CIS 3207 - Operating Systems

Non-contiguous Memory Allocation

Professor Qiang Zeng

Spring 2018

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

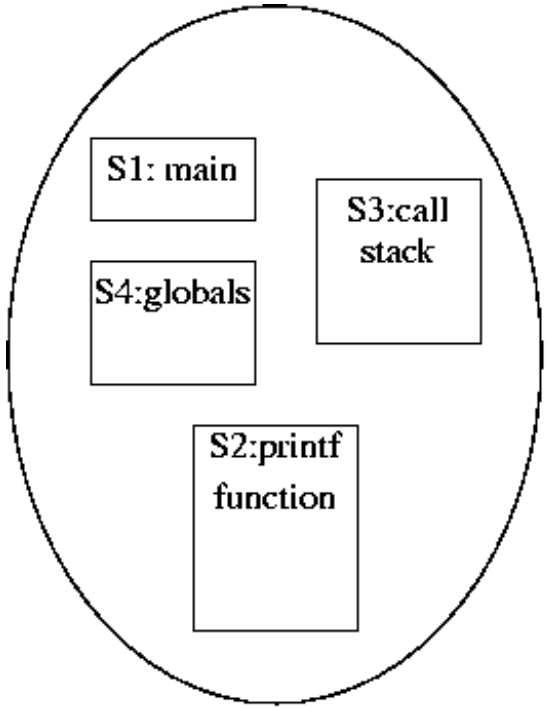
- With contiguous allocation, N processes reside in memory. Assume the memory requested by each process is equal and the working set is ¼ of the requested memory
- At most, how many active processes can be serviced by the main memory in theory if only the working set of each process resides in memory?
  - 4N

