OS ctd..

Module- 5 Storage Management

* Memory:
- Memory is central to the operation of computing systems
- Memory = large array of words / bytes
- Each byte / word has its own address.
- Memory contains program to be executed and data both.
- A typical instruction-execution cycle:

(1) Instruction fetched from memory
(2) Instruction is decoded and may cause operand to be fetched from memory
(3) Instruction is executed on operands, and results are stored back in memory.

- CPU accesses only main memory and not processor registers.
- Instructions take main memory address as program arguments and not disk addresses.
- Data & instructions are thus in main memory.
- One of the speed parameters is no of CPU cycles taken to perform memory operation.
- CPU is faster than memory
- CPU registers are given access to one CPU clock cycle
- Main memory is accessed via transaction on memory bus. This can take many CPU clocks to complete
- Thus, CPU does not have data but has program for complete instruction execution. This causes CPU stalling.
- Data is always need for instruction execution.
- Add a faster memory between main memory and CPU also called cache memory. Cache memory is a buffer to accomodate speed difference.

- OS must be protected from unauthorised ~~proc~~ user process.
- Each user process must be protected from one another.
- Each process can legally access only the given range of addresses.
- Registers like base register and limit register are used ensure that a process accesses the addresses within givin Legal range.
- The OS executes only in kernel mode. It has unrestricted access to operating system memory and users memory.
- OS can load user's programmes into users memory and dump it out in case of fatal errors.

* Address Binding:
- Usually, programs reside on a disk as a binary executable file.
- To be executed, program must be bought to the memory.
- The program in disk ready to be executed are brought to the memory to form an input queue.
- The normal procedure is to select one process from the input queue and load it into memory.
- As the process is executed, it accesses instructions and data from memory.
- As the process terminates, its memory space is declared available.
- Addresses are represented in different ways in a program's life:

(1) Source code is available as a symbolic address.
(2) Compiler will ~~convert~~ bind it into a relocatable address.
(3) Linker or loader will bind relocatable address into absolute address.
(4) Binding maps ~~are addressed to server.~~ from one address space to another.

* Binding of instructions and data to memory:

- Instructions and data can be binded into memory at 3 different time

(1) Compile Time:

- Generates absolute code and it resides in memory.
- Recompiled if location changes.

(2) Load Time:

- Generates absolute code if known at load time and not compile time.
- Binding is delayed until load time.
- If starting address changes, reload user code and incorporate the changed value.

(3) Execution Time:

- If process is moved from one memory segment to another at execution time.
- No special hardware
- Binding is delayed.

* Logical address v/s Physical address:

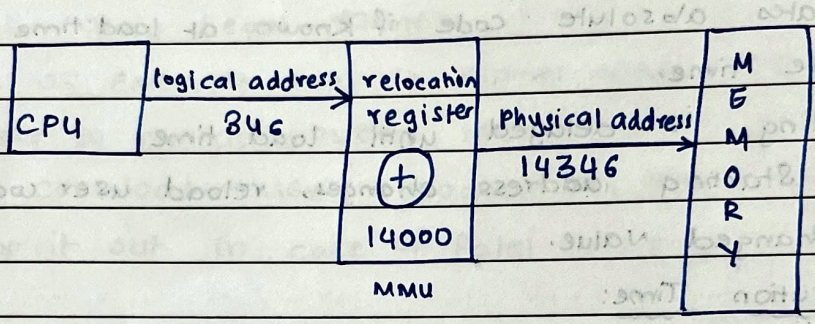
- Logical address:
  generated by CPU. Also called virtual address.

- Physical address:
  Generated by memory. It is loaded in the memory register of the memory.

- ~~load~~ logical address and physical address are same at compile time and load time. They are different at execution time and address binding time

* Memory Management Unit
- MMU maps virtual addresses to physical addresses

* Simple MMU: Generalisation of base register.
- Base register is also called relocation register
- While sending to memory, value of relocation register is added to every address generated by user process.



