Big picture

- Fixed partitions
- Dynamic partitions
- Buddy system

Contiguous allocation:
Each process occupies a contiguous memory region in the physical memory.

Segmentation

Paging

Non-contiguous allocation:
Each process comprises multiple memory regions scattered in the physical memory.
Does the “fixed partitions” scheme cause fragmentation?

Zero external fragmentation
Internal fragmentation can be severe

Analogy: street parking with meters
Does the “dynamic partitions” scheme cause fragmentation?

- Zero internal fragmentation
- External fragmentation can be severe

Analogy: street parking without meters
Working Set

- Working Set: the part of memory that a process references in a given time interval
  - You can roughly understand it as “the memory regions that are currently used”
- Usually a very small portion of the entire memory is requested by a process
- **Analogy**: the seat you currently use is your “working set”, while during your study at Temple you need much more space: library, dining, lab, classroom seat, etc.
Limitations of contiguous allocation

• Does contiguous allocation exploit the small working set?
  – No, as a contiguous memory block is allocated to meet the maximum need of a process; it means that a process cannot run unless its maximum need is met
  – But actually only a small portion of its maximum need is really accessed in a given time interval
Working Set - example

• With contiguous allocation, N processes reside in memory. Assume the memory requested by each process is equal and the working set is \( \frac{1}{4} \) of the requested memory.

• At most, how many active processes can be serviced by the main memory in theory if only the working set of each process resides in memory?
  – \( 4N \)
Swapping

• When the free memory runs low, swap area in the disk is used
• All or part of a process’s data is swapped temporarily out of memory to the swap area, and then brought back into memory for continued execution
• Swapping is found on many systems (i.e., UNIX, Linux, and Windows)
Segmentation

- A program is divided into segments
- A segment can be a procedure, a stack, a global array, etc.

- Only segments that correspond to the current working set reside in the main memory.
- Others are put at disk, and can be swapped into main memory when needed
Disadvantage of Segmentation

• When a process exits, its segments leave “holes” of varying sizes in main memory
  – Similar to dynamic partitions, external fragmentation is still severe
  – E.g., assuming two 1M holes have been created, they cannot be used to satisfy a segment request for 2M
• Inefficient handling of growing segments, such as heap and stack
  – Reserving a large memory space leads to severe internal fragmentation, while reserving a small space will result in repetitive reallocation when it grows
Paging

• Main memory is divided into equal fixed-size chunks, called *page frames*, that are relatively small

• A process is divided into small fixed-size chunks, called *pages*, of the same size
  – A Page refers to a chunk of address space

• The pages of a process can be stored in separated page frames in main memory

• *Any page can be put at any page frame*
Logical view – virtual address space

• The logical view: each process has a huge contiguous virtual address space
  – 32-bit system: $2^{32}$
  – 64-bit system: $2^{64}$ (so far, only $2^{48}$ is used)

• This largely simplifies the compiler, which assumes a uniform huge address space, regardless of the allocation in physical memory
Anatomy of the virtual address space of a process

1GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault

0xc0000000 == TASK_SIZE
Random stack offset

Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset

Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so

3GB
program break
brk

Heap
start_brk
Random brk offset

BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;

end_data

Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = “God’s own prototype”;

start_data
end_code

Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)

0x08048000
Paging Example

Internal fragmentation?
- Yes, but it only occurs for the last page when the requested size is not a multiple of pages. E.g., a process that requests 3.1 pages of space will get 4 pages.

