This Method has two steps:

i) Detection

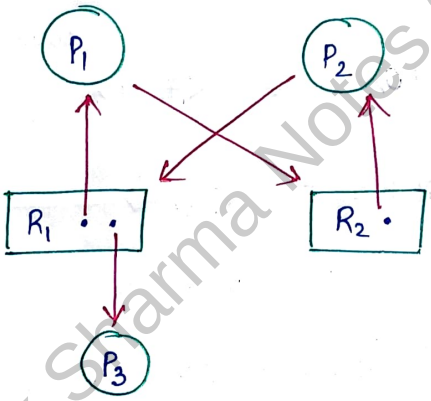
ii) Recovery.

Q) How to Detect a Deadlock ?

A:- If the Resourse Allocation Graph has a cycle and all the Resourses have single instences then there is a Deadlock.

* If there are multiple instences in a Resourse then it will not be a deadlock.

Example:



Here it is not a deadlock though all the Processes are in a cycle But R_1 has multiple instances (2 instances). So That's why it is not a deadlock condition.

* use same Bankers Algorithm to Detect if a System has multiple instances in a Resourse.

Q) How can we Recover from a Deadlock ?

A:- Deadlock Recovery can be done in Two ways. we can preempt resources. and we can Directly kill the process.

Deadlock Recovery.

Kill Process (s)

One simple procedure is kill all the processes.

Other approach is it picks a process and it kills it. It checks whether a deadlock is there or not. If deadlock is there it kills it.

Q How to pick a process to be killed ?

Ans There are multiple criteria for this. One is what is the priority of the process. How many resources the process is holding. Bcz if more resources the process is holding it more likely that this is the reason for deadlock. Usly This criteria your OS picks a process and kills it.

Preempt Resource.

There is a process who is holding Resource, you take away that ~~Process~~ Resource from that process. and This is problematic bcz. This Resource might have been used some time and process might have change state 1 to state 2.

In This case our OS must mantain states of the process. and when it preempts Resources. OS should move the process to its previous state.

Which is very much impractical Thing.

* Another problem is starvation. Same process Might taken again an again and it may lead to starvation.

⇐ This is what our OS does. and Recover from Deadlock for sure.

25/09/2021.

22. Process Synchronization.

Abhishek Sharma Notes.

So far we studied about independent Processes NOW Processes can talk to each other and they are called Cooperative Process.

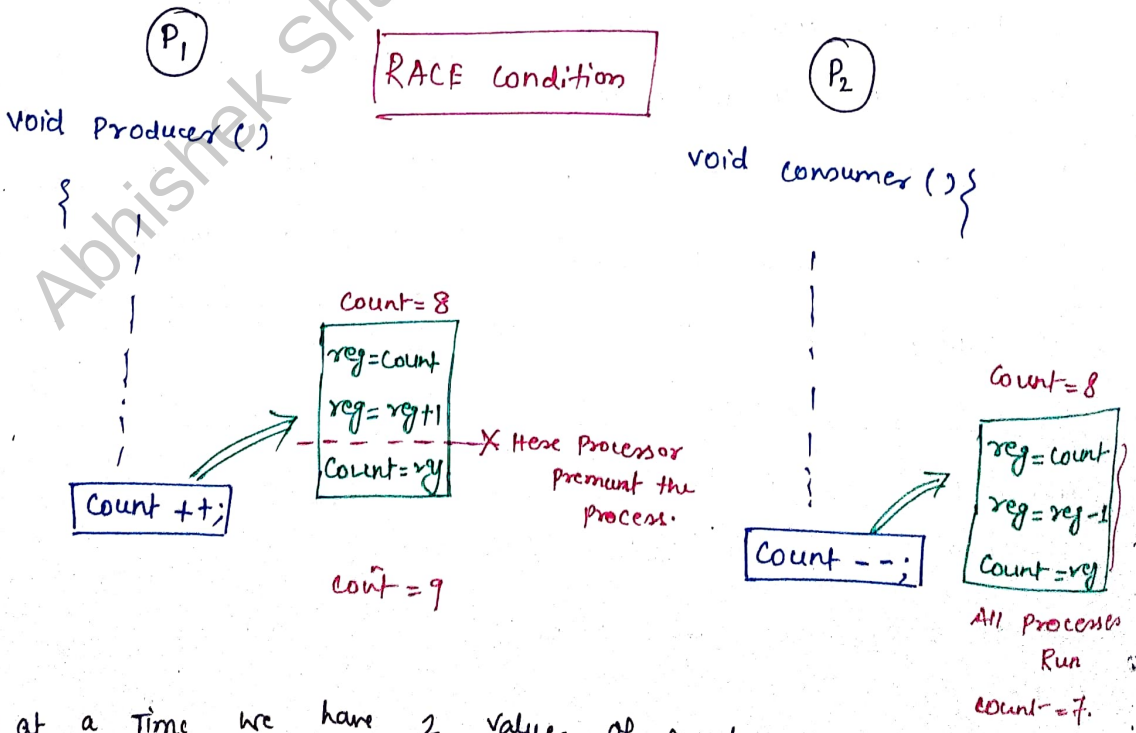
Process → Independent (Studied before).
Process → Cooperative. (NOW we will study)

