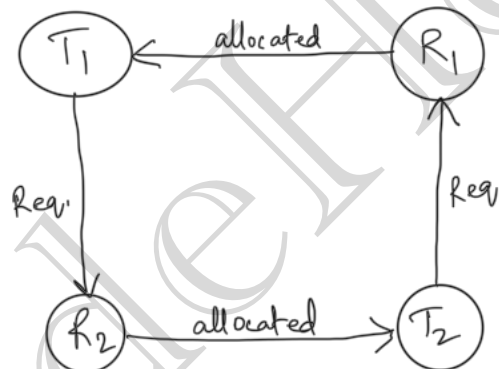
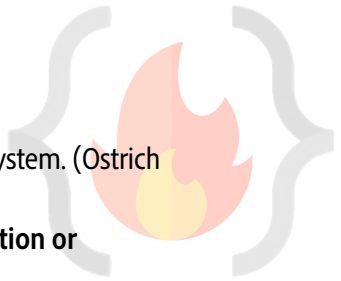


Deadlock Part-1

1. In Multi-programming environment, we have several processes competing for finite number of resources
2. Process requests a **resource (R)**, if R is not available (taken by other process), process enters in a waiting state. Sometimes that waiting process is never able to change its state because the resource, it has requested is busy (forever), called **DEADLOCK (DL)**
3. Two or more processes are waiting on some resource's availability, which will never be available as it is also busy with some other process. The Processes are said to be in **Deadlock**.
4. DL is a bug present in the process/thread synchronization method.
5. In DL, processes never finish executing, and the system resources are tied up, preventing other jobs from starting.
6. **Example of resources:** Memory space, CPU cycles, files, locks, sockets, IO devices etc.
7. Single resource can have multiple instances of that. E.g., CPU is a resource, and a system can have 2 CPUs.
8. How a process/thread utilize a resource?
 - a. Request: Request the R, if R is free Lock it, else wait till it is available.
 - b. Use
 - c. Release: Release resource instance and make it available for other processes



9. **Deadlock Necessary Condition:** 4 Condition should hold simultaneously.
 - a. **Mutual Exclusion**
 - i. Only 1 process at a time can use the resource, if another process requests that resource, the requesting process must wait until the resource has been released.
 - b. **Hold & Wait**
 - i. A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes.
 - c. **No-preemption**
 - i. Resource must be voluntarily released by the process after completion of execution. (No resource preemption)
 - d. **Circular wait**
 - i. A set $\{P_0, P_1, \dots, P_n\}$ of waiting processes must exist such that P_0 is waiting for a resource held by P_1 , P_1 is waiting for a resource held by P_2 , and so on.
10. **Methods for handling Deadlocks:**
 - a. Use a protocol to **prevent** or **avoid** deadlocks, ensuring that the system will never enter a deadlocked state.
 - b. Allow the system to enter a deadlocked state, **detect it, and recover**.

- 
- c. Ignore the problem altogether and pretend that deadlocks never occur in system. (Ostrich algorithm) aka, **Deadlock ignorance**.
 - 11. To ensure that deadlocks never occur, the system can use either a **deadlock prevention or deadlock avoidance scheme**.
 - 12. **Deadlock Prevention**: by ensuring at least one of the necessary conditions cannot hold.
 - a. **Mutual exclusion**
 - i. Use locks (mutual exclusion) only for non-sharable resource.
 - ii. Sharable resources like Read-Only files can be accessed by multiple processes/threads.
 - iii. However, we can't prevent DLs by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.
 - b. **Hold & Wait**
 - i. To ensure H&W condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it doesn't hold any other resource.
 - ii. Protocol (A) can be, each process has to request and be allocated all its resources before its execution.
 - iii. Protocol (B) can be, allow a process to request resources only when it has none. It can request any additional resources after it must have released all the resources that it is currently allocated.
 - c. **No preemption**
 - i. If a process is holding some resources and request another resource that cannot be immediately allocated to it, then all the resources the process is currently holding are preempted. The process will restart only when it can regain its old resources, as well as the new one that it is requesting. (Live Lock may occur).
 - ii. If a process requests some resources, we first check whether they are available. If yes, we allocate them. If not, we check whether they are allocated to some other process that is waiting for additional resources. If so, preempt the desired resource from waiting process and allocate them to the requesting process.
 - d. **Circular wait**
 - i. To ensure that this condition never holds is to impose a proper ordering of resource allocation.
 - ii. P1 and P2 both require R1 and R1, locking on these resources should be like, both try to lock R1 then R2. By this way which ever process first locks R1 will get R2.