- User program never sees 'real' physical address.
- The user program only deals with logical address
- Range of logical address: 0 to max
- Range of physical address: R+0 to R+max where R is relocation register value.

* Dynamic loading:
- The entire program and all data of a process are present in physical memory to be executed.
- The size of process is only limited to size of physical memory
- To obtain better memory-space utilization we use dynamic loading.
- Routine is not loaded unless its called
- All routines are kept on disk in a relocatable load format.
- The program is loaded in the memory and executed.
- When a routine calls another routine, if the calling routine first checks if the routine has been loaded or not.
- If it is not, then the relocatable link loader calls the routine and loads it into the memory. It updates the program address table to reflect this change.
- Control is given to new routine.

* Advantages:
- No special support from OS is needed.
- OS may help programmer.
- Unused routine is never loaded.
- Useful when there is a large amount of code that needs to be handled in frequently occurring case
- Implemented through program design.

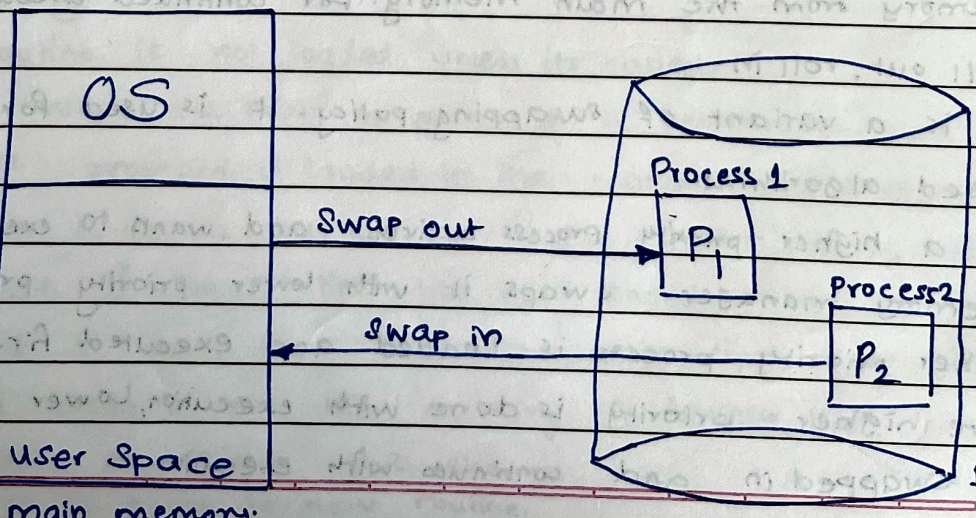
* Dynamic & Static Linking:
. Static Linking: System libraries and program code are combined by the linker/loader into an executable file.

. Dynamic Linking: System libraries are used with this feature. Without this feature, every program will have to create a copy of its language library in executable image. This leads to waste of disk space and main memory.
- This feature can be extended to library updates.
- Libraries may get replaced with a new version.
- All programs with reference to new library get linked automatically to new library. Without this feature, they will have to be relinked again and get access to updated version.

* Swapping
- Swapping is temporarily swapping of a process to a secondary memory from the main memory for continued execution
* Roll out, roll in-
- It is a variant of swapping policy. It is used for priority-based algorithms.
- If a higher-priority process arrives and wants to execute, the memory manager swaps it with lower priority process. The higher priority process is loaded and executed first. when the ~~lower~~ higher priority is done with execution, lower priority ~~process~~ is swapped in and continues with execution.

- Normally, a process that has swapped out is normaly swapped in back into the same memory space it previously occupied.
- Restriction is dictated by address binding.
- If binding is done at compiler time or load time, process cannot be easily moved to a different location.
- If binding is done at execution time then process can be swapped at into a different memory space as physical addresses are calculated at execution time.

* Dispatcher and swapper
- Swapping requires backing store. Backing store is a ~~device~~ fast disk large enough to store all copies of memory images for all users.
- It provides direct access to memory images
- System maintains ready queue of all processes whose memory images are in memory and are ready to run.
- When CPU decides to execute a process, it calls the dispatcher.
- Dispatcher checks whether next process in queue is in the memory or not.
- If not, and there is no free memory region, the swapper swaps out a process currently and swaps in the desired process.
- It reloads register and transfers control to that process.

