Chapter 9: Memory Management

- Background
- General Memory Management
- Compiling, Linking, and Loading
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Segmentation

Background

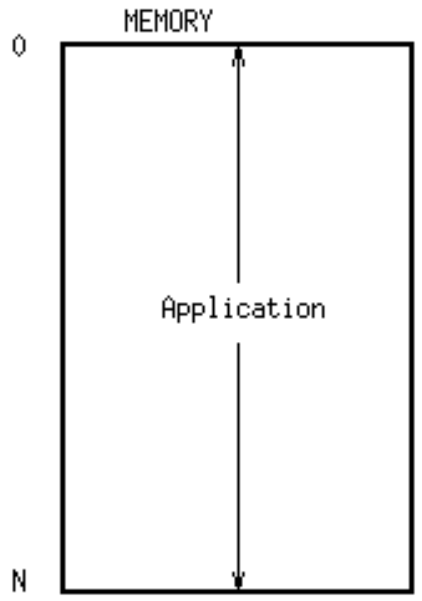
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are the only storage CPU can access directly
- Access time differs:
  - Register access in one CPU clock (or less)
  - Main memory can take many cycles
- Cache sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Memory Management

- Several questions we can ask about our O/S:
  - What restrictions does the operating system and hardware place on user (process) access to memory?
  - What facilities are provided by the operating system and hardware for memory access by processes?
  - What protection is given by the memory management scheme?
- Many different memory management schemes...

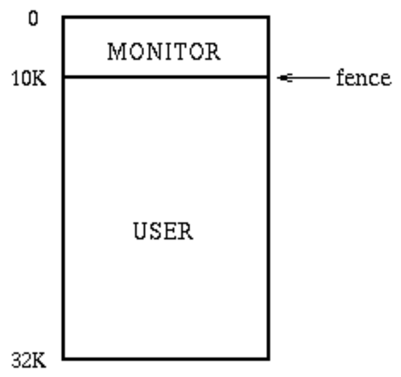
Bare Machine (No Memory Management)

- The application has complete control over all the memory in the machine
- No separate operating system is present
- There is no protection of any memory location
  - the application process can write to any memory location



Resident Monitor

- Memory is divided into 2 parts:
  - the monitor (or operating system)
  - the user (or application)



- The monitor is generally placed at the top or the bottom of memory
- We must have a way of protecting the operating system code from corruption
  - application may either maliciously or accidentally write into operating system space

Resident Monitor

- Memory protection can be implemented in hardware using a fence:
  - A memory access by the user program on the OS side of the fence generates a hardware trap (kind of like a segmentation fault)
  - Depending on the hardware, this can slow down the execution as each memory reference must be checked.
  - This requires 2 modes of execution
    - **User** mode: hardware checks all memory references
    - **Supervisor** mode: hardware checks disabled