Deadlock Part-2



1. **Deadlock Avoidance:** Idea is, the kernel be given in advance info concerning which resources will use in its lifetime.

By this, system can decide for each request whether the process should wait.

To decide whether the current request can be satisfied or delayed, the system must consider the resources currently available, resources currently allocated to each process in the system and the future requests and releases of each process.

- a. Schedule process and its resources allocation in such a way that the DL never occur.
- b. Safe state: A state is safe if the system can allocate resources to each process (up to its max.) in some order and still avoid DL.
A system is in safe state only if there exists a safe sequence.
- c. In an Unsafe state, the operating system cannot prevent processes from requesting resources in such a way that any deadlock occurs. It is not necessary that all unsafe states are deadlocks; an unsafe state may lead to a deadlock.
- d. The main key of the deadlock avoidance method is whenever the request is made for resources then the request must only be approved only in the case if the resulting state is a safe state.
- e. In a case, if the system is unable to fulfill the request of all processes, then the state of the system is called unsafe.
- f. Scheduling algorithm using which DL can be avoided by finding safe state. (Banker Algorithm)

2. Banker Algorithm

- a. When a process requests a set of resources, the system must determine whether allocating these resources will leave the system in a safe state. If yes, then the resources may be allocated to the process. If not, then the process must wait till other processes release enough resources.

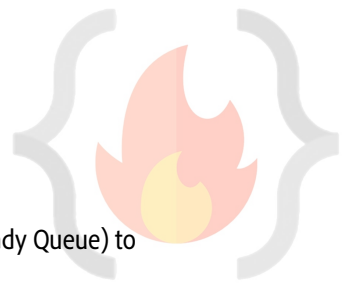
3. **Deadlock Detection:** Systems haven't implemented deadlock-prevention or a deadlock avoidance technique, then they may employ DL detection then, recovery technique.

- a. Single Instance of Each resource type (**wait-for graph method**)
 - i. A deadlock exists in the system if and only if there is a cycle in the wait-for graph. In order to detect the deadlock, the system needs to maintain the wait-for graph and periodically system invokes an algorithm that searches for the cycle in the wait-for graph.
- b. Multiple instances for each resource type
 - i. Banker Algorithm

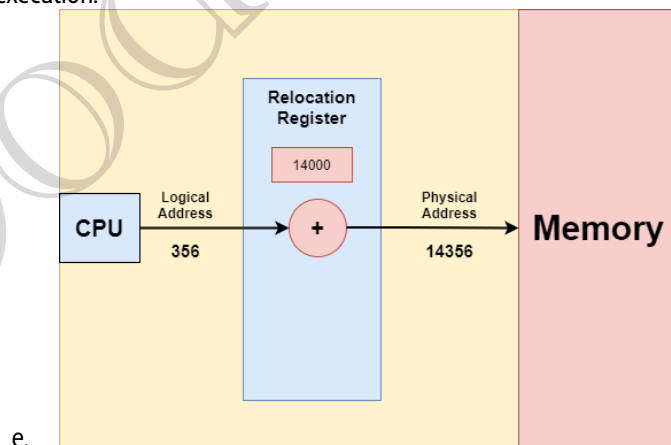
4. Recovery from Deadlock

- a. Process termination
 - i. Abort all DL processes
 - ii. Abort one process at a time until DL cycle is eliminated.
- b. Resource preemption
 - i. To eliminate DL, we successively preempt some resources from processes and give these resources to other processes until DL cycle is broken.

