Previous class...

- Uniprocessor policies
  - FCFS, Shortest Job First
  - Round Robin
  - Multilevel Feedback Queue

- Multiprocessor policies
  - A MFQ per processor
  - A process has affinity with a processor
  - Exception: idle cores can steal processes
Background

• Main memory and registers are only storage CPU can access directly
• Program must be brought (from disk) into memory for it to be run
Outline

- Fixed partitions
- Dynamic partitions
- Buddy system
- Segmentation
- Paging

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

Non-contiguous allocation:
Each process occupies multiple memory regions scattered in the physical memory.
Fixed partitions

• The bounds of each partition are fixed/predefined

• Disadvantages:
  – Cause Internal Fragmentation when the allocated space is larger than the need
    • What is “fragmentation”? Small useless chunks
    • E.g., when you put a 13M process in the 16M partition, 3M space is wasted and it is called internal fragmentation
      • Internal: the wasted space is inside allocated space
  – The number of active processes is limited

• Analogy: street parking with meters
Fixed partitions - questions

• If 8M partitions are all used, where do you place a 7M process, and what is the size of the internal fragmentation?
  – The 12M partition is the best choice
  – The internal fragmentation is 5M

• How to resolve the severe internal fragmentation?
  – Dynamic partitions
Dynamic partitions

- Process is allocated exactly the memory it requires
- The partitions are dynamic: the number and locations of partitions are not fixed
- Analogy: street parking without meters
Dynamic partitions - example

(a) 8M
(b) 20M
(c) 20M
(d) 20M

(e) 20M
(f) 20M
(g) 20M
(h) 14M
Dynamic partition - disadvantage

• External fragmentation
  – Their total size is large enough to satisfy a request, but they are not contiguous, so cannot be used to service the request
  – *External*: fragmentation is outside allocated space

• Solution: Compaction
  – OS shifts processes so that they are contiguous; thus, free memory is together in one block
  – Program execution must be paused for relocation; waste CPU time
Placement Algorithms

• When there is more than one free block of memory of sufficient size, the system must decide which free block to allocate.

<table>
<thead>
<tr>
<th>Best-fit</th>
<th>First-fit</th>
<th>Next-fit</th>
</tr>
</thead>
<tbody>
<tr>
<td>• chooses the block that is</td>
<td>• scan memory from the beginning and chooses the</td>
<td>• scan memory from the location of the last</td>
</tr>
<tr>
<td>closest in size to the request</td>
<td>first available block that is large enough</td>
<td>placement and chooses the next available block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>that is large enough</td>
</tr>
</tbody>
</table>

When there is more than one free block of memory of sufficient size, the system must decide which free block to allocate.
Assume a 16M partition is requested.
Questions

• Does Fixed Partitions Allocation has external fragmentation?
  – Zero
  – There are no small useless chunks outside a partition

• Does Dynamic Partitions Allocation has internal fragmentation?
  – Zero
  – The allocated partition size is as needed
Buddy System (or, Buddy Memory Management)

• Fixed partitions cause severe internal but zero external fragmentation, while dynamic partitions cause severe external but zero internal fragmentation

• Buddy System is between the two
  – It causes acceptable internal and external fragmentation, and has good overall performance

• Three properties
  – Split-based allocation
  – Freelists-based implementation
  – Coalescing-buddy-based deallocation
Buddy System – allocate the first block

- To begin, the entire space available for allocation is treated as a single block of size $2^U$
- The request size is first rounded up as power of 2, denoted as $S$
  - E.g., $55 \rightarrow 2^6=64$; $120 \rightarrow 2^7=128$
- If $S = 2^U$, then the entire block is allocated. Otherwise, the block is split into two equal buddies of size $2^{U-1}$
- If $S = 2^{U-1}$, then the request is allocated to one of the two buddies. Otherwise, one of the buddies is split into halves again. This process continues until the split block is equal to $S$ and it is allocated to the request
Buddy System – split-based allocation

<table>
<thead>
<tr>
<th>Request</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 K</td>
<td>128K</td>
<td>256K</td>
<td>512K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>240 K</td>
<td>128K</td>
<td>256K</td>
<td>512K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 K</td>
<td>128K</td>
<td>256K</td>
<td>512K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256 K</td>
<td>128K</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release B</td>
<td>128K</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release A</td>
<td>128K</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75 K</td>
<td>128K</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release C</td>
<td>128K</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release E</td>
<td>512K</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Release D</td>
<td>1M</td>
<td>256K</td>
<td>256K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Binary Tree Representation

- **Buddies**: sibling nodes, i.e., two nodes that share the parent node in the binary tree representation.
Buddy System – free-lists-based implementation

A free list is a linked list data structure used to connect all unallocated storage chunks. It is quick and easy to allocate and unallocate a storage chunk.

<table>
<thead>
<tr>
<th>Order</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>(2^0 \times 64k = 64k)</td>
</tr>
<tr>
<td>1:</td>
<td>(2^1 \times 64k = 128k)</td>
</tr>
<tr>
<td>2:</td>
<td>(2^2 \times 64k = 256k)</td>
</tr>
<tr>
<td>3:</td>
<td>(2^3 \times 64k = 512k)</td>
</tr>
<tr>
<td>4:</td>
<td>(2^4 \times 64k = 1024k)</td>
</tr>
</tbody>
</table>

One \(2^2\) block is allocated.
Buddy System used in Linux

• The Buddy System is used in Linux (and many other OSes) to manage the physical memory
Buddy System – coalescing-buddy-based deallocation

- Whenever a block is freed, it tries to coalesce with its buddy. The coalescing procedure is recursive
Buddy System - fragmentation

• Internal fragmentation?
  – Worst case ~ 50%
  – E.g., request 128.01K, which is rounded up to 128x2K

• External fragmentation
  – Still exists, as coalescing can only occur between buddies
  – Best-fit is always used: freelists-based implementation allows you to find the best-fit chunk quickly
  – By applying restriction on splitting, you will not see small mini tiny useless holes; e.g., previously we limit the smallest block as 64k
Working Set

• Working Set: the amount 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 ¼ 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
Big picture

- Fixed partitions
- Dynamic partitions
- Buddy system

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

Non-contiguous allocation:
Each process comprises multiple memory regions scattered in the physical memory.
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)
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
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 serious

• 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.
  - Page can also refer 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.*
Paging

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.
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

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
0
Understanding the output of `pmap`

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

• Where do global and static variable reside?
  – 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

• 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
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
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?
• What is the worst case of internal fragmentation for a memory allocation in the Buddy System?
• 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?