Sharing Main Memory, Segmentation (cont)

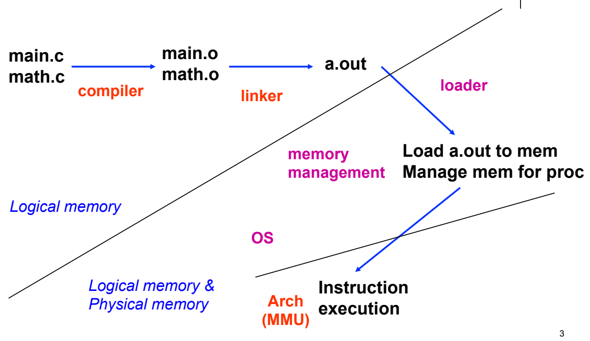
ECE595
Feb 10

Y. Charlie Hu

Outline

- Segmentation
- External fragmentation & Generic dynamic allocation problem
- OS implementation of segmentation

Connecting the dots



main.c
math.c

compiler

main.o
math.o

linker

a.out

loader

memory management

Load a.out to mem
Manage mem for proc

Logical memory

OS

Logical memory &
Physical memory

Arch
(MMU)

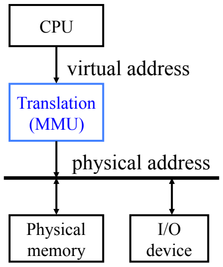
Instruction
execution

3. Dynamic memory relocation

- Instead of changing the address of a program before it's loaded, change the address dynamically *during every reference*
  - Under dynamic relocation, each program-generated address (called a *logical address* or *virtual address*) is translated in hardware to a *physical* or *real address*

Can this be done in software?

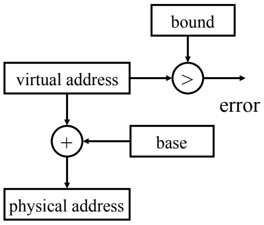
Translation overview



- Actual translation is in hardware (MMU)
- Controlled in software

- CPU view
  - what program sees, virtual addresses
- Memory view
  - physical memory addresses

3.1 Base and bound



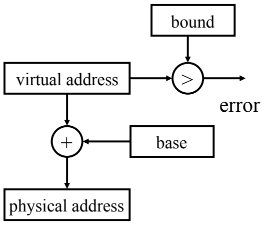
- Built in Cray-1 (1976)
- A program can only access physical memory in [base, base+bound]
- On a context switch: save/restore base, bound registers

- Pros:
  - simple, fast, cheap
  - Can relocate segment

All problems in computer science can be solved by another level of indirection.

David Wheeler

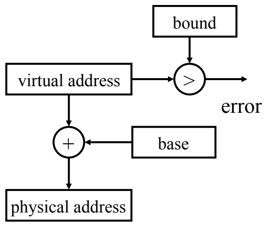
3.1 Base and bound



- The essence:
  - A level of indirection
  - Phy. Addr = Vir. Addr + base

- How to relocate segment in physical memory?
  - From Base 1 to Base 2?

3.1 Base and bound



- Cons:
  - Only one segment
  - How can two processes share code while keeping private data areas (shared editors)?

What have we achieved?

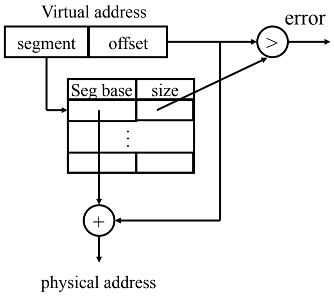
- 4 drawbacks

✓ 1. No protection

✓ 2. Low utilization -- Cannot relocate dynamically
   - Cannot do anything about holes

3. No sharing -- Single segment per process
   - Cannot share part of process address space (e.g. text)
4. Entire address space needs to fit in mem
   - Need to swap whole, very expensive!

3.2 Multiple Segments



- Have a table of (seg, size)
- Further protection: each entry has

  (nil, read, write, exec)
- On a context switch: save/restore the table (or a pointer to the table) in kernel memory

How does this allow 2 processes to share code segment?

Segmentation example

text segment [0x0000, 0x04B0]

foo:

```
019A: LD R1, 15DC
01C2: jmp      01F4
01E0: call     0320
01F4:  X:
```

bar procedure

```
0320: bar:
```

Data segment [0x1000, 0x16A0]

```
15DC: _Y:
```

2-bit segment number, 12-bit offset

Segment	Base	Bounds	RW
0	4000	4B0	10
1	0	6A0	11
2	3000	FFF	11
3	--	--	00

Segmentation example

text segment [0x0000, 0x04B0

foo:

```
019A: LD R1, 15DC
01C2: jmp      01F4
01E0: call     0320
01F4:  X:
```

bar procedure

```
0320: bar:
```

Data segment [0x1000, 0x16A0]

```
15DC: _Y:
```

2-bit segment number, 12-bit offset

Segment	Base	Bounds	RW
0	4000	4B0	10
1	0	6A0	11
2	3000	FFF	11
3	--	--	00

➔ Where is 01F4 in physical memory?

