Memory Management

- In multi-programming OS, multiple programs are loaded in memory.
- RAM memory should be divided for multiple processes running concurrently.
- Memory Mgmt scheme used by any OS depends on the MMU hardware used in the machine.
- There are three memory management schemes are available (as per MMU hardware).
    1. Contiguous Allocation
    2. Segmentation
    3. Paging

Contiguous Allocation

Fixed Partition

- RAM is divided into fixed sized partitions.
- This method is easy to implement.
- Number of processes are limited to number of partitions.
- Size of process is limited to size of partition.
- If process is not utilizing entire partition allocated to it, the remaining memory is wasted. This is called as "internal fragmentation".

Dynamic Partition

- Memory is allocated to each process as per its availability in the RAM. After allocation and deallocation of few processes, RAM will have few used slots and few free slots.
- OS keep track of free slots in form of a table.
- For any new process, OS use one of the following mechanism to allocate the free slot.
    - First Fit: Allocate first free slot which can accommodate the process.
    - Best Fit: Allocate that free slot to the process in which minimum free space will remain.
    - Worst Fit: Allocate that free slot to the process in which maximum free space will remain.
- Statistically it is proven that First fit is faster algo; while best fit provides better memory utilization.
- Memory info (physical base address and size) of each process is stored in its PCB and will be loaded into MMU registers (base & limit) during context switch.
- CPU request virtual address (address of the process) and is converted into physical address by MMU as shown in diag.
- If invalid virtual address is requested by the CPU, process will be terminated.
- If amount of memory required for a process is available but not contiguous, then it is called as "external fragmentation".
- To resolve this problem, processes in memory can be shifted/moved so that max contiguous free space will be available. This is called as "compaction".

Virtual Memory

- The portion of the hard disk which is used by OS as an extension of RAM, is called as "virtual memory".
- If sufficient RAM is not available to execute a new program or grow existing process, then some of the inactive process is shifted from main memory (RAM), so that new program can execute in RAM (or existing process can grow). It is also called as "swap area" or "swap space".

- Shifting a process from RAM to swap area is called as "swap out" and shifting a process from swap to RAM is called as "swap in".
- In few OS, swap area is created in form of a partition. E.g. UNIX, Linux, ...
- In few OS, swap area is created in form of a file E.g. Windows (pagefile.sys), ...
- Virtual memory advantages:
  - Can execute more number of programs.
  - Can execute bigger sized programs.

Segmentation

- Instead of allocating contiguous memory for the whole process, contiguous memory for each segment can be allocated. This scheme is known as "segmentation".
- Since process does not need contiguous memory for entire process, external fragmentation will be reduced.
- In this scheme, PCB is associated with a segment table which contains base and limit (size) of each segment of the process.
- During context switch these values will be loaded into MMU segment table.
- CPU request virtual address in form of segment address and offset address.
- Based on segment address appropriate base-limit pair from MMU is used to calculate physical address as shown in diag.
- MMU also contains STBR register which contains address of current process's segment table in the RAM.

Demand Segmentation

- If virtual memory concept is used along with segmentation scheme, in case low memory, OS may swap out a segment of inactive process.
- When that process again start executing and ask for same segment (swapped out), the segment will be loaded back in the RAM. This is called as "demand segmentation".
- Each entry of the segment table contains base & limit of a segment. It also contains additional bits like segment permissions, valid bit, dirty bit, etc
- If segment is present in main memory, its entry in seg table is said to be valid (v=1). If segment is swapped out, its entry in segment table is said to be invalid (v=0).

Paging

- RAM is divided into small equal sized partitions called as "frames" / "physical pages".
- Process is divided into small equal sized parts called as "pages" or "logical/virtual pages".
- page size = frame size.
- One page is allocated to one empty frame.
- OS keep track of free frames in form of a linked list.
- Each PCB is associated with a table storing mapping of page address to frame address. This table is called as "page table".
- During context switch this table is loaded into MMU.
- CPU requests a virtual address in form of page address and offset address. It will be converted into physical address as shown in diag.
- MMU also contains a PTBR, which keeps address of page table in RAM.

- If a page is not utilizing entire frame allocated to it (i.e. page contents are less than frame size), then it is called as "internal fragmentation".
- Frame size can be configured in the hardware. It can be 1KB, 2KB or 4KB, ...
- Typical Linux and Windows OS use page size = 4KB.

Page table entry

- Each PTE is of 32-bit (on x86 arch) and it contains
  - Frame address
  - Permissions (read or write)
  - Validity bit
  - Dirty bit
  - ...

