Operating Systems and Networks

Lecture 03:
Behzad Bordbar
recap

- Cpu
- Memory
- Device controller
- What happens when we start a machine
- Memory: register, cache, ...
Contents

- See the jobs processes
- Time sharing
- Memory management
- Swapping
- Logical vs physical address space
- Protection
- Segmentation
- Paging
Jobs and processes

Start a few jobs
$ gedit myfile &
$ xload &
$ xterm &
Then
$jobs -l
[jobid] processid
+ current running – previous
Processes $ps -aux
Multitasking in CPU

- OS picks and begins to execute one of the jobs in memory.
- Eventually job may have to wait for some task, such as an I/O operation, to complete.
- OS switches to, and executes, another job.
- When that job needs to wait, the CPU switches to another job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back.

**Time-sharing:**

- CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.
Time sharing

requires CPU scheduling of user tasks. But how?

- Each user has at least one separate program in memory.
- A program loaded into memory and executing is called a **process**. We will study this in details!
- When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O.
- I/O takes long long long time compare to execution! (look at the speed of access slides!)
Swapping

1. swap out

2. swap in
swapping

- System maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- Dispatcher checks to see whether the next process in the queue is in memory.
- If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process.
- It then reloads registers and transfers control to the selected process.

Cost is high:
- The user process is 100 MB and a transfer rate of 50 MB/sec.
- Takes 2 second to swap out (total 4 sec for swap in and out).
Challenges of swapping

- If we want to swap a process, we must be sure that it is completely idle.
- Example: A process may be waiting for an I/O. Assume that the I/O operation is queued because the device is busy. If we were to swap out process P1 and swap in process P2, the I/O operation might then attempt to use memory that now belongs to P2.
- Some OS' don't support swapping: Mobile phones
- Execute I/O into the buffer. Transfers between operating-system buffers and process memory then occur only when the process is swapped in (extra overhead).
Logical vs physical address

Logical address: address produced by the CPU
Physical address: the address loaded into memory-address registered

At runtime the mapping between logical and physical address is carried out via a hardware piece called “Memory management Unit (MMU)”

The user program works with the logical address and the MMU translate it to physical address.

Think of for example, local address for a process from location 0 to 15000. Then we add some amount R the physical address is from R to R+15000.
Memory protection

- prevent a process from accessing memory it does not own.
- relocation register: contains the value of the smallest physical address
- limit register: contains the range of logical addresses
- Each logical address must fall within the range specified by the limit register.
- MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory.
Memory protection

- CPU
- Logical address
- Limit register
- Relocation register
- Physical address
- Memory
- Trap: Addressing error
Fragmentation

- processes are loaded and removed from memory, the free memory space is broken into little pieces.
- External Fragmentation: when there is enough total memory space to satisfy a request but the available spaces are not contiguous. Lots of holes.
- Internal Fragmentation: unused memory that is internal to a partition. Process allocated 18,464 bytes but requests 18,462 bytes
- Compactification: to shuffle the memory contents so to place all free memory together in one large block. Not possible always possible and can be expensive
- Other methods: allow non-contiguous logical address space, i.e. allocated memory space when available. segmentation and paging.
Segmentation

- A logical address space is a collection of segments.
- Each segment has a name (number) and a length. The addresses specify both the segment name and the offset within the segment.

\[
<\text{segment-number, offset}>
\]

C compiler might create separate segments for
- 1. The code
- 2. Global variables
- 3. The heap, from which memory is allocated
- 4. The stacks used by each thread
- 5. The standard C library. Will be discussed later
Segmentation hardware

s: segmentation number
d: offset

CPU

segment table

physical memory

trap: addressing error
paging

- Segmentation may still result in external fragmentation
- Need for compactification
- Sizes are fixed!

Paging addresses the above problems and is widely used.
Paging (general idea)

- break physical memory into fixed-sized blocks called **frames**
- breaking logical memory into blocks of the same size called **pages**.
- When a process is executed, its pages are loaded into **any** available memory frames from source (file system or backing store)
- backing store is divided into same fixed-sized

Advantages:

- the logical address space is now totally separate from the physical address space,
- process can have a logical address space larger than physical memory.
Hardware support
32 byte memory with 4-byte pages

Formula: page_number × size + offset

Example: logical address 7: page1, offset 3 ... count it :-) 6×4+4 = 28
Further reading
about memory, fragmentation,... paging sections 8.1 to 8.5.

- It is an interesting, well written and informative.
Dual mode: user mode to kernel mod

Figure from our book.

- Dual mode OS protect from harm caused by privileged instructions
- Extended to multi mode by domains: Dom0, DomU
Why do we need hardware support?

- MS-DOS: Intel 8088 architecture, which has no mode bit
- User program can wipe out the whole OS
- Programs are able to write to a device ...

In dual mode:
- Hardware detects errors that violate modes and handle them by Os
- Stops user program attempts to execute an illegal instruction or to access memory of other users

When error detected:
- OS must terminate the program
- OS gives error message
- Produces memory dumps by writing to a file (users can check or OS vendors can check (Sun)).
Demo lecture

Contents

- Service view (provider of services) of the OS
- Shell
- Everything a directory
- mkdir, mv, cp,...
- Access control
- Find, grep
- |, >, >>, ; and their differences
- wget,...