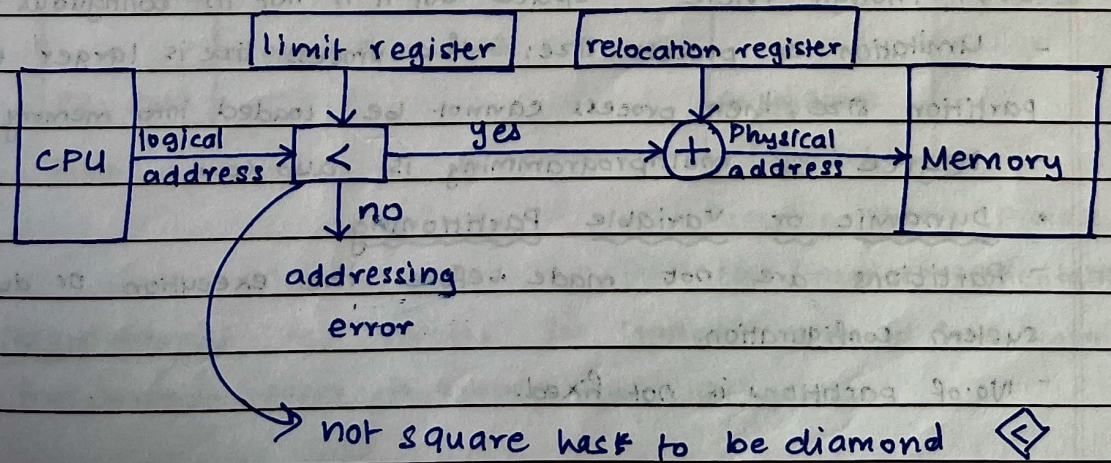


* Memory allocation to OS and user programs:

- Main memory accomodates OS and user processes.
- OS needs to allocate parts of main memory in the best way possible.
- Main memory is divided into 2 partitions:
  (1) Residing OS
  (2) User Process.
- Operating System occupies either low memory or high memory.
- This can be decided by input vector.
- The input vector lies in low memory. Thus, programmers prefer to have OS in low memory.
- User processes are kept in high memory.

* Memory mapping and protection:

- To protect user processes from each other and from changing OS code, and data using limit register and relocation/base register.
  (1) Relocation register contains value of smallest physical address.
  (2) Value of logical address range must be less than the limit register. If it is greater it throws an exception.
- MMU maps logical address dynamically by adding value of relocation register. Mapped address is sent to memory.



not square hash to be diamond

* Types of Memory management techniques:

① Contiguous memory allocation:

- In Continguous memory allocation, every process is contained in a single contiguous section of memory.
- The process is loaded entirely into the memory.
- Contiguous Technique can be of two types:

s • Fixed or Static Partitioning:

- In this partitioning, the no. of partitions in RAM is fixed but size of each partition is ~~not~~ may or may not be the same.

- Physical memory is divided into fixed size partitions. Each partition is assigned to one specific user.
- Partitions are made before or during execution.
- Each partition remains dedicated to the specific process unless it terminates or releases the partition.

~~* Disadv~~

* Disadvantages:

- It can lead to internal fragmentation, where memory in partition remains unused. This can happen when the process's memory requirements are smaller than partition size leaving some memory of partition unused.
- It can lead to external fragmentation, where total unused space of various partitions cannot be used to load the process even though there is space but it is not in contiguous form.
- Limitation on process size: If a process size is larger than partition size, then process cannot be loaded into memory.
- Degree of multiprogramming is also low.

