CMPUT 379 Lab

ETLC E1003: Tuesday, 5:00 – 7:50 PM.

Tianyu Zhang, Peiran Yao

CAB 311: Thursday, 2:00 – 4:50 PM.

Max Ellis, Aidan Bush

Today’s Lab

- Examples on memory management
- Feedback on the midterm exam

Memory management examples

- Download the examples from eClass
- Play!

Memory managements

- Relocation (Translation)
  - translate virtual address (**VA**) to physical address (**PA**)
- Protection
- Options
  - Base and bounds
  - Segmentation
  - Paging
  - Multi-level translation

Dynamic relocation with base and bounds

- Each process has 2 registers: **base** and **limit**
- Translation: **PA = VA + base**
- Protection: **VA < limit**

Dynamic relocation - simulator

- Go to **HW-relocation/** folder
- Run **python relocation.py** to generate a base and bound pair and VAs

```
prompt> ./relocation.py
...
Base-and-Bounds register information:

  Base   : 0x00003082 (decimal 12418)
  Limit  : 472

Virtual Address Trace
  VA  0: 0x01ae (decimal:430) -> PA or violation?
  VA  1: 0x0109 (decimal:265) -> PA or violation?
  VA  2: 0x020b (decimal:523) -> PA or violation?
  VA  3: 0x019e (decimal:414) -> PA or violation?
  VA  4: 0x0322 (decimal:802) -> PA or violation?
```

```
--------------- 0KB
|    Code     |
--------------- 2KB
|   Stack     |
--------------- 4KB
|    Heap     |
|     |       |
|     v       |
--------------- 7KB
|   (free)    |
|     ...     |
```

Virtual memory layout

Dynamic relocation - simulator

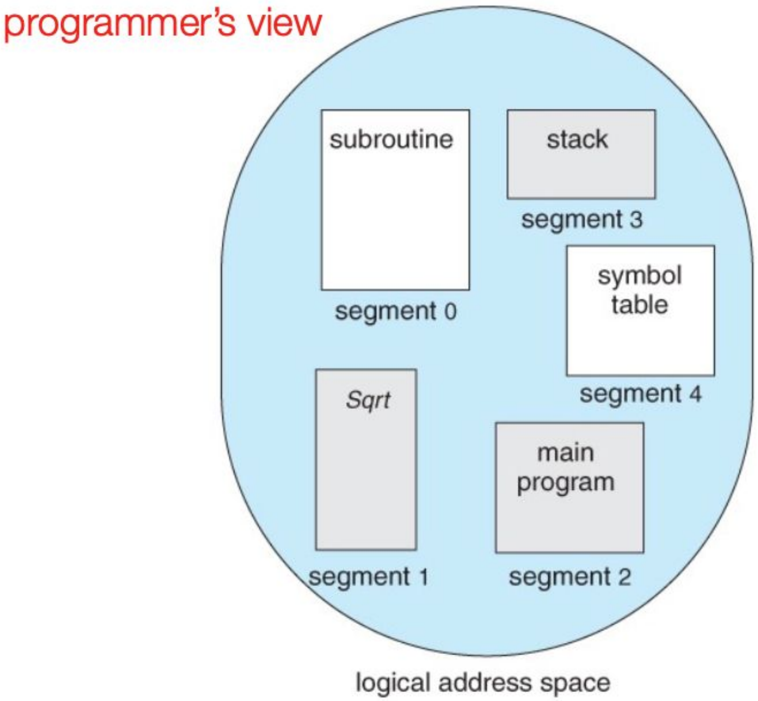
- Try to find the PA for each VA, and see if a VA is valid
- Run **python relocation.py -c** to get the answers

```
---------------- 0KB
|     Code     |
---------------- 2KB
|    Stack     |
---------------- 4KB
|     Heap     |
|      |       |
|      v       |
---------------- 7KB
|    (free)    |
|      ...     |
```