Os have to decide How the processes will talk to each other in a Proper way.

* Process communitate within a system using shared memory. (Global varibles).

* we can also have Race condition in Process synchronization. bcz. Processes might Premupt in the middle and o/p will be vary if Two Processes are communicate (talk) to each other.

Example.



So at a Time we have 2 values of count and this is called Race condition.

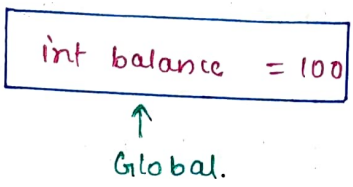
Inconsistant value.

23. Critical Section.

Abhishek Sharma Notes

Another Example of process synchronization and Race condition critical section comes when you have write or modify operation on shared variable. instead operation we don't have critical section.

Example.



Critical section: The part of code where you access shared varible is called critical section.

Non-critical Section: The part of code which is the remainig of shared varible is called non-critical section.

Entry Section: we put some logic before entering in critical section. so that only one process should be executing critical section.

Exit Section: After critical section it is Exit section.

Four
(4) Goals of synchronization Mechanism.

i) Mutual exclusion. (Only one process should be allowed in critical section)

ii) Progress : (If a process wish not to use critical section should not block other process).

Mendatory.

iii) Bounded waiting (Fair): (It should not happen that a process is waiting forever to enter into the critical section and other processes are getting chance again and again). So There should be bound on waiting on critical section.

iv) Performance: (your Locking Mechanism or Mechanism that allows processes to enter in the critical section should be fast. It shouldn't happen that the Mechanism takes a lot of time. There are Two types of Locking Mechanism. i) Software ii) Hardware.

* Hardware Locking Mechanism is usually fast. and used in operating system. many others.

Example of Syncronization Mechanism.

Example of Restroom.

As critical section.

⇒ Imagine it is a Restroom (common). Example of.

i) only one person should use it at a Time ⇒ Mutual Exclusion

ii) The people who don't want to use it, the can't lock it ⇒ Progress. so that other person can use the restroom.

iii) It shouldn't happen people using it again and again ⇒ Bounded waiting. and others are waiting

iv) The process of intering into Restroom should be fast ⇒ Performance.

25. Overview of Synchronization Mechanism. Abhishek Sharma Notes

Let's talk about different synchronization mechanism.

1) Disabling Interrupts:

This method Applicable for only single processor system. Another problem with this synchronization mechanism is it allows a process to disabling interupts. and it causes serious issues. ✗ not feasible.

Ex. In a restroom, one person is going inside and is not coming outside. It stays there for longtime. This causes problem for other persons Those are waiting outside.

It is not feasiable Mechanism bcz of 2 Reasons.

i) It not possible for multiple processors.

ii) It causes security issues.

2) Lock (or Mutex)

This method is basic building block to implement Synchronization.

Idea is we accure a lock going to the critical section and we release the lock.

Lock can be implemented in software and in Hardware.

- Lock
  - Software → Peterson's Implementation (for 2 Prosses).
    - Bakery Algorithm (Multiple Processors)
  - Hardware (Most common) ✓ (bcz it is fast)

3) Semaphores. : It is the High level Mechanism for basic Locks.

It has 2 operations. → wait
→ signal.

Wait : If a process wants to enter into the critical section. it calls wait.

Signal : Once the process releases the critical section. It causes signal. So that other processes can enter into critical section

4) Monitors : Monitors are mainly used for thread synchronizat

They are implemented by Java. It's a software solution.