Segmentation example

text segment [0x0000, 0x04B0

foo:

```
019A: LD R1, 15DC
01C2: jmp      01F4
01E0: call     0320
01F4:  X:
```

bar procedure

```
0320: bar:
```

Data segment [0x1000, 0x16A0]

```
15DC: _Y:
```

2-bit segment number, 12-bit offset

Segment	Base	Bounds	RW
0	4000	4B0	10
1	0	6A0	11
2	3000	FFF	11
3	--	--	00

➔ Where is 15DC in physical memory?

Segmentation example

text segment [0x0000, 0x04B0

foo:

```
019A: LD R1, 15DC
01C2: jmp      01F4
01E0: call     0320
01F4:  X:
```

bar procedure

```
0320: bar:
```

Data segment [0x1000, 0x16A0]

```
15DC: _Y:
```

2-bit segment number, 12-bit offset

Segment	Base	Bounds	RW
0	4000	4B0	10
1	0	6A0	11
2	3000	FFF	11
3	--	--	00

➔Suppose SP is initially 265C. Where is it in physical mem?

Segmentation example

text segment [0x0000, 0x04B0

foo:

```
019A: LD R1, 15DC
01C2: jmp       01F4
01E0: call      0320
01F4:  X:
```

bar procedure

```
0320: bar:
```

Data segment [0x1000, 0x16A0]

```
 15DC: _Y:
```

2-bit segment number, 12-bit offset

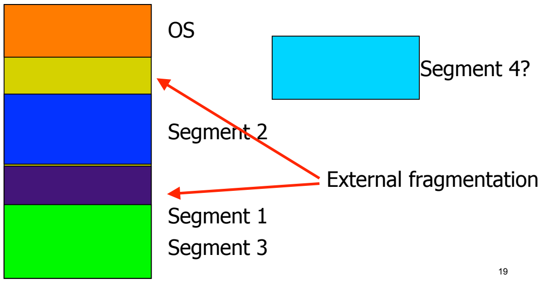
Segment	Base	Bounds	RW
0	4000	4B0	10
1	0	6A0	11
2	3000	FFF	11
3	--	--	00

➔ which portions of the virtual and physical address spaces are used by this process?

Pros/cons of segmentation

- Pros:
  - Process can be split among several segments
    - Allows sharing
  - Segments can be assigned, or swapped independently
- Cons:
  - External fragmentation: many holes in physical memory
    - Also happens in base and bound scheme

Simple multiprogramming:
Single segment per process, static relocation



OS

Segment 4?

Segment 2

External fragmentation

Segment 1

Segment 3

What fundamentally causes external fragmentation?

1. Segments of many different sizes
2. Each has to be allocated contiguously

Can we solve Dynamic memory allocation problem?

- Problem: External fragmentation caused by holes too small
- How much can a smart allocator help?
  - The allocator maintains a free list of holes
  - Allocation algorithms differ in how to allocate from the free list

Dynamic allocation algorithms

- Best fit: allocate the smallest chunk big enough
- First fit: allocate the first chunk big enough
  - Rotating first fit
- Is best fit necessarily better than first fit?
  - Example: 2 free blocks of size 20 and 15
  - If allocation ops are 10 then 20, which one wins?
  - If ops are 8, 12, then 12, which one wins?

Dynamic allocation algorithms

- Analysis shows
  - First fit tends to leave average-size holes
  - Best fit tends to leave some very large holes, very small holes
- Knuth claims that if storage is close to running out, it will run out regardless of which scheme is used

  → Pick the easiest or most efficient (e.g. first fit)

Segmentation: OS implementation

- Keep segment table in PCB
- When creating process, allocate space for segments, fill in PCB bases/bounds
- When process dies, return physical space used by segments to free pool
- Context switch?
  - Saves old segment table / Loads new segment table to MMU
  - What about context switch of threads?
  - True-or-false: CS between threads of same process cheaper than CS between processes

[lec2] Kernel data structure: Process Control Block (Process Table)

- Process management info
  - State (ready, running, blocked)
  - PC & Registers, parents, etc
  - CPU scheduling info (priorities, etc.)
- Memory management info
  - Segments, page table, stats, etc
- I/O and file management
  - Communication ports, directories, file descriptors, etc.

Managing segments (cont)

To enlarge a segment:

- See if space above segment is free. If so, just update the bound and use that space
- Or, move this segment to disk and bring it back into a larger hole (or maybe just copy it to a large hole)

Managing segments (cont)

- When there is no space to allocate a new segment:
  - Compact memory – how?

Summary: Evolution of Memory Management (so far)

Scheme	How	Pros	Cons
Simple uniprogramming	1 segment loaded to starting address 0	Simple	1 process 1 segment No protection
Simple multiprogramming	1 segment relocated at loading time	Simple, Multiple processes	1 segment/process No protection External frag.
Base & Bound	Dynamic mem relocation at runtime	Simple hardware, Multiple processes Protection	1 segment/process, External frag.
Multiple segments	Dynamic mem relocation at runtime	More hardware, Protection, multi segs/process	External frag.