Virtual memory layout

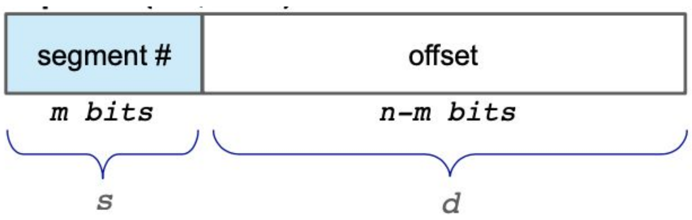
Segmentation

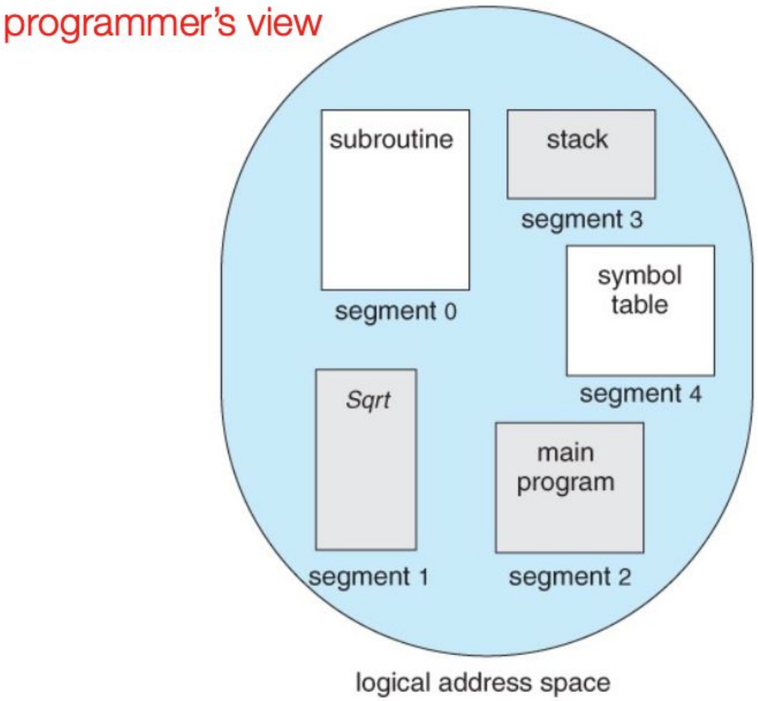
- The memory space of a process is divided into segments with arbitrary size.



Segmentation

- The memory space of a process is divided into segments with arbitrary size.
- The logical address (VA) is divided into segment number and offset





Segmentation

- The memory space of a process is divided into segments with arbitrary size.
- The logical address is divided into segment number and offset
- Segment table tracks the start physical address of each segment

base: start physical address

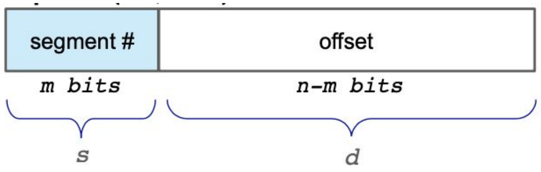
limit: segment size

Segment number	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation - translation

1. Read the segment number (first m bits) and offset (last n-m bits) from VA
2. Look up **base** and **limit** of that segment in the segment table
3. **PA = base + offset**



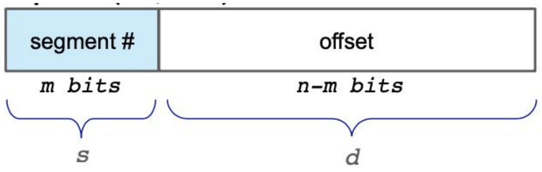
Segment number

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation - protection

- There must be: **offset < limit**
- The origin of **segmentation fault** (but modern computers do not use segmentation)



Segment number

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table

Segmentation - simulator

- Go to **HW-segmentation/** folder
- Run **python ./segmentation.py** to generate segment table and virtual addresses

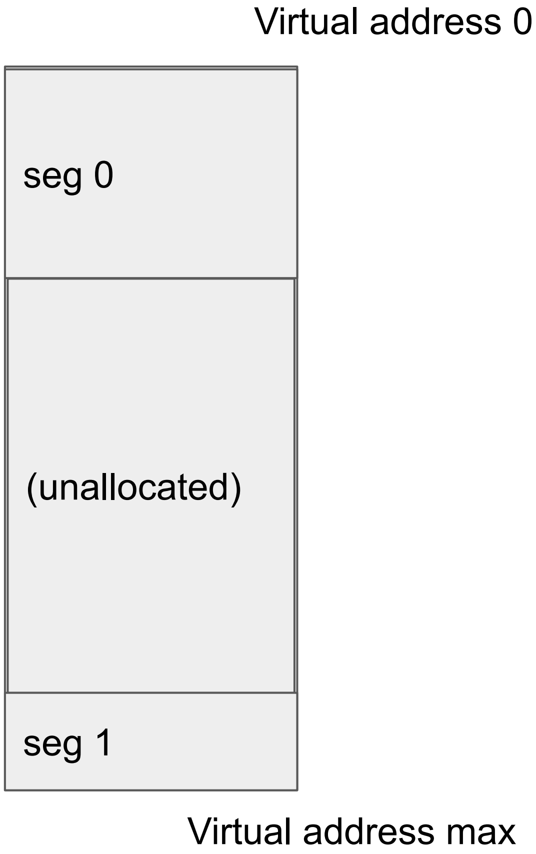
```
ARG address space size 1k
ARG phys mem size 16k

Segment register information:

  Segment 0 base  (grows positive) : 0x00001aea (decimal 6890)
  Segment 0 limit                  : 472

  Segment 1 base  (grows negative) : 0x00001254 (decimal 4692)
  Segment 1 limit                  : 450

Virtual Address Trace
  VA  0: 0x0000020b (decimal:  523) --> PA or segmentation violation?
  VA  1: 0x0000019e (decimal:  414) --> PA or segmentation violation?
```

