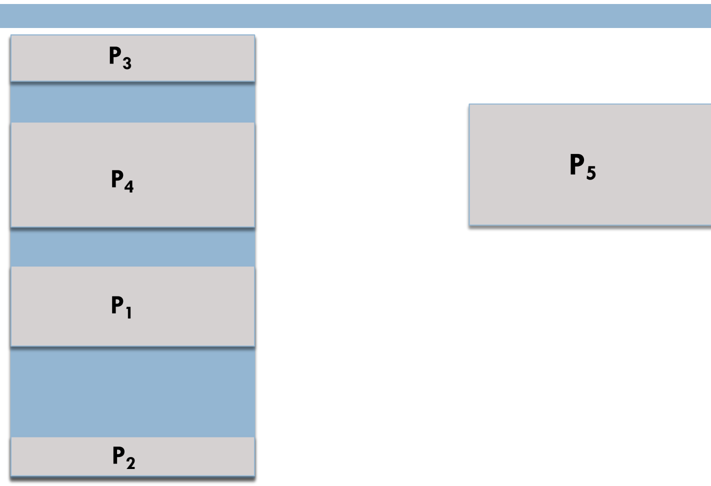
Fragmentation can be internal as well

- Memory allocated to process may be *slightly larger* than requested
- **Internal fragmentation**
  - Unused memory is internal to blocks

Compaction: Solution to external fragmentation

- **Shuffle** memory contents
  - Objective: Place free memory into large block
- Not possible if relocation is static
  - Load time
- Approach involves moving:
  1. Processes towards one end
  2. Gaps towards the other end

Compaction: Example

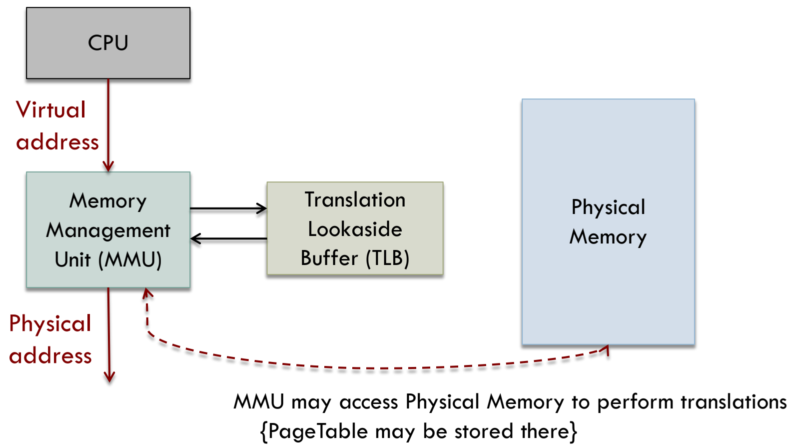


Memory compaction is time intensive and is usually not done

- Let's consider a machine with 1 GB of RAM
- The machine can copy 4 bytes in 20 nanoseconds
- Time to compact all the memory?
  $10^9 \times (20 \times 10^{-9}/4) = 5$ seconds (approximately)
  Note: 1 GB is approximately 10^9 bytes

PAGING:
OVERVIEW OF THE MAPPING PROCESS

Overview of how mapping of logical and physical addresses is performed



44

PAGING

Noncontiguous memory management

45

The Paging memory management scheme

- Physical address space of process can be **non-contiguous**
- Solves problem of fitting variable-sized memory chunks to backing store
  - Backing store has fragmentation problem
    - Compaction is impossible

Basic method for implementing paging

- Break memory into **fixed-sized** blocks
  - Physical memory: **frames**
  - Logical memory: **pages**

  } **Same size**
- Backing store is also divided the same way

What will seem odd, and perhaps cool, about paging [1/2]

- While a program **thinks of its memory as linear** …
  - It is usually **scattered** throughout physical memory in a kind of abstract mosaic
- The processor will execute one instruction after another using virtual addresses
  - The virtual addresses are still linear
  - However, an instruction located at the end of a page will be located in a **completely different region** of physical memory from the next instruction at start of another page