Monitors are Higher level than semaphors. and semaphore are higher level than lock synchronization.

---

* Application of synchronization :

There are process synchronization and as well Thread synchron

* Thread synchronization happen by the user space or by kernal.

As a software Engineer you use Thread synchronization a lot.

* Process synchronization you might rarely come across.

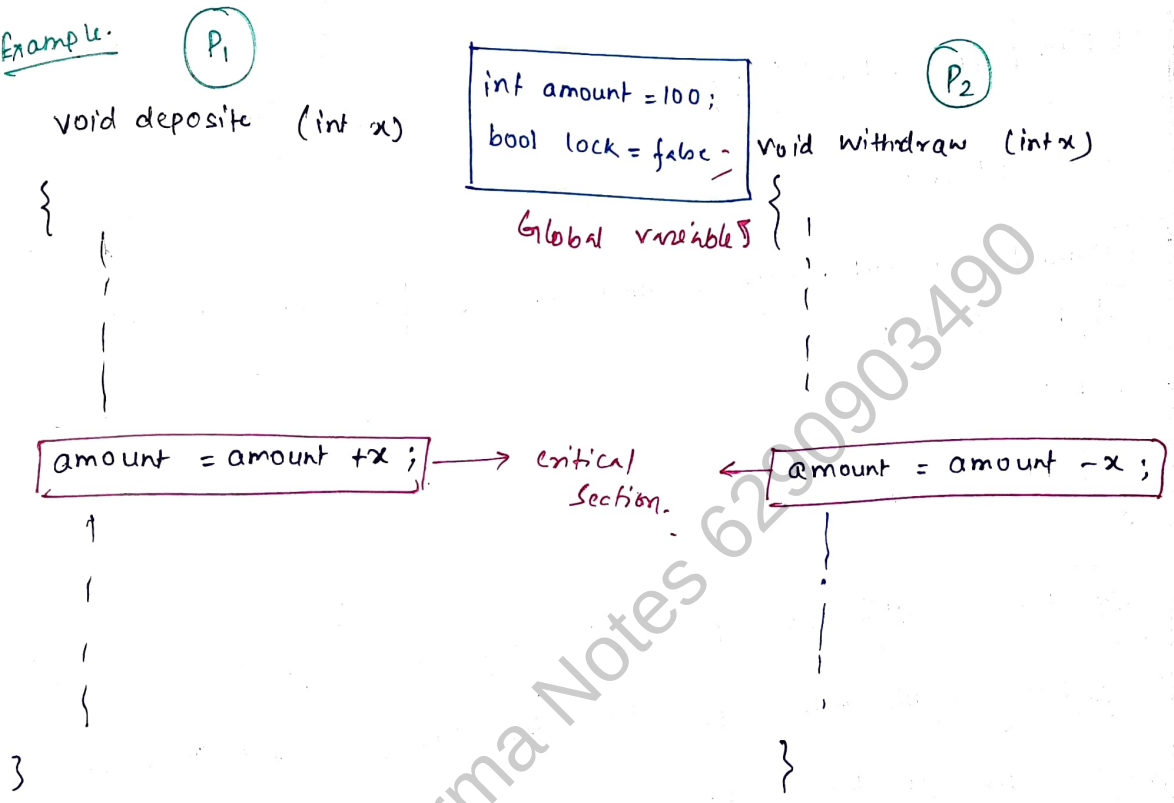
* Q) How do we decide which synchronization Mechanism is good enough?

A There are classical problem on synchronization which you should have try solving.

26. Locks for Synchronization.

This is the basic synchronization Method. Other Mechanisms like semaphores, Monitors uses lock to synchronize processes.

Example.



We have Two process P_1 and P_2 and They may be preempt at any time and may be resumed at any time. so we use Locks to disable preumption. we will syncronise results. So that they can not give incosistant result.

We will add a while loop which check lock is available or not. if a lock is available then we will wait for it become false. We will accure the critical Section Viz marking lock as true and once we have finished our work from critical section we will mark lock as false.

```
int amount = 100;
bool lock = false;
```
Global variable

P_1

```
Void deposite (int x) {
    :
    :
While (lock == true) { ; }
lock == true ;
amount = amount + x;
lock = false ;
    :
    :
}
```

P_2

```
void withdraw (int x) {
    :
    :
While (lock == true) { ; }
lock == true ;
amount = amount - x;
lock = false ;
    :
    :
}
```

→ critical section ←

If we use this code it doesn't give us guarenty for mutual exclusion. (The first requirement for process synchronization).