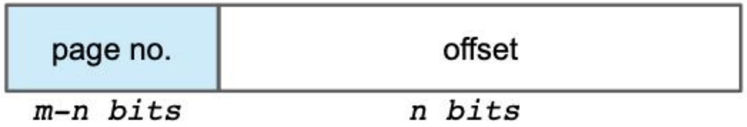


Segmentation - simulator

- Try to find the physical address of each VA
- Run **python ./segmentation.py -c** to check the answer

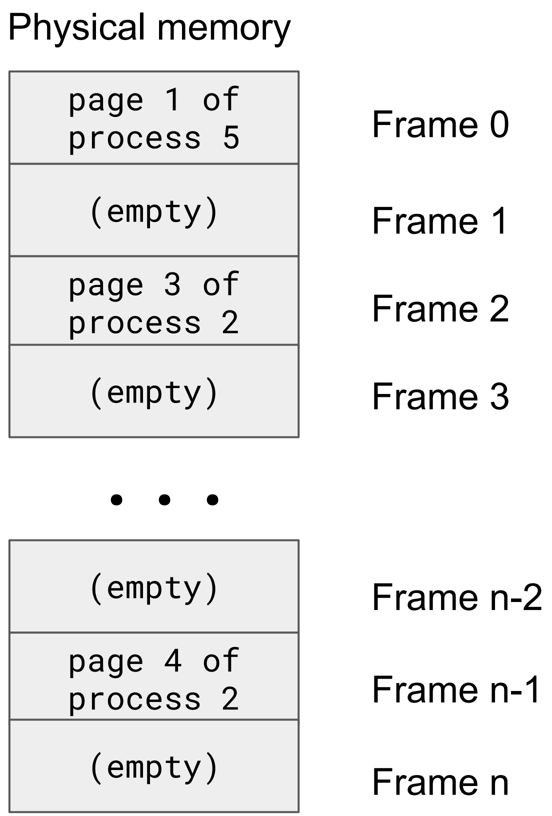
Paging

- The memory space of a process is divided into fixed-size pages.
- The logical address is divided into page number and offset



Paging

- The memory space of a process is divided into fixed-size pages.
- The logical address is divided into page number and offset
- The physical memory is divided into fixed-size frames that holds pages



Paging

- The memory space of a process is divided into fixed-size pages.
- The logical address is divided into page number and offset
- The physical memory is divided into fixed-size frames that holds pages
- The page table of each process stores in which frame each page locates