Memory Management Techniques | Contiguous Memory Allocation

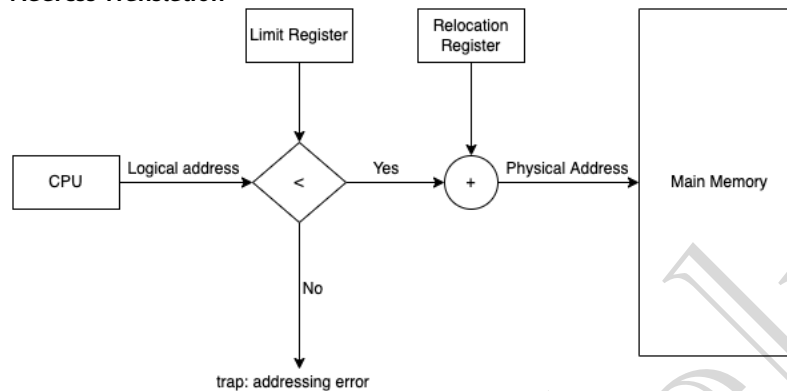


1. In Multi-programming environment, we have multiple processes in the main memory (Ready Queue) to keep the CPU utilization high and to make computer responsive to the users.
2. To realize this increase in performance, however, we must keep several processes in the memory; that is, we must **share** the main memory. As a result, we must **manage** main memory for all the different processes.
3. **Logical versus Physical Address Space**
 - a. Logical Address
 - i. An address generated by the CPU.
 - ii. The logical address is basically the address of an instruction or data used by a process.
 - iii. User can access logical address of the process.
 - iv. User has indirect access to the physical address through logical address.
 - v. Logical address does not exist physically. Hence, aka, **Virtual address**.
 - vi. The set of all logical addresses that are generated by any program is referred to as Logical Address Space.
 - vii. **Range: 0 to max.**
 - b. Physical Address
 - i. An address loaded into the memory-address register of the physical memory.
 - ii. User can never access the physical address of the Program.
 - iii. The physical address is in the memory unit. It's a location in the main memory physically.
 - iv. A physical address can be accessed by a user indirectly but not directly.
 - v. The set of all physical addresses corresponding to the Logical addresses is commonly known as Physical Address Space.
 - vi. It is computed by the **Memory Management Unit (MMU)**.
 - vii. **Range: (R + 0) to (R + max), for a base value R.**
 - c. **The runtime mapping from virtual to physical address is done by a hardware device called the memory-management unit (MMU).**
 - d. The user's program mainly generates the logical address, and the user thinks that the program is running in this logical address, but the program mainly needs physical memory in order to complete its execution.



4. How OS manages the isolation and protect? (**Memory Mapping and Protection**)
 - a. OS provides this Virtual Address Space (VAS) concept.
 - b. To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
 - c. The relocation register contains value of smallest physical address (Base address [R]); the limit register contains the range of logical addresses (e.g., relocation = 100040 & limit = 74600).
 - d. Each logical address must be less than the limit register.

- e. MMU maps the logical address dynamically by adding the value in the relocation register.
- f. When CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Since every address generated by the CPU (Logical address) is checked against these registers, we can protect both OS and other users' programs and data from being modified by running process.
- g. Any attempt by a program executing in user mode to access the OS memory or other users' memory results in a trap in the OS, which treat the attempt as a fatal error.
- h. **Address Translation**

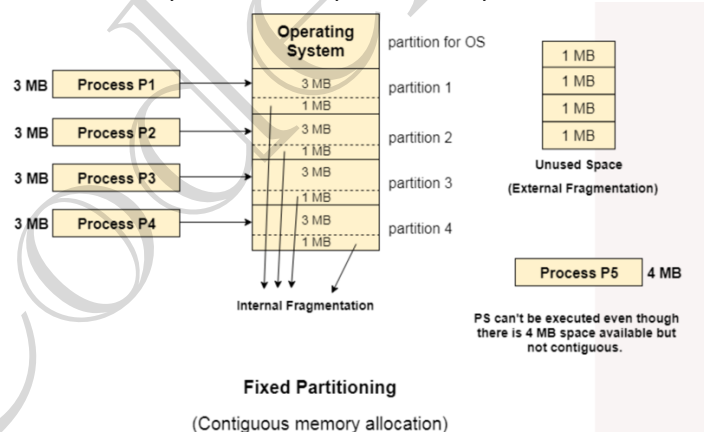


5. Allocation Method on Physical Memory

- a. Contiguous Allocation
- b. Non-contiguous Allocation

6. Contiguous Memory Allocation

- a. In this scheme, each process is contained in a single contiguous block of memory.
- b. **Fixed Partitioning**
 - i. The main memory is divided into partitions of equal or different sizes.