This is the problem.

Solution for this problem : TSL Lock.

Test and set lock system.

RACE condition atleast doesn't Happen.

Supported by your Hardware.

implemented by Hardware

---

* Problems with lock synchronization.

1) In Lock system there is a loop which continues waiting till lock is false.

This is called Busy Waiting.

(Ex: If a person is waiting outside the restroom. He has to wait until the room is lock.) This is called Busy waiting.

2) Another problem is there is no Queuing among the people.

Ex. A person use the restroom. Again takes the key and uses the restroom Another person are waiting outside. ⇒ Bounded waiting.

27. Semaphore.

To Deal with problems on Lock synchronization System. Semaphore came.

semaphore is simply a count variable and a Queue.

Semaphores are variables and implemented by OS. your cernal has to imple_ment.

```
struct Sem {
    int count;
    Queue q;
};
```

* count variable represents no. of Resourse you have available.

if these are 3 restrooms you have and no person is there.

Count = 3 and Queue = Empty.

* whenever a person uses the Restroom your count decreased.

and if count becomes < 0 after decreasing. that means.

no restrooms are available to use. So what we will do ??

we add this person to the Queue.

Two processes in Semaphore.

i) wait ()

ii) signal ().

wait (): Before acuring the resourse you call the wait process.

signal (): After using the resourse of critical section you call the signal process.

* wait process ⇒ Decreases the count
* signal process ⇒ increases the count.

Concept of Semaphore: Example: There is a security guard outside (Semaphore) your Restroom and a Queue of people who want to use the restroom.

If a person comes, (Assume count = 3 previously), It further decrease the count (count = 3-1 = 2) and ask the person to use the Restroom. so that at a time when count = 0 After 3 person comes.

Now a person comes from the queue and ask the Semaphore (~~Gate~~ Security Guard) to use the Restroom but count = 0 and no restrooms are available, So semaphore says to the person to take a sleep, whenever a restroom is available I will wake you up and permitted to use the restroom.

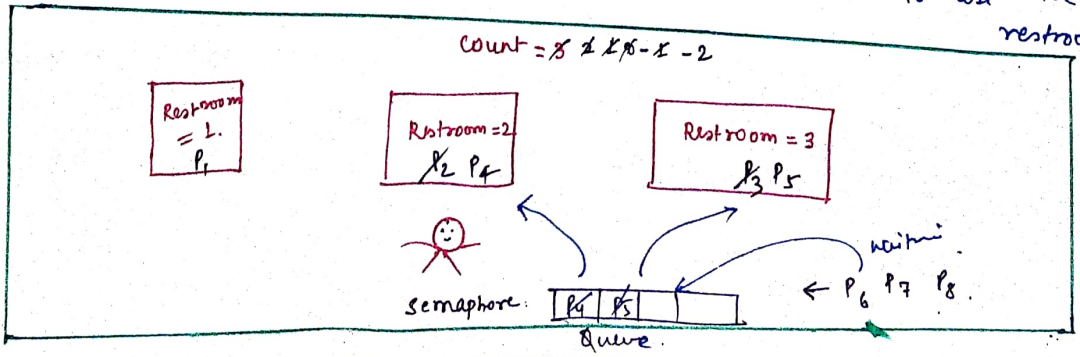
Semaphore stores that person's information in a Queue so that it will call that person after the Rooms are empty.

and the count goes to $\boxed{-1}$ count = -1.

⇓

Man's Rooms are not available and one person is waiting in the Queue.

* When a person leaves the restroom, Semaphore calls the signal() Process and increases the count. by 1. and it also does that it take the process from the Queue and grants permission to use the restroom



Pseudo code of wait() and signal().

```
struct sem {
    int count;
    Queue q;
};
```

⇒ Semaphore.

count, Queue.

Sem S (3, empty);

wait()	signal()
void wait () { S.count --; if (S.count < 0) { i) Add the caller process P to q ii) Sleep (P); } }	void signal () { S.count ++; if (S.count <= 0) { i) Remove a process P from q. ii) wakeup (P); } }

* original implementation ~~of Semaphore~~ by Dijkstra.

