Text, logo

Description automatically generated **San Francisco Bay University**

**CE521 - Real-time Systems and Programming**

**Homework Assignment #6**

**Due day: 4/17/2022**

**Student ID: 19590**

**Instructions:**

1. **Push the answer sheet to Github**
2. **Overdue homework submission could not be accepted.**
3. **Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**
4. Explain the concepts in the steps of the processing of programs: compile time/load time/execution time

Program is a sequence of instructions written by the user that instructs the computer to perform the task of solving some problem. The program resides on the disk as a binary executable file. To run the program, it must be brought from the disk into the main memory.

As the program to be executed is brought from the disk to the main memory, it is placed within the context of a process, where it becomes available for execution on the CPU. During the execution, it accesses data and instructions from the memory and once the execution is completed, the process is terminated, and the memory is reclaimed for use by another process.

The addresses in the source code are generally symbolic. The compiler binds these addresses to relocatable addresses, and these are bonded by the linker or loader to the absolute addresses.

The binding of instructions and data to the memory addresses can be done at any of the following steps of program execution:

1. [**Compile Time**](https://www.geeksforgeeks.org/difference-between-compile-time-and-load-time-address-binding/)**:**   
   If we initially know where the process is residing in the memory at the time of compilation, then absolute code can be generated. If the starting location changes, then the code must be recompiled.

1. [**Load Time**](https://www.geeksforgeeks.org/difference-between-compile-time-and-load-time-address-binding/)**:**   
   If the memory address where the process resides is not known at the compile time, then the compiler must generate relocatable code. If the starting address changes, then the program must be reloaded to incorporate this value.

1. [**Execution Time**](https://www.geeksforgeeks.org/difference-between-compile-time-and-execution-time-address-binding/)**:**   
   If the process can be moved from one segment to another during its execution time, then binding must be delayed till execution time. Special hardware is required for this type of binding.

Diagram

Description automatically generated

1. Explain why the virtual memory is commonly built in the modern computing system

Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory. In Virtual memory, the user can store processes with a bigger size than the available main memory.

We know that a process is divided into various pages and these pages are used during the execution of the process. The whole process is stored in the secondary memory. But to make the execution of a process faster, we use the main memory of the system and store the process pages into it. But there is a limitation with the main memory. We have limited space and less space in the main memory. So, if the size of the process is larger than the size of the main memory Virtual Memory comes into play.

Virtual Memory is commonly built-in modern computing system because:

* Whenever your computer doesn’t have space in the physical memory it writes what it needs to remember to the hard disk in a swap file as virtual memory.
* If a computer running Windows needs more memory/RAM, then installed in the system, it uses a small portion of the hard drive for this purpose.
* Benefit of using the Virtual Memory is that if we are having some program that is larger than the size of the main memory then instead of loading all the pages we load some important pages.
* Virtual memory helps to gain speed when only a particular segment of the program is required for the execution of the program.
* It is very helpful in implementing a multiprogramming environment.
* It allows you to run more applications at once.
* It helps you to fit many large programs into smaller programs.
* Common data or code may be shared between memory.
* Process may become even larger than all of the physical memory.
* Data / code should be read from disk whenever required.
* The code can be placed anywhere in physical memory without requiring relocation.
* More processes should be maintained in the main memory, which increases the effective use of CPU.
* Each page is stored on a disk until it is required after that, it will be removed.
* It allows more applications to be run at the same time.
* There is no specific limit on the degree of multiprogramming.
* Large programs should be written, as virtual address space available is more compared to physical memory.

1. Describe how many methods can be taken to map virtual address to physical address, and compare the pros/cons for each

**Mapping Virtual Addresses to Physical Addresses:**  
In Contiguous memory allocation mapping from virtual addresses to physical addresses is not a difficult task, because if we take a process from secondary memory and copy it to the main memory, the addresses will be stored in a contiguous manner, so if we know the base address of the process, we can find out the next addresses.

**1) Base and bound:**

The Memory Management Unit is a combination of 2 registers:

1. Base Register (Relocation Register)
2. Limit Register.

**Base Register –**contains the starting physical address of the process.  
**Limit Register** -mentions the limit relative to the base address on the region occupied by the process.

The logical address generated by the CPU is first checked by the limit register, If the value of the logical address generated is less than the value of the limit register, the base address stored in the relocation register is added to the logical address to get the physical address of the memory location.  
If the logical address value is greater than the limit register, then the CPU traps to the OS, and the OS terminates the program by giving fatal error.

Diagram

Description automatically generated

In Non-Contiguous Memory allocation, processes can be allocated anywhere in available space. The address translation in non-contiguous memory allocation is difficult.  
There are several techniques used for address translation in non-contiguous memory allocation like Paging, Multilevel paging, Inverted paging, Segmentation, Segmented paging. Different data structures and hardware support like TLB are required in these techniques.

**Pros:**

• OS protection and program isolation  
• Low overhead address translation

**Cons:**  
• Expandable heap Issue   
• Expandable stack Issue  
• Memory sharing between processes -overwriting  
• Non-relative addresses – hard to move memory around Memory fragmentation

**Paging:**

* Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.
* Logical Address or Virtual Address: An address generated by the CPU
* Logical Address Space or Virtual Address Space: The set of all logical addresses generated by a program
* Physical Address: An address available on memory unit
* Physical Address Space: The set of all physical addresses corresponding to the logical addresses

**Pros:**

* Paging reduces external fragmentation, but still suffer from internal fragmentation.
* Paging is simple to implement and assumed as an efficient memory management technique.
* Due to equal size of the pages and frames, swapping becomes very easy.

**Cons:**

* Page table requires extra memory space, so may not be good for a system having small RAM.