TLB (Translation Look-Aside Buffer) Cache

- TLB is high-speed associative cache memory used for address translation in paging MMU.
- TLB has limited entries (e.g. in P6 arch TLB has 32 entries) storing recently translated page address and frame address mappings.
- The page address given by CPU, will be compared at once with all the entries in TLB and corresponding frame address is found.
- If frame address is found (TLB hit), then it is used to calculate actual physical address in RAM (as shown in diag).
- If frame address is not found (TLB miss), then PTBR is used to access actual page table of the process in the RAM (associated with PCB). Then page-frame address mapping is copied into TLB and thus physical address is calculated.
- If CPU requests for the same page again, its address will be found in the TLB and translation will be faster

Two Level Paging

- Primary page table has number of entries and each entry point to the secondary page table page.
- Secondary page table has number of entries and each entry point to the frame allocated for the process.
- Virtual address requested by a process is 32 bits including
  - p1 (10 bits) -> Primary page table index/addr
  - p2 (10 bits) -> Secondary page table index/addr
  - d (12 bits) -> Frame offset

Demand Paging

- When virtual memory is used with paging memory management, pages can be swapped out in case of low memory.
- The pages will be loaded into main memory, when they are requested by the CPU. This is called as "demand paging".
  - Swapped out pages
  - Pages from program image (executable file on disk)
  - Dynamically allocated pages

Virtual pages vs Logical pages

- By default all pages of user space process can be swapped out/in. This may change physical address of the page. All such pages whose physical address may change are referred as "Virtual pages".
- Few kernel pages are never swapped out. So their physical address remains same forever. All such pages whose physical address will not change are referred as "Logical pages".

Thrashing

- If number of programs are running in comparatively smaller RAM, a lot of system time will be spent into page swapping (paging) activity.
- Due to this overall system performance is reduced.
- The problem can be solved by increasing RAM size in the machine.

Page Fault

- Each page table entry contains frame address, permissions, dirty bit, valid bit, etc.
- If page is present in main memory its page table entry is valid (valid bit = 1).
- If page is not present in main memory, its page table entry is not valid (valid bit = 0).
- This is possible due to one of the following reasons:
  - Page address is not valid (dangling pointer).
  - Page is on disk/swapped out.
  - Page is not yet allocated.
- If CPU requests a page that is not present in main memory (i.e. page table entry valid bit=0), then "page fault" occurs.
- Then OS's page fault exception handler is invoked, which handles page faults as follows:
  1. Check virtual address due to which page fault occured. If it is not valid (i.e. dangling pointer), terminate the process (sending SEGV signal). (Validity fault).
  2. Check if read-write operation is permitted on the address. If not, terminate the process (sending SEGV signal). (Protection fault).
  3. If virtual address is valid (i.e. page is swapped out), then locate one empty frame in the RAM.
  4. If page is on swap device or hard disk, swap in the page in that frame.
  5. Update page table entry i.e. add new frame address and valid bit = 1 into PTE.
  6. Restart the instruction for which page fault occurred.

Page Replacement Algorithms

- While handling page fault if no empty frame found (step 3), then some page of any process need to be swapped out. This page is called as "victim" page.
- The algorithm used to decide the victim page is called as "page replacement algorithm".
- There are three important page replacement algorithms.
  - FIFO
  - Optimal
  - LRU

FIFO

- The page brought in memory first, will be swapped out first.

- Sometimes in this algorithm, if number of frames are increased, number of page faults also increase.
- This abnormal behaviour is called as "Belady's Anomaly".

OPTIMAL

- The page not required in near future is swapped out.
- This algorithm gives minimum number of page faults.
- This algorithm is not practically implementable.

LRU

- The page which not used for longer duration will be swapped out.
- This algorithm is used in most OS like Linux, Windows, ...
- LRU mechanism is implemented using "stack based approach" or "counter based approach".
- This makes algorithm implementation slower.
- Approximate LRU algorithm close to LRU, however is much faster.

Dirty Bit

- Each entry in page table has a dirty bit.
- When page is swapped in, dirty bit is set to 0.
- When write operation is performed on any page, its dirty bit is set to 1. It indicate that copy of the page in RAM differ from the copy in swap area.
- When such page need to be swapped out again, OS check its dirty bit. If bit=0 (page is not modified) actual disk IO is skipped and improves performance of paging operation.
- If bit=1 (page is modified), page is physically overwritten on its older copy in the swap area.

Computer structure

- CPU: Genral purpose processor for program/OS execution
- Memory: RAM
- Storage: Disk
- IO: Keyboard, Monitor
- Connected by "bus".
- Each IO device has a "dedicated" "internal" processing unit -- IO device controller.

Computer IO (Input Output)

- Synchronous IO: CPU is waiting for IO to complete.
  - Hw technique: Polling
- Asynchronous IO: CPU is not waiting for IO to complete (doing some other task)
  - Hw technique: Interrupts
  - OS maintains a device status table to keep track of IO devices (busy/idle) and processes waiting for those IO devices.

Interrupt Processing

- IO event is sensed by IO device controllers.
- It will be conveyed to CPU as a special signal - Interrupt.

- CPU pause current execution and execute interrupt handler.
- "Interrupt handler" will get address of "ISR" (from IVT) and execute ISR.
- When ISR is completed, execution resumes where it was paused.

Hardware vs Software interrupt

- Hardware -- interrupts from hardware peripherals.
- Software interrupt
  - Special instructions (Assembly/Machine level) when executed, current execution is paused, interrupt handler is executed and then the paused execution resumes.
  - Arch specific:
    - 8085/86: INT
    - ARM 7: SWI
    - ARM Cortex: SVC
  - Also called as "Trap" in few architecture.