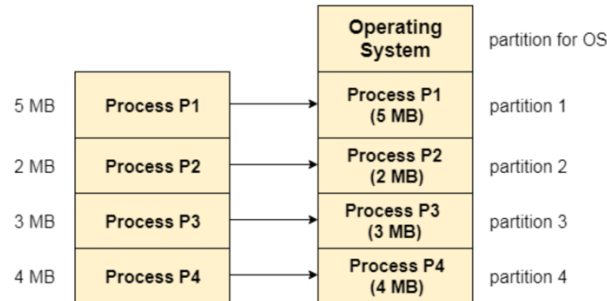
- ii.
- iii. Limitations:

1. **Internal Fragmentation:** if the size of the process is lesser than the total size of the partition then some size of the partition gets wasted and remain unused. This is wastage of the memory and called internal fragmentation.
2. **External Fragmentation:** The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.
3. **Limitation on process size:** If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Low degree of multi-programming: In fixed partitioning, the degree of multiprogramming is fixed and very less because the size of the partition cannot be varied according to the size of processes.

c. Dynamic Partitioning

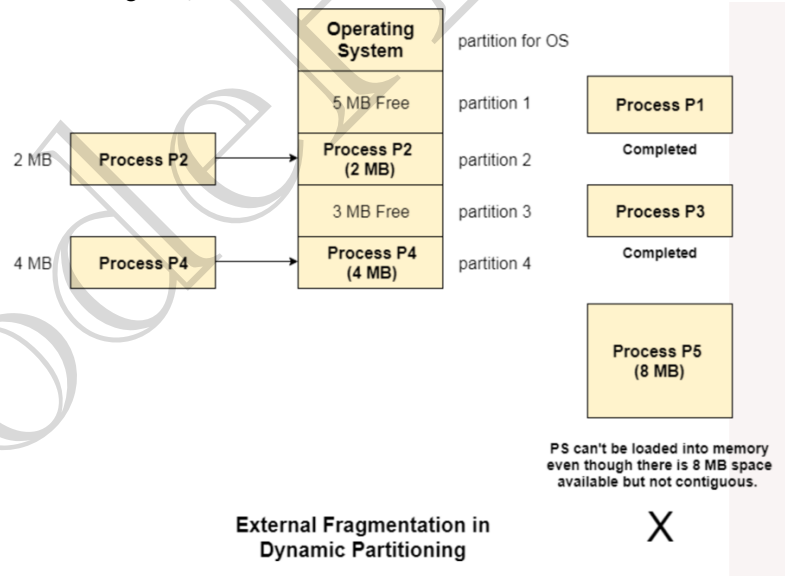
- i. In this technique, the partition size is not declared initially. It is declared at the time of process loading.



Dynamic Partitioning

(Process Size = Partition Size)

- ii.
- iii. Advantages over fixed partitioning
 1. No internal fragmentation
 2. No limit on size of process
 3. Better degree of multi-programming
- iv. Limitation
 1. External fragmentation



External Fragmentation in Dynamic Partitioning



Free Space Management

1. Defragmentation/Compaction

- a. Dynamic partitioning suffers from external fragmentation.
- b. Compaction to minimize the probability of external fragmentation.
- c. All the free partitions are made contiguous, and all the loaded partitions are brought together.
- d. By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called **defragmentation**.
- e. The efficiency of the system is decreased in the case of compaction since all the free spaces will be transferred from several places to a single place.

2. How free space is stored/represented in OS?

- a. Free holes in the memory are represented by a free list (Linked-List data structure).

3. How to satisfy a request of a of n size from a list of free holes?

- a. Various algorithms which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.

b. First Fit

- i. Allocate the first hole that is big enough.
- ii. Simple and easy to implement.
- iii. Fast/Less time complexity

c. Next Fit

- i. Enhancement on First fit but starts search always from last allocated hole.
- ii. Same advantages of First Fit.

d. Best Fit

- i. Allocate smallest hole that is big enough.
- ii. Lesser internal fragmentation.
- iii. May create many small holes and cause major external fragmentation.
- iv. Slow, as required to iterate whole free holes list.

e. Worst Fit

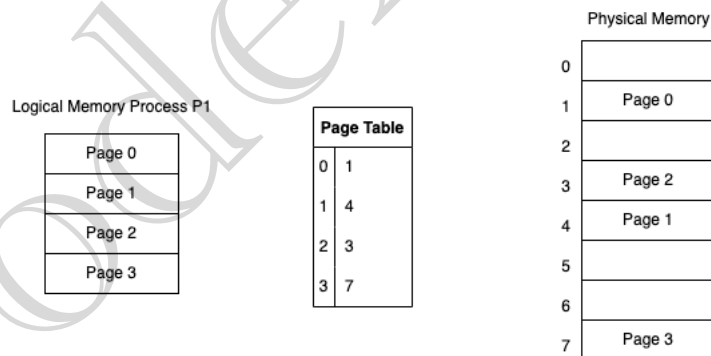
- i. Allocate the largest hole that is big enough.
- ii. Slow, as required to iterate whole free holes list.
- iii. Leaves larger holes that may accommodate other processes.