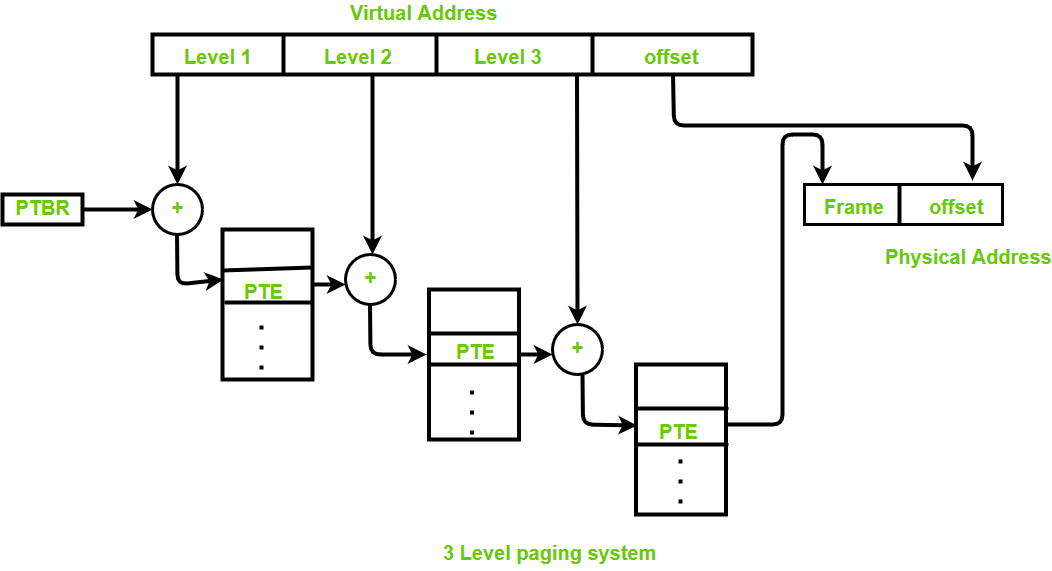
**Multilevel Paging:**

**Multilevel Paging** is a paging scheme which consist of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2-page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table are storing actual frame information. Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).

Virtual address:



In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory. So, it requires more than one memory access to get the physical address of page frame. One access for each level needed. Each page table entry **except** the last level page table entry contains base address of the next level page table.



**Cons:**

* Extra memory references to access address translation tables can slow programs down by a factor of two or more. Use translation look aside buffer (TLB) to speed up address translation by storing page table entries.

**Inverted Page Table:**

Most of the Operating Systems implement a separate page table for each process, i.e. for ‘n’ number of processes running on a Multiprocessing/ Timesharing operating system, there are ‘n’ number of page tables stored in the memory. Sometimes when a process is very large in size and it occupies virtual memory then with the size of the process, it’s page table size also increases substantially.

Through the inverted page table, the overhead of storing an individual page table for every process gets eliminated and only a fixed portion of memory is required to store the paging information of all the processes together. This technique is called as inverted paging as the indexing is done with respect to the frame number instead of the logical page number. Each entry in the page table contains the following fields.

Page number – It specifies the page number range of the logical address.

Process id – An inverted page table contains the address space information of all the processes in execution. Since two different processes can have similar set of virtual addresses, it becomes necessary in Inverted Page Table to store a process Id of each process to identify it’s address space uniquely. This is done by using the combination of PId and Page Number. So, this Process Id acts as an address space identifier and ensures that a virtual page for a particular process is mapped correctly to the corresponding physical frame.

Control bits – These bits are used to store extra paging-related information. These include the valid bit, dirty bit, reference bits, protection and locking information bits.

Chained pointer – It may be possible sometime that two or more processes share a part of main memory. In this case, two or more logical pages map to same Page Table Entry then a chaining pointer is used to map the details of these logical pages to the root page table.

**Pros:**

* Reduced memory space

**Cons:**

* Longer lookup time
* Difficult shared memory implementation

**Segmentation:**

A process is divided into Segments. The chunks that a program is divided into which are not necessarily all the same sizes are called segments. Segmentation gives user’s view of the process which paging does not give. Here the user’s view is mapped to physical memory.

There are types of segmentation:

Virtual memory segmentation: Each process is divided into several segments, not all of which are resident at any one point in time.

Simple segmentation: Each process is divided into several segments, all of which are loaded into memory at run time, though not necessarily contiguously.

There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

Segment Table – It maps two-dimensional Logical address into one-dimensional Physical address. It’s each table entry has:

Base Address: It contains the starting physical address where the segments reside in memory.

Limit: It specifies the length of the segment.

**Pros:**

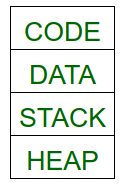
* No Internal fragmentation.
* Segment Table consumes less space in comparison to Page table in paging.

**Cons:**

* As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

**Segmented Paging:**

* A solution to the problem is to use segmentation along with paging to reduce the size of page table. Traditionally, a program is divided into four segments, namely code segment, data segment, stack segment and heap segment.



* The size of the page table can be reduced by creating a page table for each segment. To accomplish this hardware support is required. The address provided by CPU will now be partitioned into segment no., page no. and offset.

**Pros:**

* The page table size is reduced as pages are present only for data of segments, hence reducing the memory requirements.
* Gives a programmers view along with the advantages of paging.
* Reduces external fragmentation in comparison with segmentation.
* Since the entire segment need not be swapped out, the swapping out into virtual memory becomes easier.

**Cons:**

* Internal fragmentation still exists in pages.
* Extra hardware is required
* Translation becomes more sequential increasing the memory access time.
* External fragmentation occurs because of varying sizes of page tables and varying sizes of segment tables in today’s systems.