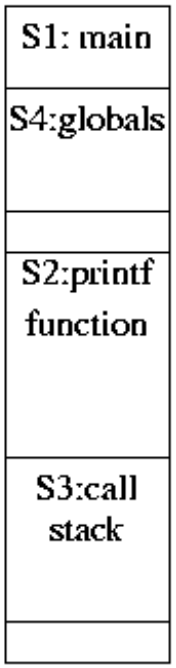
Swapping

- 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

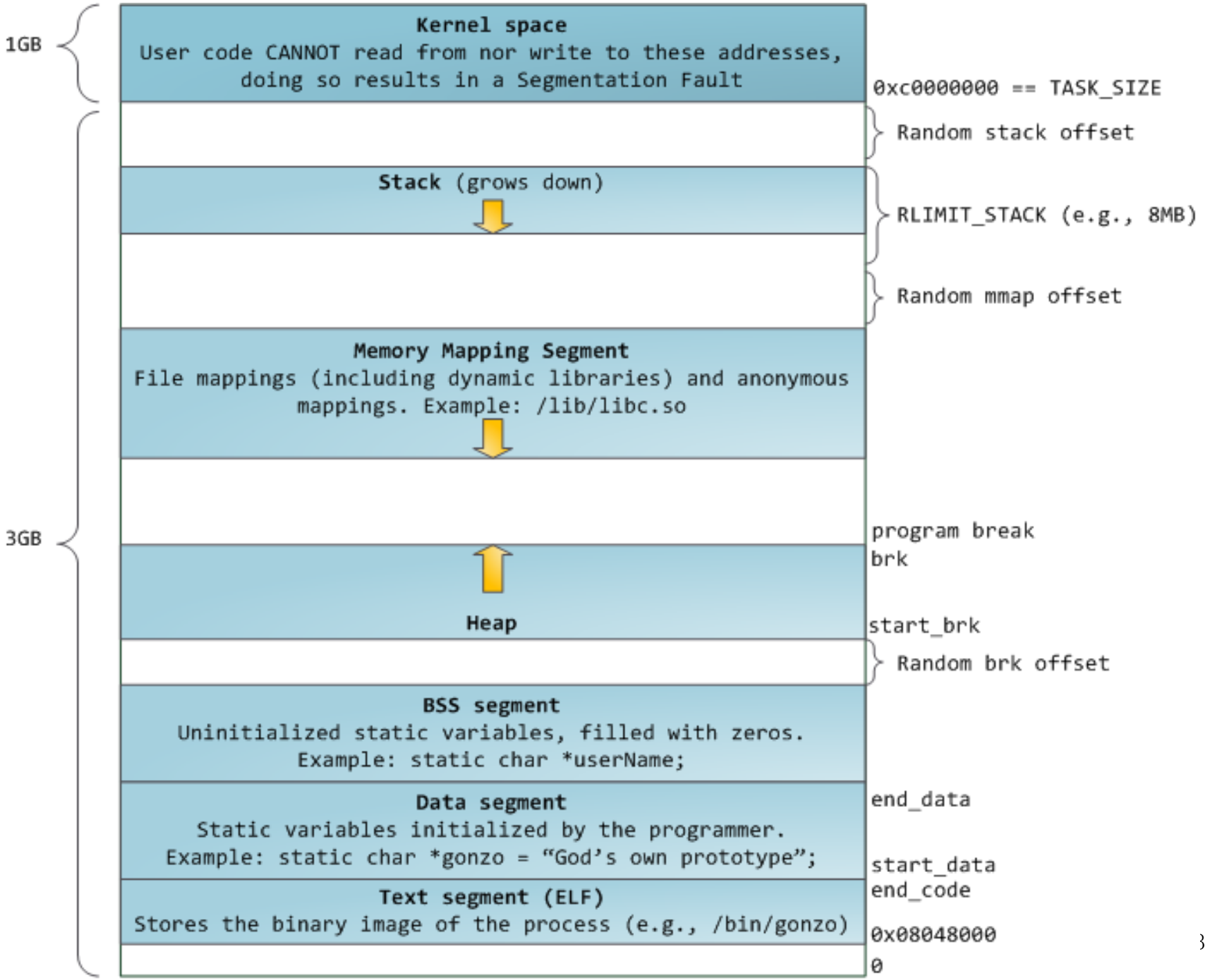
- When a process exits, its segments leave "holes" of varying sizes in main memory
  - Similar to dynamic partitions, external fragmentation is still severe
  - E.g., assuming two 1M holes have been created, they cannot be used to satisfy a segment request for 2M
- Inefficient handling of growing segments, 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