S = 3

```
P() {
    While (S == 0);
    S = S - 1;
}
```

⇒ wait().

```
V() {
    S = S + 1;
}
```

⇒ signal().

Here we have Problems like Busy waiting, Bounded waiting and we can solve this problems using Semaphore as we discussed previously. By using Queue.

28. Binary Semaphore.

Abhishek Sharma Notes.

Previously we have talked about Counting Semaphore.

In many Instances like in critical section or in many places we only need a **Binary Semaphore.**

A Binary Semaphore has value only True or False.

We use Binary Semaphore as a lock (or mutex) also.

Implementation of Binary Semaphore:

* wait() & signal() has to atomic.
* Atomicity acheived by lock.
* Semaphore are some variables and implemented by OS. using Lock.

V.V.I.

```
Struct BinSem
{
   bool val;
   Queue q;
};
```

BinSem S (true, empty)

val. → true

Queue → empty

⇓

Globale.

```
void wait ()
{
   if (S.val == 1)
      S.val = 0;
   else {
      1) put this Process P in q;
      2) sleep (P);
   }
}
```

```
void signal () {
   if (q is empty)
      S.val = 1;
   else {
      1) Pick a process P from q;
      2) wake up (P);
   }
}
```

⇓

These operations should be atomic.

29. Monitors.

This is the Last Synchronization Method. that we will study. Monitors are also built on top of locks. and implemented by MultiThreading Systems. Ex. Java. (In Java There is a Java virtual Machine that is like manager like OS. But it runs on top of the operating system. This Java virtual Machine manages multiple Threads) and This Java virtual Machine uses Monitors for manging the Threads.

Idea for Monitors: Instead of putting wait signals at multiple places you make a class, where you put your shared variables. and your function that operate upon this shared variables. What we do that we ~~[illegible]~~ make these function syncronized. (There is a keyword in Java ⇒ "synchronized".)

Monitors are. (Higher level of ~~Sema~~ Synchronization than semaphores).

* Monitors are better than semaphores.

Example of Monitors:

```
Class AccountUpdate {
    private int bal;
    void synchronized deposite (int x) {
        bal = bal + x;
    }
    void synchronized withdraw (int x) {
        bal = bal - x;
    }
}
```

Explanation.

We have one class where we put shared variable or critical section part which are shared by Multiple Thread and we have syncronized functions under the class.

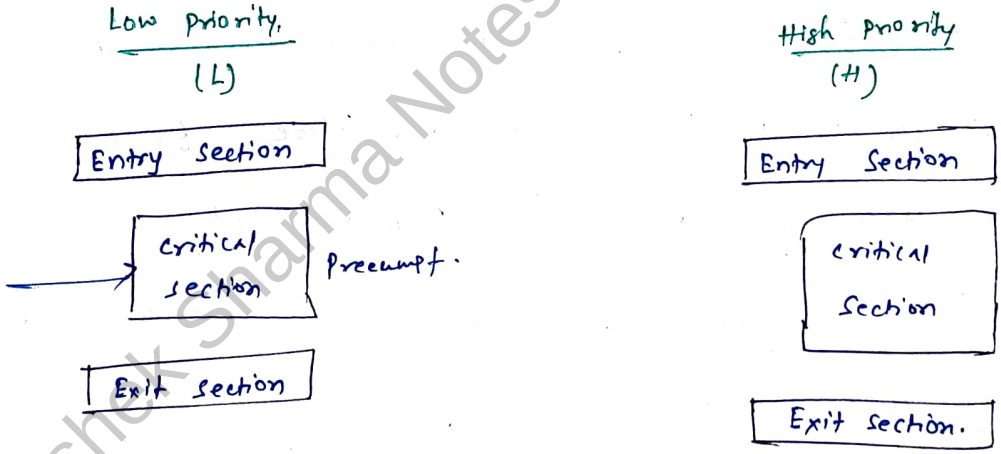
* If your class has only one synchronized function Then also it is a Monitor.

30. Priority Inversion.

It is a problem in synchronization. named Priority Inversion.

Problem: There is a Low priority process (L) and There is a High Priority process (H). They Both share a critical section. It might Happen Low priority process (L) Intors into the critical section and while it was There in critical section (L) was preempted. Now This High priority process (H) gets into CPU But it has to wait for Low priority Process (L) To be scheduled again and finishes.