External fragmentation?
- No, any “holes”, i.e., page frames, left by the exited process can be reused happily.
Understanding the output of `pmap`

Please try "pmap –XX [pid]" after class

The source of the mapping – either a file, an anonymous mapping or a special mapping such as [heap], [stack], [video], [vmapcall]

<table>
<thead>
<tr>
<th>Address</th>
<th>Perm</th>
<th>Direct Device</th>
<th>inode Size Mode</th>
<th>Proc</th>
<th>Clean</th>
<th>Dirty</th>
<th>Shared</th>
<th>Private</th>
<th>Private</th>
<th>Referenced</th>
<th>Anonymous</th>
<th>AnonObjectPages</th>
<th>Swap</th>
<th>Size</th>
<th>Size Locked</th>
<th>Kernel MMU House Pages</th>
<th>VFS Flags</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
<tr>
<td>00000000</td>
<td>r-x</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

The permissions which apply to this mapping:
- r – Read, the process is allowed to read the memory contents
- w – Write, the process is allowed to write to the memory area
- x – Execute, the process is allowed to execute instructions from the memory area
- p – Mapping is private (or sharable – “p”)

This refers to write operations into this memory mapping – if the “p” flag is set, writes to this memory area (if allowed at all by the other access permissions) will not be visible to other processes generally causing the respective page to be copied if it was shared with other processes – “copy on write”

Additional information for the example mapping:
- Device end major file system inode number of the file to which the mapping refers and the offset of the mapping within the file
- Size of mapped address range in virtual address space
- Resident Size
- Amount of memory which is currently in RAM (not swapped) in MB
- Proportional Share Size
- Size of shared size divided by number of mappings in MB
- Amount of memory which is shared with other processes. Note: if memory which can be shared is mapped only once, this memory is counted as private. Once it is mapped by at least one other process, it will be counted as shared. "Dirty" refers to pages which have been modified since the mapping was created.
- Amount of memory which is private to this process. "Dirty" refers to pages which have been modified since the mapping was created.
- Amount of memory that does not belong to a file. Note: that even file based mappings can contain non-virtual pages in the case of "copy on write"
- Page sizes (in KB) used in this mapping
- Indicates whether the mapping is locked or not. Locked memory is prevented from being swapped out. See below (3)

Kernel flag for the memory area – see the "maps" section at https://www.kernel.org/doc/man-pages/vmscan.9 for more details.

The starting address of this mapping within the virtual linear address space of the process.

The source of the mapping – either a file, an anonymous mapping or a special mapping such as [heap], [stack], [video], [vmapcall]
Questions

• Where do global and static variables reside during execution?
  – It depends
  – **BSS** if you declare them w/o init values
    • System zeros page content automatically, so you can expect they are all zeros
  – **Data** otherwise
• Why cannot I declare a big array (>8M) in my function?
  – Stack limit
  – You should use malloc to allocate from the heap
• Why do I encounter segmentation fault exceptions?
  – Many possibilities
  – Refer to some unallocated holes
  – Refer to some heap buffers that have been freed
  – Access protected areas, such as kernel space; write to read-only
  – How to debug: Use “bt” (backtrace) of gdb to debug
Summary

• Fixed partitions
• Dynamic partitions
• Buddy system
  – Split-based allocation
  – Coalescing-buddy-based deallocation
  – Freelists-based implementation

• Segmentation
• Paging

• Something you can use in your future design and coding
  – Free list
Writing Assignment

• What is internal fragmentation? What is external fragmentation?
• If the swapping mechanism is not used, do the schemes of segmentation/paging still have advantages over contiguous memory allocation?
• What will happen if the RAM is less than the total size of the working sets of the processes?
Extra stuff: Swapping on Mobile Systems

• Not typically supported
  – Flash memory based
    • Small amount of space
    • Limited number of write cycles
    • Poor throughput between flash memory and CPU on mobile platform

• Instead use other methods to free memory if low
  – iOS *asks* apps to voluntarily relinquish allocated memory
    • Read-only data thrown out and reloaded from flash if needed
    • Failure to free can result in termination
  – Android terminates apps if low free memory, but first writes application state to flash for fast restart
A Little Bit Strange Comparison

• CPU time allocation strategies
  – Allocate the whole CPU to a process until the process finishes: FCFS, Shortest-job first
  – The CPU time is divided into slices and allocated: Round-robin, MFQ
  – The second category is for better multitasking

• Memory allocation strategies
  – The memory need of a process is allocated as a whole: fixed partitions, dynamic partitions, Buddy
  – The memory need of a process is divided into segments or pages: segmentation, paging
  – The second category is to accommodate more processes in the memory