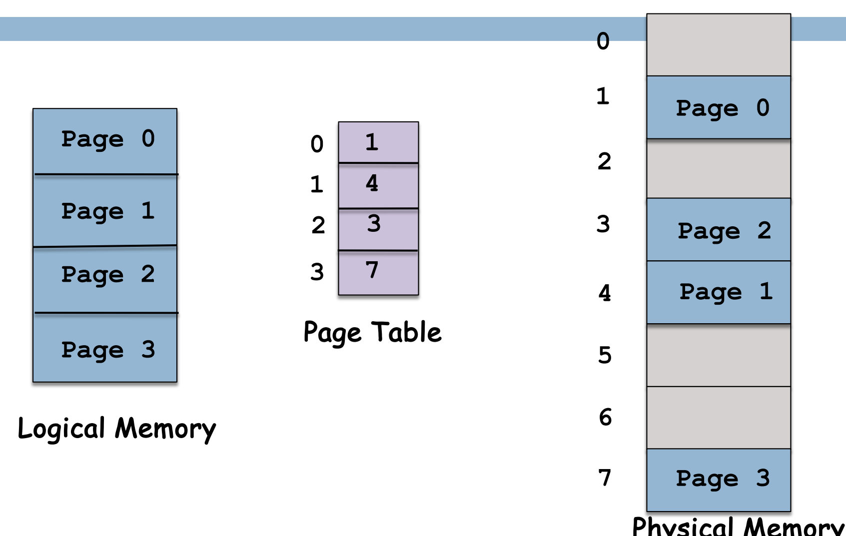
What will seem odd, and perhaps cool, about paging [2/2]

- Data structures appear to be contiguous using virtual addresses
  - But a large matrix is scattered across many physical page frames

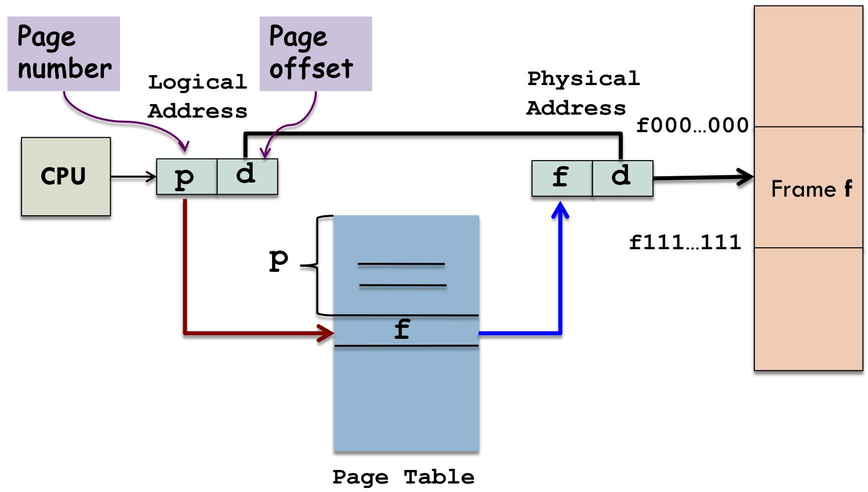
Paging: Analogy

- Shuffling several decks of cards together
- A single process in its virtual address page sees the cards of a single deck in order
  - A different process sees a completely different deck, but it will also be in order
- In physical memory, however, the **decks of all processes** currently running will be **shuffled** together, apparently at random
- **Page tables are the magician's assistant** in locating cards from the shuffled decks

Paging: Logical and Physical Memory



Paging Hardware: Performing address translation



The contents of this slide-set are based on the following references

- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 8]
- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 3]*
- *Thomas Anderson and Michael Dahlin. Operating Systems Principles and Practice. 2nd Edition. Recursive Books. ISBN: 978-0985673529.* [Chapter 8]
- *Remzi Arpaci-Dusseau and Andrea Arpaci-Dusseau. Operating Systems: Three Easy Pieces. 1st edition. CreateSpace Independent Publishing Platform. ISBN-13: 978-1985086593. [Chapter 14]*