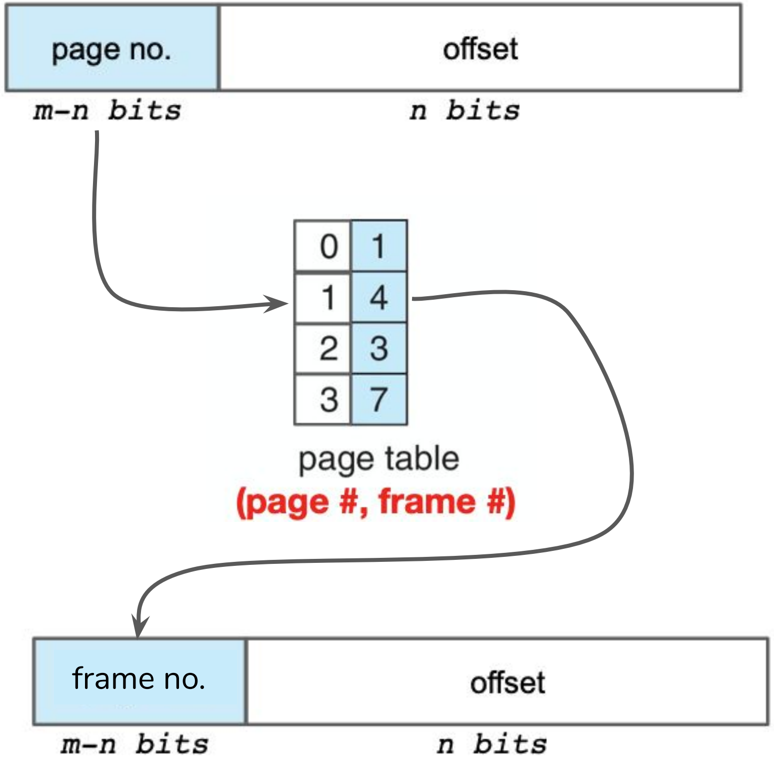
0	1
1	4
2	3
3	7

page table

(page #, frame #)

Paging - translation

1. Read the **page no.** and **offset** from a VA
2. Look up the **frame no.** of that **page no.** in the process' page table
3. Replace the **page no.** in the VA with **frame no.** to get the **PA**



Paging - protection

- The **page no.** must exist in the page table
- Check the control bits of a page table entry: **valid**, **read**, **write**, **execute**...

Paging - simulator

- Go to **HW-Paging-LinearTranslate/** folder
- Run **python ./paging-linear-translate.py** to generate page table and virtual addresses

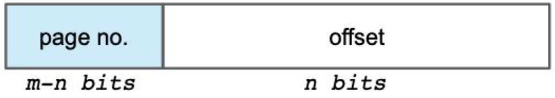
```
     Page Table (from entry 0 down to the max size)
      0x8000000c
      0x00000000
      0x00000000
      0x80000006

   Virtual Address Trace
     VA  0: 0x00003229 (decimal:    12841) --> PA or invalid?
     VA  1: 0x00001369 (decimal:     4969) --> PA or invalid?
     VA  2: 0x00001e80 (decimal:     7808) --> PA or invalid?
     VA  3: 0x00002556 (decimal:     9558) --> PA or invalid?
     VA  4: 0x00003a1e (decimal:    14878) --> PA or invalid?
```

Paging - simulator

- The first bit of a page table entry (PTE) is a valid bit
- Try to translate all VA into PA, and check the valid bit

Page table size



- Assuming we can address 1 byte increments
- The page size is P bytes. Each address has m bits
- How many pages?
  - How many bits for the offset (to address each byte in the page)? n = log2(P)
  - How many bits for the page number? m - n
  - How many pages? 2^(m-n)
- What to store in a page table entry (PTE)?
  - Frame no., control bits, …
- What is the page table size? 2^(m-n) * PTE size

Page table size - simulator

- Go to **HW-Paging-LinearSize/** folder
- Run **python ./paging-linear-size.py** to generate VA size, page size and PTE size
- Compute how big the page table is

Virtual memory

- There could be more pages than frames
- e.g. my x86-64 CPU allows addressing 256TB VM for each process, I currently have 491 processes. The OS allows addressing 491 * 256 = 125,696 TB of VM space, but I only have 16 GB physical RAM…
- Excessive pages will be swapped out the the disk (unfortunately I don’t have that much disk space either)

Virtual memory

- When accessing a page on disk, the OS needs to load it back (page fault)
- But disk is much slower than RAM or registers

Storage	**Access time**	**Rescaled time**
Register	0.38 ns (3.8e-10s)	1 s
Memory	10 ns (1e-8s)	26.3 s
Disk	10 ms (1e-2s)	304 days 14 hours

Virtual memory - have a try

- Go to **HW-Paging-BeyondPhys-Real/** folder
- Run **make** to compile
- Run **./mem 1024** to allocate and access 1024 MB (1GB) memory.

  The program will report the running time
- Try to increase the number

Cache replacement policy

- Cache / RAM are fast but small
  - Physical memory < memory used by processes
  - TLB entries < virtual memory pages
- Which one to replace? (cache replacement policies)
  - First in first out (FIFIO)
  - Least recently used (LRU)
  - Least frequently used (LFU)
  - Most recently used (MRU)
  - etc.

Cache replacement policy - have a try

- Go to **HW-Paging-BeyondPhys-Real/** folder
- Try **./paging-policy.py --addresses=0,1,2,0,1,3,0,3,1,2,1 --policy=LRU --cachesize=3 -c** with different policies, and compare the number of misses

Midterm exam

- Average grade
- Frequently made mistakes

Next

- Download the examples from eClass and have a try
- Ask questions any time