

CS 414 : Operating Systems

UNIVERSITY OF VIRGINIA
Department of Computer Science

Spring 2008

Topic 12: Sharing Main Memory -- Segmentation

- Goal: let several processes co-exist in memory.
- Issues:
 - Transparency: No process should need to be aware of the fact that memory is shared. Each must run regardless of the number and/or locations of processes.
 - Protection: processes must not be able to corrupt each other.
 - Efficiency (both of CPU and memory) shouldn't be degraded badly by sharing. After all, what is the purpose of sharing?
- Simple uniprogramming with a single segment per process:
 - Highest memory holds OS.
 - Process is allocated memory starting at 0, up to the OS area.
 - When loading a process, just bring it in at 0.
- Relocation.
 - Because several processes share memory, we can't predict in advance where a process will be loaded in memory. This is similar to a compiler's inability to predict where a subroutine will be after linking.
 - Relocation adjusts a program to run in a different area of memory. Linker is an example of static relocation used to combine modules into programs.
- Static relocation
 - Highest memory holds OS.
 - Processes allocated memory starting at 0, up to the OS area.

- When a process is loaded, relocate it so that it can run in its allocated memory area
 - Problem: not much protection; hard to relocate later, once it starts running.
- Dynamic relocation: instead of changing the addresses of a program before it's loaded, change the address dynamically *during every reference*.
 - Under dynamic relocation, each program-generated address (called a *logical* or *virtual* address) is translated in hardware to a *physical* or *real* address. This happens as part of each memory reference.
 - Dynamic relocation leads to two views of memory, called *address spaces*.
 - A good case of a problem solved by indirection. What is the name of the game? Flexibility -- to move things around freely.
- Base & limit relocation:
 - Two hardware registers: base address for process, limit register that indicates the last valid address the process may generate.
 - Each process must be allocated contiguously in real memory.
 - On each memory reference, the virtual address is compared to the limit register, then added to the base register. A limit violation results in an error trap.
 - Each process appears to have a completely private memory of size equal to the limit register plus 1. Processes are protected from each other. No address relocation is necessary when a process is loaded.
 - OS runs with relocation turned off (a bit in the processor status word controls relocation). It is called unmapped. Why is it necessary?
 - OS must prevent users from turning off relocation or modifying the base and limit registers
 - Problem: how do you get kernel bit on when you jump?
 - Base & limit is cheap -- only 2 registers -- and fast -- the add and compare can be done in parallel.
 - Problem with base & limit relocation:
 - Only one segment. How can two processes share code while keeping private data areas (e.g. shared editor)?

- Multiple segments:
 - Permit process to be split between several areas of memory.
 - Use a separate base and limit for each segment, and also add two protection bits (read and write).
 - Each memory reference indicates a segment and offset.
 - Top bits of address select segment, low bits the offset.
 - Segment table holds the bases and limit for all the segments of a process.
 - Memory mapping procedure consists of table lookup + add + compare.
- Segmentation example: 3-bit segment number, 9-bit offset.

- Segment table (all numbers in octal):

Segment	Base	Limit	RW
0	4000	577	10
1	1000	777	11
2	2000	677	10
3	0	777	11
4	-----	-----	00
5	-----	-----	00
6	-----	-----	00
7	3000	777	11

- Main program (addresses are virtual, octal):

```

242:      PUSH #3106
246:      CALL SIN
250:      ...

```

- SIN procedure (addresses are virtual):

```

2360:     MOVE 2(SP),R2
2364:     MOVE @R2,R3
...
RETURN

```

- Where is 242 in physical memory?
- Suppose the PC was 242 and SP was 7650. What will they contain after the execution of PUSH #3106?
- What will be in physical address 3650?
- What will be lowest address that will incur address violation?
- Which portions of the virtual and physical address spaces are used by this process?

- Managing segments:
 - Keep copy of segment table in process control block.
 - When creating process, allocate space for segment, fill in PCB bases and limits.
 - When switching contexts, save segment table in old process's PCB, reload it from new process's PCB.
 - When process dies, return segments to free pool.
 - When there's no space to allocate a new segment:
 - Compact memory (move all segments, update bases) to get all free space together.
 - Or, swap one or more segments to disks to make space (must bring segments back in before letting process run).
 - To enlarge segment:
 - See if space above segment is free. If so, update limit and use that space.
 - Or, move the segment above this one to disk, in order to make the memory free.
 - Or, move this segment to disk and bring it back into a larger hole (or, maybe just copy it to a larger hole).

- Advantage of segmentation?
- Problem:
 - External fragmentation: segments of many different sizes, have to be allocated contiguously. This problem also applies to base and limit schemes.
 - Solution?