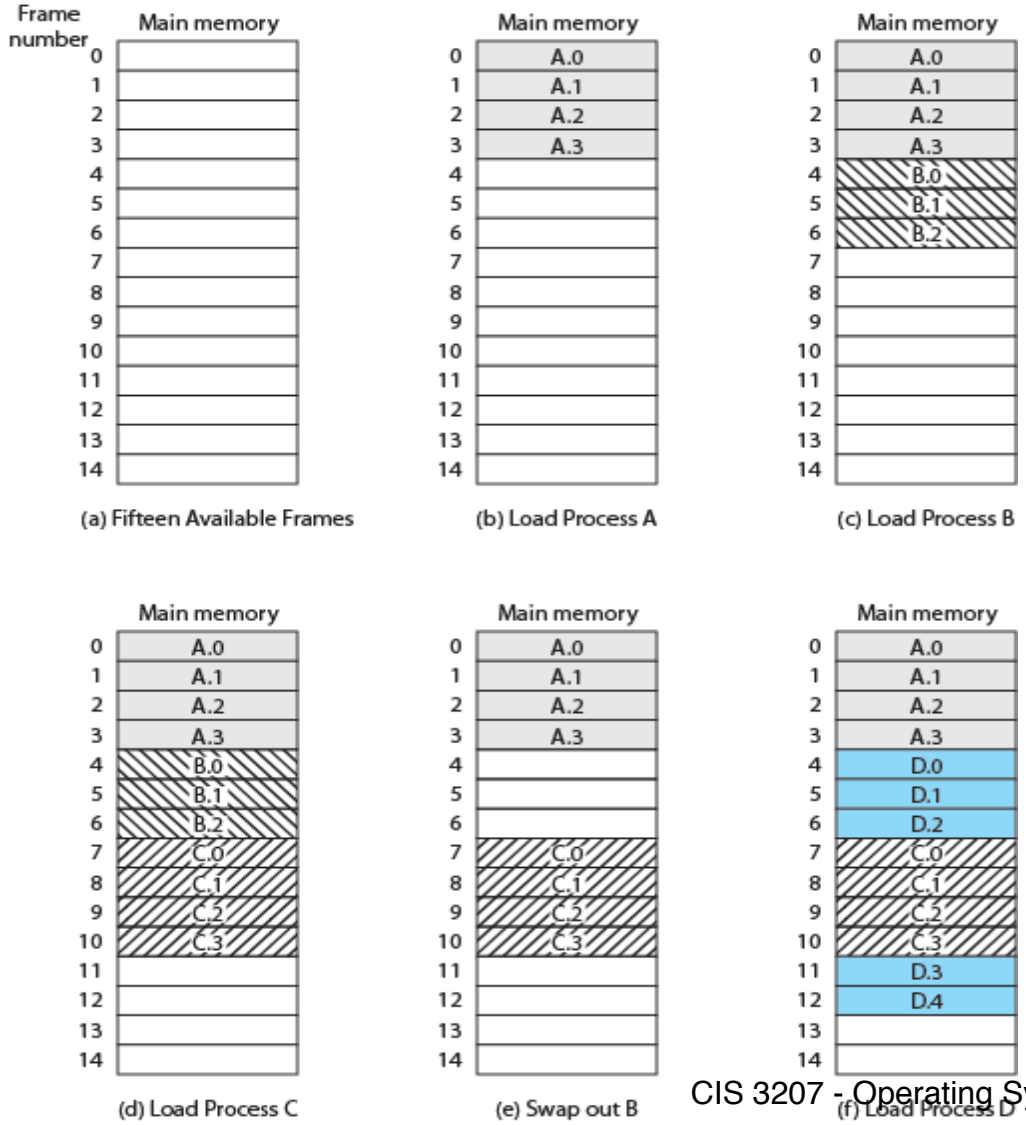
- Main memory is divided into equal fixed-size chunks, called *page frames*, that are relatively small
- A process is divided into small fixed-size chunks, called *pages*, of the same size
  - A Page refers to a chunk of address space
- The pages of a process can be stored in separated page frames in 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


1GB
3GB
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
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;
Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = "God's own prototype";
end_data
start_data
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
end_code
0x08048000
0

