CIS452 Week-09 Notes

Winter 2016

March 14 - March 18, 2016

Scribes: *Hayden Miedema* and *Sage Heiss*

OS as a resource manager

- Chapters 3-6
  - As a process manager, the OS creates an illusion that..
  - A user owns the CPU to his/herself
  - A process is running on the CPU at all times
  - Programmers can ignore the fact at runtime, their program may not be running on the CPU all the time

OS: The memory illusionist - Creates illusion of...

- A process owns the entire RAM to itself
- A process resides in RAM all the time
- The code (generated by the compiler) starts at address zero
- Code and data are contiguous in memory
- A process has access to (tera—peta—exa)bytes* of memory space

OS: The memory facts

- Your process has to share the RAM with many other processes
  - OS task: Memory Protection and sharing
- At times a process may be swapped out of RAM
  - OS task: Process Relocation

The code may be loaded (and reloaded) at any memory address

- OS task: Load a process to a free memory region

Code and data may be split into segments/pages
The actual amount of space accessible to a process is the RAM size

Memory sharing and Protection

- Several processes may concurrently reside in RAM
  - Each process has diff. size
  - Each process is placed at diff. memory location
- Sharing: divide the RAM into regions, place a process in one region
- Protection: prohibit one process to peek into anothers region
  - Protection violation must be (initially) detected/reported by the memory hardware and (later) handled by the OS
- Two special Registers:
  - Base register: the starting address of a region
  - Limit register: size of the region
  - These two registers are saved/restored during a context-switch
- Select Hightest Priority Task first
- Always preemptive - Let High priority tasks interrupt low priority
- Without aging or some other algorithm, low priority processes can starve

Address Binding

- A user program refers to data/variables/functions using symbolic names
- Compile-Time Address Binding
  - Compiler and linker bind each symbolic name to a memory address
- Load-Time Address Binding
  - The OS loads the binary executable to an open space in RAM
- Run-Time Address Binding
  - A process may be relocated at a different address
  - A process may call functions shared among several process (the actual location of these functions must be resolved at runtime)
  - Windows (.DLL) and Unix Shared Objects (.so) (see full diagram in Google Slides)
- Logical/Virtual Address: generated by CPU
- Physical Address: address issued to memory

Dynamic Loading/Linking

- Problem: Large Process size, limited RAM size
- Dynamic Loading (Overlay)
  - Routines needed by a process are loaded on-demand
  - No special OS support needed, the user process is responsible for loading its overlay
- Dynamic Linking Shared Libraries
  - Opposite variant of static linking
  - At linking time, the linker includes only a stub about the target functions in the shared library
  - The actual linking to the shared libraries are postponed until runtime
  - When a new version of shared libraries becomes available, a user program will invoke the newer version without recompilation/relinking
- Similar to multi-feedback, long period processes get low priority and vice versa.

Swapping

- A process must be resident in RAM to run
- When more memory is needed, the Medium Term Scheduler may swap out processes
- Processes in the ready queue are good candidates for swapping out
- When a process is (being) swapped out
  - The entire current process image is dumped to swap space
  - All memory areas owned by the process are released
- Swapping allows the system to host several processes whose total memory requirement may exceed the physical RAM size
- Swap space/Swap disk: a designated disk used for the binary process image of swapped out processes

Swapping-Related Issues

- OS maintains two ready queues
  - Processes which are ready and resident in RAM
  - Processes which are ready but swapped out
- When a process is swapped (back) in, it may resume execution at a different physical
- A process to be swapped out should be completely IDLE
  - No pending I/O (because pending I/O requires target buffer to be resident in RAM)
- Swapping is very SLOOOOW (may take seconds to complete)
  - HD transfer rate is about 50 MB/sec
  - SSD transfer rate is about 750 MB/sec
- Swapping is not beneficial in terms of speed, but swapping is very key/important to allow the system to take more processors, thus more processes in the system and greater CPU utilization.

Contiguous Memory Allocation

- Goal: shared RAM by many processes
- Memory Partitions
  - One partition can only hold one process
- Fixed-size vs. Variable size partitions
  - Fixed: location and size of each partition are predefined by the OS
    - Size of each partition will stay the same throughout that boot cycle
  - Variable: location and size of each partition are determined on-demand, per partition request.

Fixed-Sized Partitions

- RAM is divided into N partitions (of different sizes, but fixed)
- Only max N processes can resided in RAM at any time
  - One partition = one process
  - Fixed multiprogramming level
- When a process is loaded to RAM, the system selects a free partition big enough to fit the process
- (Internal) Fragmentation: unused portion (wasted space) of a partition
  - Scheduling Issues? Ready Queue(s)?
    - N ready queues, since each partition is a different size

Variable-Sized Partitions

- Partitions are created on-demand as processes are loaded
  - Partition size = size of the process just loaded
  - No internal fragmentation
- But, when a process terminates, it creates a hole (free partition)
- The OS maintains
  - A list of allocated partitions
  - A list of holes (free partitions) link the holes themselves into a linked list
- When a process is loaded, the OS looks for a hole big enough for the process
  - Unused portion of the hole becomes a smaller hole

Algorithms

- First Fit: scan from head and put in first found partition
- Next Fit: scan from last fit point and scan until the first fit appears
- Best Fit: scan entire RAM and give minimum leftover partition
- Worst Fit: scan entire RAM and give maximum leftover partition

Hex Arithmetic

- See table on handout or Google Slides

Segmentation and Segmentation Hardware

- Base and Limit Register pairs get replaced with a segment table
- Since each is contiguous (stack, heap, etc.) we can keep them in the table sequentially
- More significant bits are used for the segment ID (seg)
- Less significant bits are then used for the offset
- These can both be used to find the base address and size of that given segment
- If no errors, the hardware will add the two numbers (size and base addr.) to give the final physical address
- Improvements over Variable-sized Segmentation:
    * Less wasted space

Paging

- Partition RAM into equal fixed-size blocks (frames) and split processes into blcoks of the same size
- Logical pages ust be mapped
- Only the current page(s) must be visible, all other pages can be inaccessible
- The base address of a page table is kept in Page Table Base Register (PTBR)
- pages (of a process) can be loaded to RAM on-demand
- Page tables have the same structure as the segmentation table, but we do not need to keep the size since they are all the same

Paging vs. Segmentation

- design of a page table is much simpler than a segment table
- In segmentation, one segment (for instance code segment) is mapped contiguously
- Obviously, page is now used over segment