• Dynamic or Variable Partitioning

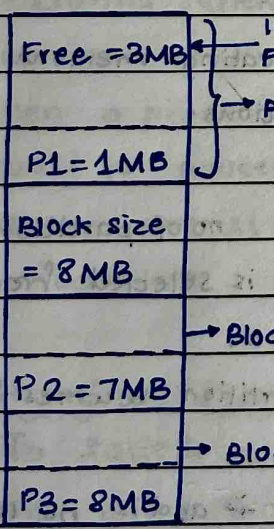
- Partitions are not made before the execution or during system configuration.
- No. of partitions is not fixed.

- Initially a RAM is empty and partitions are made during run time, according to the process's need instead of making partitions during system configuration.
- Size of partition will be equal to the size of the incoming process.
- Partition size varies as per process size to avoid internal fragmentation and ~~avoi~~ allow effective use of RAM.
- The number of partitions in RAM is not fixed and depends on no. of incoming process and main memory size.

+ Advantages:
- Better degree of multiprogramming
- No internal fragmentation.
- No limit on process size.

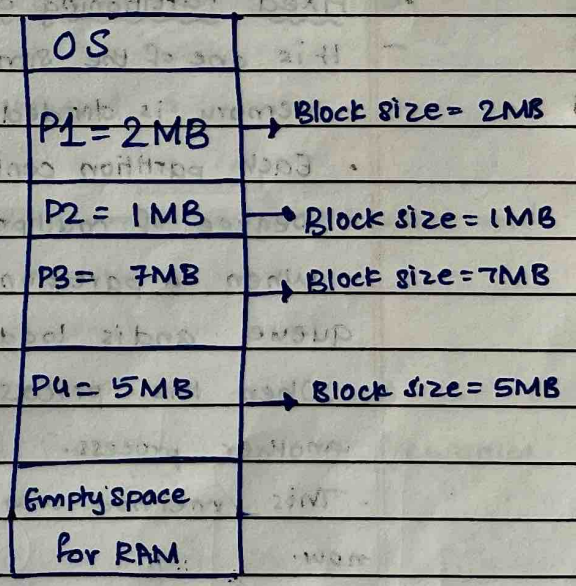
+ Disadvantages
- External fragmentation.

Static Partitioning Diagram:



~~Fixe~~

Dynamic Partitioning Diagram



Partition size = Process size
No internal fragmentation.

Fragmentation: As process is loaded / removed from memory, the free memory space is broken into little pieces

(2) Non-contiguous memory allocation:

- Each process is allocated non-contiguous blocks of memory that can be located anywhere in physical memory.
- Paging and Segmentation are two methods that allow non-contiguous allocation of an address space to memory. It avoids memory wastage but increases overheads due to address translation. It slows the execution of memory because time is consumed in address translation.

ctd in detail ahead... →

Contiguous Memory allocation ctd...

- ~~Multiple Partitioning:~~
- Fixed Partitioning ctd...
  - It is one of the simplest methods for allocating memory.
  - Memory is divided into fixed size partitions.
  - Each partition contains only one process.
  - Degree of multiprogramming is bound by no. of partitions.
  - When a partition gets free, a process is selected from input queue and is loaded in the free partition.
  - When the process terminates, the partition becomes free for another process.
  - This method was used by IBM os/360 and is no longer in use now.
  - As a process enters the system it is put in the input queue.
  - We have a list of available block sizes and the input queue.
  - The os can order input queue as per the scheduling algos.
  - Process is selected from input queue and is allocated memory large enough to accomodate it.

- OS can wait until a large block is available or it can skip down the input queue to see if the smaller memory requirements of other processes can be met.
- When a process arrives, the system searches for a hole large enough to fit the process. If hole is too large to fit the process, it is split into two parts: (1) One part allocated to arriving process (2) Second part is returned to other set of holes.
- Once the process terminates, it releases the holes and is placed back with the other set of holes.
- If the new hole is adjacent to the other set of holes, it combines with them to form one large hole
- The system checks if there are any processes waiting to enter the memory and whether this newly combined hole could satisfy the demands of any of these waiting processes.

* Dynamic Partitioning ctd...
- Operating System keeps list of allocated partitions and free partitions.
- When a process is arrived, it is allocated a partition size equal to process size.
- Once, process is terminated partition memory is released and it combines with adjacent free partitions.