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

- **Address** – The starting address of this mapping within the virtual linear address space of the process
- **Perm** – 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 into the memory area
  - **x** – Execute, the process is allowed to execute instructions from this memory area
  - **p** – Mapping is private (or shared – „s“). 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 (probably causing the respective page to be copied if it was shared with other processes – „copy on write“)
- **Offset / Device / Inode** – Device minor/major number, file system inode number of the file to which the mapping refers and the offset of the mapping within the file (in case of file based mappings)
- **Size** – Size of mapped address range in virtual address space (in KiB)
- **Rss** – Resident Set Size. Amount of memory which is currently in RAM (not swapped) in KiB
- **Pss** – Proportional Share Size. Private size plus (shared size divided by number of mappings) in KiB
- **Shared Clean / Shared Dirty** – 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.
- **Private Clean / Private Dirty** – Amount of memory which is private to this process. „Dirty“ refers to pages which have been modified since the mapping was created.
- **Anonymous** – Amount of memory that does not belong to a file. Note that even file based mappings can contain anonymous pages in the case of „copy-on-write“
- **Swap** – Amount of anonymous memory which is swapped out – remember that file based read-only mappings (e.g. Code) does not need to be swapped out since it can be re-read from the file!
- **Kernel Page Size / MMU Page Size** – Page sizes (in KiB) used in this mapping
- **Locked** – Indicates whether the mapping is locked or not. Locked memory is prevented from being swapped out. See `mlock(2)`.
- **VmFlags** – Kernel flags for this memory area – see the „smaps“ section at https://www.kernel.org/doc/Documentation/filesystems/proc.txt for more details
- **Mapping** – The source of the mapping – either a file, an anonymous mapping or a special mapping such as `[heap]`, `[stack]`, `[vdso]`, `[vsyscall]`

```
$ pmap -XX 13305
13305:   ./minimal
                                                          Shared  Shared Private Private                                                    Kernel MMU
                                                                                                                                            Page   Page
         Address Perm   Offset Device   Inode Size Rss Pss Clean   Dirty   Clean   Dirty Referenced Anonymous AnonHugePages Swap Size   Size Locked                VmFlags Mapping
        00400000 r-xp 00000000  08:01 3948320    4   4   1     4       0       0       0          4         0             0    0    4      4      0    rd ex mr mw me dw sd minimal
        00600000 r--p 00000000  08:01 3948320    4   4   4     0       0       0       4          4         4             0    0    4      4      0    rd mr mw me dw ac sd minimal
        00601000 rw-p 00001000  08:01 3948320    4   4   4     0       0       0       4          4         4             0    0    4      4      0 rd wr mr mw me dw ac sd minimal
    7fd7c5dd3000 r-xp 00000000  08:01 1587252 1772 220   2   220       0       0       0        220         0             0    0    4      4      0       rd ex mr mw me sd libc-2.19.so
    7fd7c5f8e000 ---p 001bb000  08:01 1587252 2044   0   0     0       0       0       0          0         0             0    0    4      4      0             mr mw me sd libc-2.19.so
    7fd7c618d000 r--p 001ba000  08:01 1587252   16  16  16     0       0       0      16         16        16             0    0    4      4      0       rd mr mw me ac sd libc-2.19.so
    7fd7c6191000 rw-p 001be000  08:01 1587252    8   8   8     0       0       0       8          8         8             0    0    4      4      0    rd wr mr mw me ac sd libc-2.19.so
    7fd7c6193000 rw-p 00000000  00:00       0   20  12  12     0       0       0      12         12        12             0    0    4      4      0    rd wr mr mw me ac sd
    7fd7c6198000 r-xp 00000000  08:01 1587249  140 112   1   112       0       0       0        112         0             0    0    4      4      0    rd ex mr mw me dw sd ld-2.19.so
    7fd7c6390000 rw-p 00000000  00:00       0   12  12  12     0       0       0      12         12        12             0    0    4      4      0    rd wr mr mw me ac sd
    7fd7c63b7000 rw-p 00000000  00:00       0   12   8   8     0       0       0       8          8         8             0    0    4      4      0    rd wr mr mw me ac sd
    7fd7c63ba000 r--p 00022000  08:01 1587249    4   4   4     0       0       0       4          4         4             0    0    4      4      0    rd mr mw me dw ac sd ld-2.19.so
    7fd7c63bb000 rw-p 00023000  08:01 1587249    4   4   4     0       0       0       4          4         4             0    0    4      4      0 rd wr mr mw me dw ac sd ld-2.19.so
    7fd7c63bc000 rw-p 00000000  00:00       0    4   4   4     0       0       0       4          4         4             0    0    4      4      0    rd wr mr mw me ac sd
    7fff927e8000 rw-p 00000000  00:00       0  136  12  12     0       0       0      12         12        12             0    0    4      4      0    rd wr mr mw me gd ac [stack]
    7fff9292c000 r-xp 00000000  00:00       0    8   4   0     4       0       0       0          4         0             0    0    4      4      0    rd ex mr mw me de sd [vdso]
ffffffffff600000 r-xp 00000000  00:00       0    4   0   0     0       0       0       0          0         0             0    0    4      4      0                   rd ex [vsyscall]
                                             ==== === === ===== ======= ======= ======== ========== ========= ============= ==== ==== ====== ======
                                             4196 428  92   340       0       0       88        428        88             0    0   68     68      0 KB
```

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?
- 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?