Interrupt Controller

- Convey the interrupts from various peripherals to the CPU.
- Also manage priority of the interrupt (when multiple interrupts arrives at same time).
- e.g. 8085/86 <-- 8259, Modern x86 processors (apic), ARM-7 (VIC), ARM-CM3 (NVIC), ...

System Calls

- Software interrupt is used to implement OS/Kernel services.
- Functions exposed by the kernel so that user programs can access kernel functionalities, are called as "System calls".
  - e.g. Process Mgmt: create process, exit process, communication, synchronization, etc.
  - e.g. File Mgmt: create file, write file, read file, close file, etc.
  - e.g. Memory Mgmt: alloc memory, release memory, etc.
  - e.g. CPU Scheduling: Change process prioty, change process CPU affinity, etc.
- System calls are specific to the OS:
  - UNIX: 64 syscalls e.g. fork(), ..
  - Linux: 300+ syscalls e.g. fork(), clone(), ...
  - Windows: 3000+ syscalls e.g. CreateProcess(), ...

Types of kernel

Monolithic Kernel

- Multiple kernel source files are compiled into single kernel binary image. Such kernels are "monolithic" kernels.
- Since all functionalities present in single binary image, execution is faster.
- If any functionality fails at runtime, entire kernel may crash.
- Any modification in any component of OS, needs recompilation of the entire OS.
- Examples: BSD Unix, Windows (ntoskrnl.exe), Linux (vmlinuz), etc.

Micro-kernel

- Kernel is having minimal functionalities and remaining functionalities are implemented as independent processes called as "servers".
  - e.g. File management is done by a program called as "file server".
- These servers communicate with each other using IPC mechanism (message passing) and hence execution is little slower.
- If any component fails at runtime, only that process is terminated and rest kernel may keep functioning.
- Any modification in any component need to recompile only that component.
- Examples: Symbian, MACH, etc.

Modular Kernel

- Dynamically loadable modules (e.g. .dll / .so files) are loaded into calling process at runtime.
- In modular systems, kernel has minimal functionalities and rest of the functionalities are implemented as dynamically loadable modules.
- These modules get loaded into the kernel whenever they are called.
- As single kernel process is running, no need of IPC for the execution and thus improves performance of the system.
- Examples: Windows, Linux, etc.

Hybrid Kernel

- Mac OS X kernel is made by combination of two different kernels.
- BSD UNIX + MACH = Darwin

Linux - OS Structure

Linux components

- Linux kernel has static and dynamic components.
- Static components are
  - Scheduler
  - Process management
  - Memory management
  - IO subsystem (core)
  - System calls
- Dynamic components are
  - File systems (like ext3, ext4, FAT)
  - Device drivers
- Static components are compiled into the kernel binary image.
  - They are kernel components.
  - The kernel image is /boot/vmlinuz.
- Dynamic components are compiled into kernel objects (*.ko files).
  - They are non-kernel components.
  - They are located in /lib/modules/kernel-version.

Booting

- Process from computer power on to OS startup.

Bootable device

- Stoarage device (disk, pen drive, CD/DVD, etc.) whose boot block contains bootstrap program, is said to be Bootable device.

Bootstrap program

```
* Bootstrap program can load OS kernel into RAM and start its execution.
* Bootstrap program is different for each OS each version.
* Bootstrap is located in first sector (512 bytes) of a disk/partition.
```

Bootloader program

- Bootloader program can be configured to show multiple boot options to end user.
- Depending on user selection, it runs Bootstrap program of corresponding OS.
- There are many important bootloader programs.
  - ntldr (Before Windows Vista) --> boot.ini
  - "bootmgr" (Windows Vista Onwards) --> bcd (boot config data)
    - (admin command prompt) cmd> bcdedit
  - LiLo --> Linux Loader
  - "GrUB" (Grand Unified Boot Loader) --> menu.lst or grub.cfg
  - BTX (BooT eXtended) --> BSD Unix
  - SILo (Sparc Interactive Loader) --> Solaris
  - Bootcamp --> Mac OS X
  - "uBoot" --> Linux bootloader for Embedded
- The Bootloader has some config file associated with it.

Bootstrap loader

- The set of programs fixed in Base ROM (Motherboard) is called as "Firmware".
  - BIOS (Basic Input Output System) -- Firmware developed for PC by IBM + MS.
  - EFI (Extensible Firmware Interface) -- Firmware developed for PC by Intel + HP.
- Bootstrap loader is a program from Base ROM.
- It find the bootable device as per "boot device priority" in the BIOS setup.
- Once bootable device is found, it starts its bootloader.

Booting steps

- Computer power on.
- Load firmware (BIOS or EFI) programs from base ROM into RAM.
- Run POST/BIST (from firmware).
- Run Bootstrap loader (from firmware) to find bootable device.
- Bootstrap loader loads bootloader program from bootable device.
- Bootloader program shows multiple options to end user and user select one of them.
- Bootloader program run corresponding bootstrap program.
- Bootstrap program load OS kernel into main memory and OS boot