* Dynamic Storage Allocation Problem:
- To satisfy a request of 'n' size from a list of free holes we use 3 algorithms:

(1) First Fit
(2) Best Fit
(3) Worst Fit
(4) Next Fit

(1) First Fit: Allocate the first hole that is big enough to fit the process.

Searching starts from beginning or from where the previous first-fit ended. Once the free hole is found we stop searching. ↳ that is large enough

* Advantage: ① Fastest search, since it searches only the first block that is enough to assign the process.

(2) ~~Next Fit:~~

* Disadvantage: Process cannot take space, even if it was possible to allocate

(2) Next Fit: Next Fit is a modification of first fit. It begins as a first fit to find a free partition. When it is called again next time it picks up from where it left off and not from the beginning.

(3) Best Fit: Allocate the smallest hole that is big enough. We must search entire list unless ordered by size. This strategy produces smallest left over hole.

Advantages: ① Improved memory utilization as it allocates the smallest block that can accomodate the memory request from process

② It also reduces memory fragmentation as it allocates smaller blocks of memory that tend to become fragmented.

Disadvantages: - It can also cause increased fragmentation as it can leave smaller memory blocks scattered throughout memory space. - Slow

(4) Worst Fit: Process travels the whole memory to find the largest hole / partition and process is placed in that partition.

- It is a slow process as it has to travel full memory to search for largest hole.
- We can get largest leftover hole which is more useful than

smaller left over hole from best fit approach.
- Both less external fragmentation

• Both first fit and best fit are better than worst fit in terms of decreasing time and storage utilization.
• Neither first fit nor best fit are better than other in terms of storage utilization but first fit is generally faster.

* Fragmentation:

(1) External Fragmentation: - Total memory exists to satisfy a request but it is not contiguous.
- Storage is fragmented into large number of small holes.
- In worst case: A block of free memory b/w every 2 processes.
If all these small pieces of memory were in one block instead, one several more processes could be executed.

(2) Internal Fragmentation: - Allocated memory is larger than requested.
- Unused memory is internal to partition.

- We can reduce external fragmentation by compaction.
- Shuffle memory contents to place all free memory together in one block.
- compaction is possible only for dynamic partitioning and is done at execution time.
- If addresses are relocated dynamically, relocation requires only moving program and data and changing the base register to reflect new address.
- Simplest method of compaction is:
(1) Move all processes to one end of memory.
(2) All holes in another direction.
(3) Produce one large hole of free memory.
(4) Expensive.

- Another way of compaction is:
  (1) Permit the logical address space of processes to be non-contiguous.
  (2) This allows a process to be allocated to physical memory wherever such memory is available.

(2) Non-contiguous Memory Allocation ctd...

* Paging:

- Paging is a memory management scheme that permits a ~~process~~ physical address of process to be non-contiguous.
- Process is allocated physical memory whenever it is available.
- This ~~causes~~ avoids external fragmentation and need for compaction. It also avoids problems of varied size chunks.
- Divide physical memory into fix-sized blocks called frames.
- Size is power of 2, between 512 bytes and 16 bytes.
- Divide logical memory into blocks of same sizes called pages.
- Keep track of all frames.
- To run a program of N pages, we need N free ~~pages~~ frames and load the program.
- Set up a page table to transfer logical to physical address.
- Backing store likewise split into pages.
- Still have internal fragmentation.

• Address Translation Scheme:

Address generated by CPU is divided into:

- Page Number (p): used as an index into page table.
- Page offset (d): combined with base address to define physical memory address to be sent to memory unit.
- Page Table contains base address of each page in physical memory.

0 1 1 0 0 1
16 8 4 2 1

DATE

- Page size is defined by hardware. It is usually in a power of 2.
- Selection of power of 2 makes translation of logical address to a page number and offset relatively easy.

If size of logical address space = 2^m and page size = 2^n addressing units,
then the higher order m-n bits of logical address will give page number and n low-order bits will give page offset.