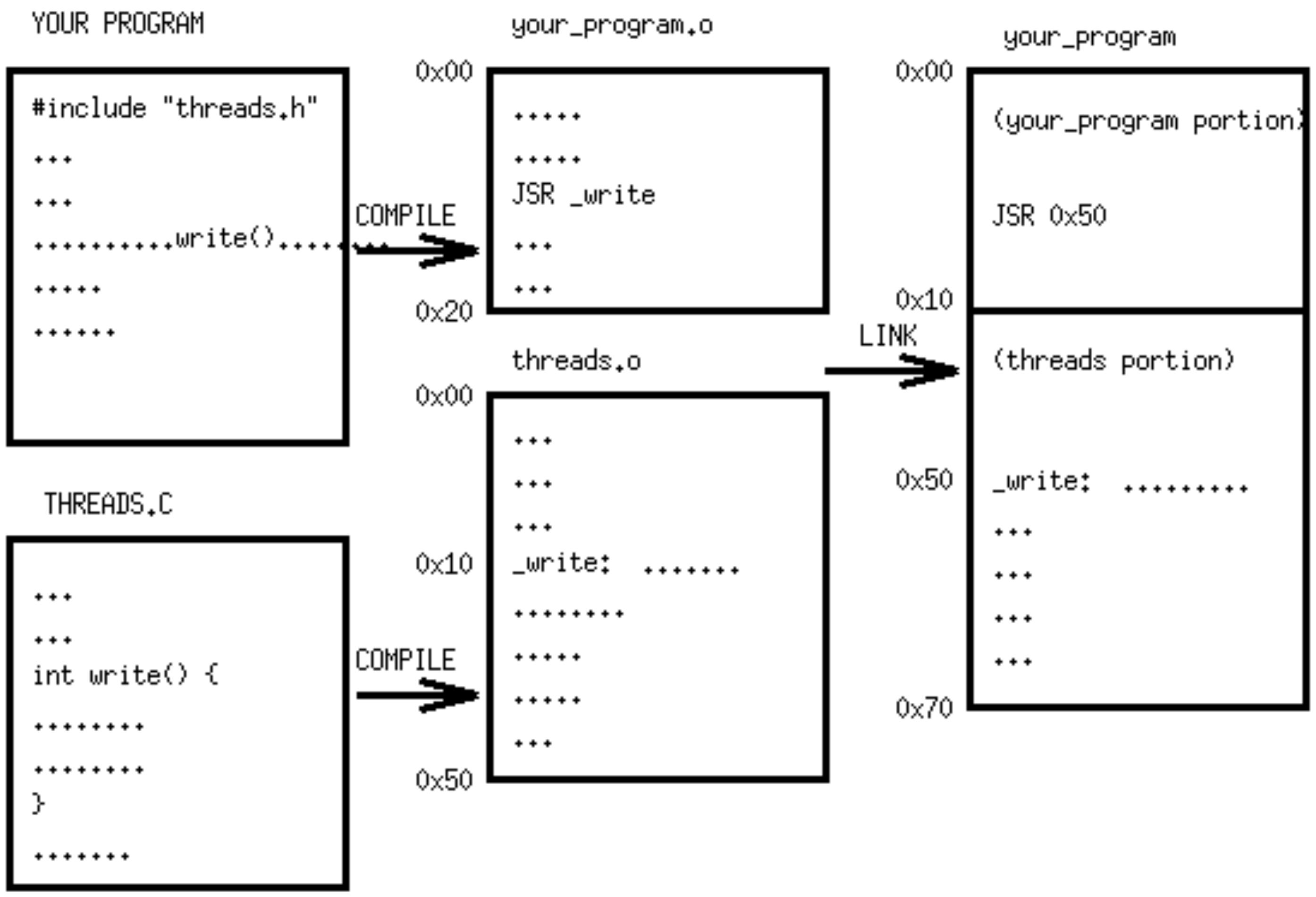
Problems with Resident Monitors

- Early monitors: the fence had to be built into the hardware
  - The fence remained constant
  - Constrained O/S size: Either some memory was wasted, or part of the O/S was unprotected
- Later: the fence value was placed in a register
  - The fence register could accommodate the size of the O/S
- Fence location change $\Rightarrow$ problems:
  - means re–linking programs to set addresses

Compiling, Linking, and Loading

- compiling: source $\rightarrow$ object file
- linking: object files $\rightarrow$ load module
  - resolve addresses of references (internal and external)
  - if the load module will be loaded at the same address each time, the linker can generate absolute addresses
  - otherwise, relocatable addresses/objects
- loading: load module $\rightarrow$ main memory

Linking



YOUR PROGRAM
#include "threads.h"
...
...
..........write().....
.....
......
THREADS.C
...
...
int write() {
....
....
}
....
COMPILE
COMPILE
your_program.o
0x00
.....
.....
JSR _write
...
...
0x20
threads.o
0x00
...
...
...
0x10
_write: ....
....
.....
.....
...
0x50
LINK
your_program
0x00
(your_program portion)
JSR 0x50
0x10
(threads portion)
0x50
_write: ....
...
...
...
...
0x70

Static Relocation

- The linker always generates a module with addresses assuming the module will be loaded at address zero
- The loader scans the module and adjusts every address by adding the actual load location to each address
- The loader must be "smart" enough to be able to interpret the instructions in the load module - at least enough to identify the addressing modes of the instructions it finds
- Once a module has begun execution it cannot be relocated

Dynamic Relocation

- In this scheme, the linker still generates addresses assuming a start position of zero
- MMU updates each memory reference by adding the actual load location to the memory address as the program runs
- An application never sees a *physical memory address* - only the *logical memory addresses* generated by the linker
  - Conversion between physical and logical addresses is done by the Memory Management Unit (MMU)
- The loader need not be "smart" - it simply copies the load module into its load location (simpler, faster)