Address Space Protection

- How are address spaces protected?
  - Separation of translation data

- How is translation data protected?
  - Can update translation data only if mode bit set

- How is mode bit protected?
  - Sets/reset mode bit when transitioning from user-level to kernel-level code and back
  - Transitions limited by interrupt vector table

Protection boils down init process which sets up interrupt vector table when system boots up

Process creation

- Steps
  - Allocate process control block
  - Initialize translation data for new address space
  - Read program image from executable into memory
  - Initialize registers
  - Set mode bit to “user”
  - Jump to start of program

- Need hardware support for last few steps
  - Similar to switching from kernel to user process after system call

Multi-process issues

- How to partition physical memory allocation among processes?
  - Fairness versus efficiency

- Global replacement
  - Can evict pages from this process or other processes

- Local replacement
  - Can evict pages only from this process
  - Must still determine how many pages to allocate to this process

- Pros and cons?

Thrashing

- What would happen if many large processes all actively used their entire address space?

- Performance degrades rapidly as miss rate goes up
  - Avg access time = hit rate * hit time + miss rate * miss time
  - E.g., hit time = .0001 ms; miss time = 10 ms
    - Average access time (100% hit rate) = .0001 ms
    - Average access time (1% miss rate) = ?
    - Average access time (10% miss rate) = ?

Solutions to Thrashing

- Buy more DRAM
  - Price per GB fallen by 4x since 2009

- Run fewer processes for longer
  - Example: Longer time slice
  - Reduces page faults
**Working set**

- Thrashing depends on portion of address space actively used by each process
  - What do we mean by “actively using”?  
- Working set = all pages used in last T seconds  
  - Larger working set → process needs more physical memory to run well (i.e., avoid thrashing)  
- Sum of all working sets should fit in memory  
  - Only run subset of processes that fit in memory  
- How to measure size of working set?  
  - Periodic sweep of clock hand in LRU clock

**Project 3**

- Hand in mid-term regrade requests  
- Draw state machine for any virtual page  
  - Write test cases to exercise any path in state machine  
- Multi-process test cases  
  - fork() to create multiple processes within a program  
  - Use vm_yield() to coordinate between processes

**Unix process creation**

- System calls to start a process:  
  1. Fork() creates a copy of current process  
  2. Exec() replaces current address space with specified program  
- Why first copy and then overwrite?  
  - Windows: CreateProcess(program, args)  
- Any problems with child being an exact clone of parent?

**Cloning**

**Subtleties in handling fork**

- Buggy code from autograder:  
  ```c
  if (!fork()) {  
    exec(command);  
  }  
  while(child is alive) {  
    if (size of child address space > max) {  
      print “process took too much memory”;  
      kill child;  
      break;  
    }  
  }  
  ```
- What is the race condition here?
Avoiding work on fork

- Copying entire address space is expensive
- Instead, Unix uses **copy-on-write**
  - Assign reference count to each physical page
  - On fork(), copy only the page table of parent
    - Increment reference count by one
  - On store by parent or child to page with refcnt > 1:
    - Make a copy of the page with refcnt of one
    - Modify PTE of modifier to point to new page
    - Decrement reference count of old page

Copy-on-write: Example

Parent about to fork()

Copy-on-write: Example

Copy-on-write of parent address space

Child modifies 2\textsuperscript{nd} virtual page

Parent modifies 2\textsuperscript{nd} virtual page

Parent exits
Implementing a shell

while (1) {
    print prompt
    ask user for input (cin)
    parse input //split into command and args
    fork a copy of current process (the shell prog.)
    if (child) {
        redirect output to a file/pipe, if requested
        exec new program with arguments
    } else { //parent
        wait for child to finish, or
        run child in the background and ask for
        another command
    }
}