Virtual Memory

CS 217

Memory Management

• Problem 1:
  ◦ Two programs can’t control all of memory simultaneously

• Problem 2:
  ◦ One program shouldn’t be allowed to access/change the memory of another program

• Problem 3:
  ◦ Machine may have only 256MB of memory, while virtual address space is 4GB

Operating system must manage sharing of physical memory between many processes
Virtual Memory

- Basic idea
  - Programs don’t (and can’t) name physical addresses
  - Instead, they name virtual addresses
    (each process has own address space)
  - The kernel translates each virtual address into a physical address
    before the operation is carried out

- Advantages
  - Can run many programs at once, without them worrying that they will use the same physical memory
  - Kernel controls access to physical memory, so one program can’t access or modify the memory of another
  - Can run a program that uses more virtual memory than the computer has available in physical memory

Segmentation

- Allocate memory for segments
  - Provide mapping from addresses in segments to physical memory

- Use base and limit registers to translate virtual addresses to physical addresses
**Segmentation**

- Allocate memory for segments
  - Provide mapping from addresses in segments to physical memory
- Problems:
  - Segments may grow
  - Fragmentation
  - Large processes
  - Swapping efficiency

**Paging**

- Motivation
  - Mapping entire segments is too coarse granularity
  - Mapping individual bytes is too fine granularity
- Pages
  - Divide up memory into blocks, called pages (~4KB)
  - Each virtual page can be mapped to any physical page
  - Each translation involves two steps:
    - Decide which physical page holds the virtual address
    - Decide what offset the virtual address is inside the page
  - The physical address is formed by gluing together the physical page number and the offset within the page
Paging

- Page table maps virtual addresses to physical addresses

Paging (cont)

Paging Example

- Each process has its own page table

4-byte pages
- Consider the virtual address \(11_{10}=1011_2\)
- Chop it into two parts
  - Virtual page number \(2_{10}=10_2\)
  - Offset within page \(3_{10}=11_2\)
- Look up the page table and find that virtual page 2 is stored at physical page 1
- The physical address is \(7_{10}=0111_2\)

Silberschatz & Peterson
Paged Segmentation

Swapping

- What happens if cumulative sizes of segments exceeds physical memory?
Swapping to Disk

• If all the virtual memory can’t fit in physical memory, the OS can temporarily stash some pages on disk
  ◦ Can support virtual memory bigger than physical memory

Page Table

• The OS stores for each page ...
  ◦ Physical page number (24 bits)
  ◦ Cacheable bit (C)
  ◦ Modified bit (M)
  ◦ Referenced bit (R)
  ◦ Access permissions (Read only, Read/write)
  ◦ Valid/invalid (V)
Page Faults

• If process accesses virtual address that maps to a page not in memory, then the OS must fetch that page from disk
• Since most references follow others on same page, the cost of reading from disk is amortized across many references

Page Replacement

• When read one page from disk, another page must be evicted?
• Which page should be replaced?
  ◦ Ideal:
    – One that will be accessed furthest in future
  ◦ Practical heuristics:
    – Least recently used
    – Least frequently used
    – Etc.

```c
void StringArray_read(StringArray_T s, FILE *fp)
{
    char string[MAX_STRING_LENGTH];
    s->nstrings = 0;
    while (fgets(string, MAX_STRING_LENGTH, fp)) {
        StringArray_grow(nstrings+1);
        s->strings[*(s->nstrings)++] = strdup(string);
    }
}
```
Working Sets

• Locality of reference
  ◦ Most memory references are nearby previous ones

• Working set
  ◦ At any point in a program’s execution, usually a small region of memory is accessed frequently
  ◦ The region of memory (working set) changes during the course of execution

```c
int main()
{
    Array_T *strings;
    strings = ReadStrings(stdin);
    SortStrings(strings);
    WriteStrings(strings, stdout);
    return 0;
}
```
Thrashing

- What happens when cumulative size of working sets exceeds capacity of physical memory?

Storage Hierarchy

- Registers
  - ~128, 1-5ns access time (CPU cycle time)
- Cache
  - 1KB – 4MB, 20-100ns (multiple levels)
- Memory
  - 64MB – 2GB, 200ns
- Disk
  - 1GB – 100GB, 10ms
- Long-term Storage
  - 1TB, 1-10s
Summary

- Memory management
  - Important function of operating system
  - Understanding how it works is critical to effective system development

- Virtual memory
  - OS & Hardware support for mapping virtual addresses to physical addresses
  - Mapping is usually at page granularity, which facilitates ...
    - Relocation
    - Swapping to disk
    - Protection
    - Fragmentation
    - Sharing