See sum from ppt

- An important aspect of paging is to have clear separation between user's view of memory address and actual physical memory.
- User program views memory as one single space containing this only one program
- User program is scattered through memory which also holds other programs.
- Difference between user's view of memory and actual physical memory is reconciled by ~~hardware~~ address translation hardware.
- Mapping of logical address to physical address is hidden from the user and controlled by OS.

* Free Frame:

- When a process arrives, in the system to be executed, its size must be expressed in pages.
- Each page has one frame.
- If there are n pages, atleast n frames are needed
- If there are n frames, they are allocated to this process.
- First page is loaded into one of the frames.
- Frame number can be obtained from page table.

* Hardware Support

- Each OS has its own methods for storing page table.
- Mostly allocate a page table for each process.
- A pointer to the page table and other register values are stored in process control block. (PCB)
- When dispatcher is told to start a process, it must be reload user register and define correct hardware page-table values.

* Implementation of Page Table.

1. Page Table is implemented as a set of dedicated registers.
- Registers should be built with very high speed to make paging address-translation efficient.
- These registers are useful only if page table is small.
- Modern day computers allow 1million entries in the page table
- Hence, using fast registers is not feasible.

2. Page Table is kept in memory.
- PTBR (Page Table Base Register) points to current page table. Changing tables require changing this one register.
- PTLR (Page Table Length Register) indicates size of page table.

* TLB:

- To solve the two memory access problem caused due to paging, we use Translation look-aside buffer.
- TLB is special fast-lookup hardware cache,
- It is associative and has high-speed memory.
- Each entry has two parts: a key and a value.
- When memory is presented with an item, it compares with all keys simultaneously and if matching value is found it is returned. This is however fast but expensive.

- Total no of entries is small, numbering between 64 and 1024.
- when a logical address is generated by cpu, ~~it is~~ page number is presented to TLB.
- If page number found, TLB = Hit, frame number is available and used to access memory.
- If page number not found, then TLB = miss and reference should ~~be~~ be made to the page table. Once ~~page~~ frame number is obtained we use it to access the memory.
- In addition we add the page number and frame number to the TLB so that it ~~the it~~ will be found quickly on the next reference.

see flowchart

- If TLB is already full, then OS must replace one of the entries.
- Replacement policies like least recently used can be applied.

* TLB for context switching
- Some TLBs store ASID (Address-Space identifiers in each TLB entry
- ASID identifies each process and provides address space protection for that process.
- ASID provides a TLB to contain entries of several different processes simultaneously.

* Effective Access Time
- Hit ratio: % of times the particular page number is found in TLB.
- TLB Hit: If it takes 20 nanoseconds to search TLB and 100 nanosecond to access memory, then mapped memory access take 120 nanoseconds.

- TLB Miss: If we fail to find page number then 6 20 nanoseconds used there, then we must access the page table and search for frame number (100 nanosecond) and then access the memory (100 nanosecond) for a total for 220 nanosecond.

Effective Access Time with :

80 % Hit ratio

$EAT = 0.80 \times 120 + 0.20 \times 220$

$= 140$ nanosecond.

98 % Hit ratio

$EAT = 0.98 \times 120 + 0.02 \times 220$

$= 122$ nanosecond.

* Memory Protection, valid invalid bit & Structure of Page Table see from ppt: CD0 132 onwards till end from ppt)

Structure of Page Table

① Hierarchical Paging:

- Breaks up logical address into multiple page tables.
- Simple technique is Two-level page table.
- We page a page table.
- A logical address is divided into:
  - → Page Number
  - → Page offset
- Since page number is paged, we can further divide it into

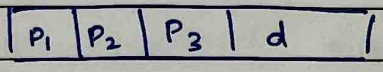
P_1	P_2	d

P_1 : index to outer page table

P_2: page of outer page table

eg: Look at 64-bit Logical address space.

- Similar way for 3-level paging scheme.



① Hash page Table

③ Inverted Page Table

* Segmentation: do from ppt.