Paging | Non-Contiguous Memory Allocation

1. The main **disadvantage of Dynamic partitioning is External Fragmentation.**
 - a. Can be removed by Compaction, but with overhead.
 - b. **We need more dynamic/flexible/optimal mechanism, to load processes in the partitions.**
2. **Idea behind Paging**
 - a. If we have only two small non-contiguous free holes in the memory, say 1KB each.
 - b. If OS wants to allocate RAM to a process of 2KB, in contiguous allocation, it is not possible, as we must have contiguous memory space available of 2KB. (External Fragmentation)
 - c. **What if we divide the process into 1KB-1KB blocks?**
3. **Paging**
 - a. **Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.**
 - b. It avoids external fragmentation and the need of compaction.
 - c. Idea is to divide the physical memory into fixed-sized blocks called **Frames**, along with divide logical memory into blocks of same size called **Pages**. (# Page size = Frame size)
 - d. **Page size** is usually determined by the processor architecture. Traditionally, pages in a system had uniform size, such as 4,096 bytes. However, processor designs often allow two or more, sometimes simultaneous, page sizes due to its benefits.
 - e. **Page Table**
 - i. A Data structure stores which page is mapped to which frame.
 - ii. **The page table contains the base address of each page in the physical memory.**
 - f. Every address generated by CPU (logical address) is divided into two parts: a page number (p) and a page offset (d). The p is used as an index into a page table to get base address the corresponding frame in physical memory.

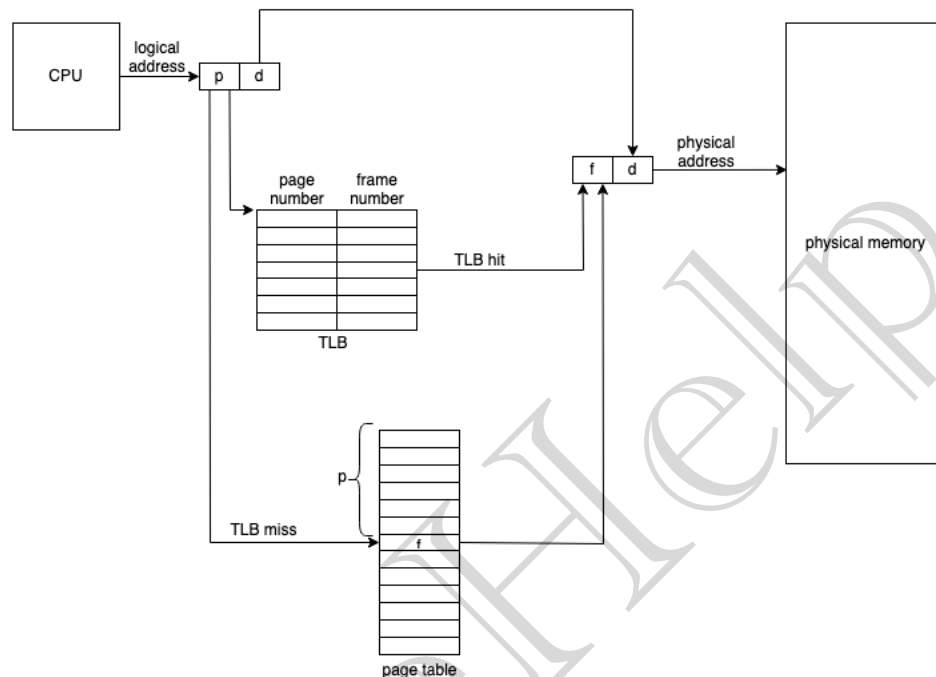
Paging model of logical and physical memory



- g. Page table is stored in main memory at the time of process creation and its base address is stored in process control block (PCB).
 - h. A page table base register (PTBR) is present in the system that points to the current page table. Changing page tables requires only this one register, at the time of context-switching.
4. **How Paging avoids external fragmentation?**
 - a. Non-contiguous allocation of the pages of the process is allowed in the random free frames of the physical memory.
5. **Why paging is slow and how do we make it fast?**
 - a. There are too many memory references to access the desired location in physical memory.
6. **Translation Look-aside buffer (TLB)**
 - a. A Hardware support to speed-up paging process.
 - b. It's a hardware cache, high speed memory.
 - c. TBL has key and value.

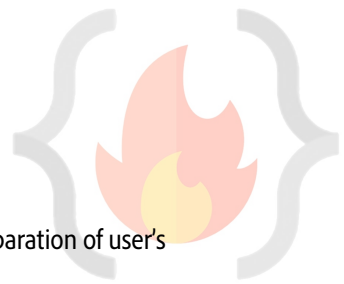
- d. Page table is stores in main memory & because of this when the memory references is made the translation is slow.
- e. When we are retrieving physical address using page table, after getting frame address corresponding to the page number, we put an entry of the into the TLB. So that next time, we can get the values from TLB directly without referencing actual page table. Hence, make paging process faster.

Paging hardware with TLB

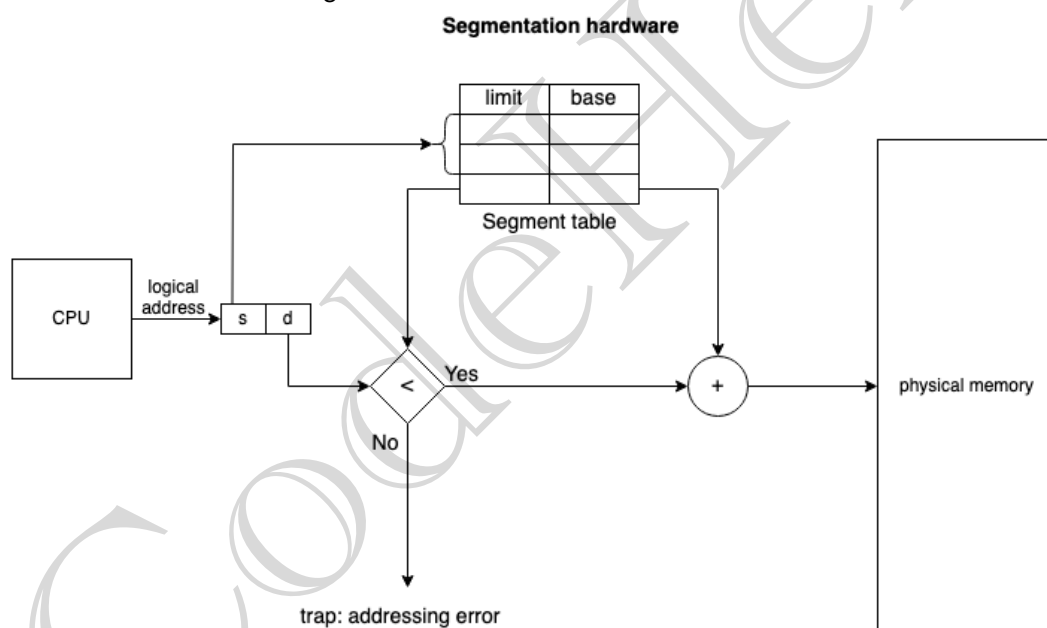


- f. **TLB hit, TLB contains the mapping for the requested logical address.**
- g. **Address space identifier (ASIDs)** is stored in each entry of TLB. ASID uniquely identifies each process and is used to **provide address space protection and allow to TLB to contain entries for several different processes**. When TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently executing process matches the ASID associated with virtual page. If it doesn't match, the attempt is treated as TLB miss.

Segmentation | Non-Contiguous Memory Allocation



1. An important aspect of memory management that become unavoidable with paging is separation of user's view of memory from the actual physical memory.
2. Segmentation is memory management technique that supports the **user view of memory**.
3. A logical address space is a collection of segments, these segments are based on **user view** of logical memory.
4. Each segment has **segment number and offset**, defining a segment.
<segment-number, offset> {s,d}
5. Process is divided into **variable segments based on user view**.
6. **Paging** is closer to the Operating system rather than the **User**. It divides all the processes into the form of pages although a process can have some relative parts of functions which need to be loaded in the same page.
7. Operating system doesn't care about the **User's view** of the process. It may **divide the same function into different pages** and those **pages may or may not be loaded at the same time into the memory**. It decreases the efficiency of the system.
8. It is better to have segmentation which divides the process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.



- 9.
10. **Advantages:**
 - a. No internal fragmentation.
 - b. One segment has a contiguous allocation, hence efficient working within segment.
 - c. The size of segment table is generally less than the size of page table.
 - d. It results in a more efficient system because the compiler keeps the same type of functions in one segment.
11. **Disadvantages:**
 - a. External fragmentation.
 - b. The different size of segment is not good that the time of swapping.
12. Modern System architecture provides both segmentation and paging implemented in some hybrid approach.