So Here Priority Inversion Happen. A High priority process (H) is waiting for a low priority process (L). to finish first.

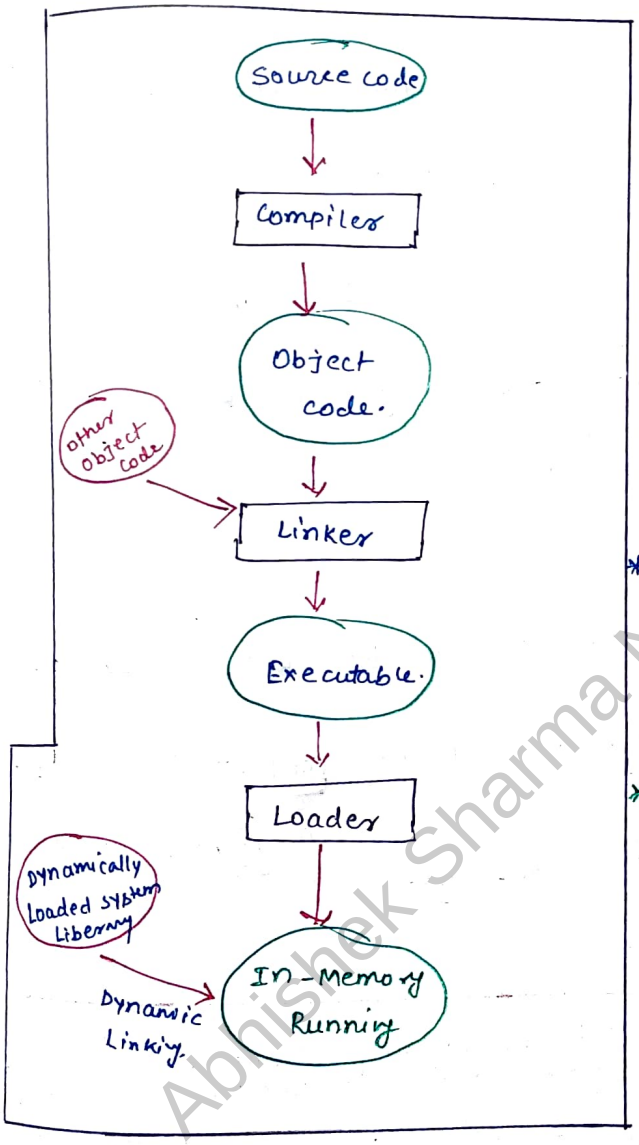


Q) How to solve this problem of priority Inversion.

Sol) **Priority Inheritence.** And priority Inheritence says that If a Low ~~priority~~ priority process (L) is sharing the critical section with a High Priority process (H). Then the Low priority Process (L) while executing critical section must ~~inherits~~ inheritates the priority of High Priority Process (H). So that a low or medium priority process can't block a Higher Priority process (H).

31. How are programs compiled and Run.

* How programs compiles and Run ?? What are the process when we press [Ctrl + F9] Button or GCC in Linux and we run a.out file.
  - ↳ windows.



Explanation.

Source ~~dara~~ code is first given to compiler, compiler converts source code to object code.

Ex.

```
# include <stdio.h>
int main(){
    cout << "Hello";
}
```

⇒ a.c file.

** This a.c file is first given to compiler and compiler converts into a machine specific object code.

** Once object code is produced. Linker comes into picture. Linker Links the functions that you have called inside your main function.

Ex. Here we have called "cout" and "cout" is a Liberary function. Linker takes the code of "cout" and puts in our file.

* Linking can be done dynamically or statically. In statically Linking code of the cout function is copied to your object file and one executable file is generated. * In Dynamic Linking. we don't copy the whole "cout" function in our object file we just put a place Holder. and when your program loaded in the memory. when it needs "cout" first then it searched in the memory during runtime.

Linker: Combines codes of other (or Library) functions and produced an executable file.

After a executable file is made Loader comes into picture.

* Loader is something that runs the executable file. It copies the executable file Binary into main memory. So that it can be run these step by step.

While it is runnig it can be dynamically Linked with the other libereries. Ex. may be "cout" is dynamically linked then it will be linked with "cout" dynamically during runtime.

* Dynamically linking requires support of the operating system.

All these Things Happen when we press ctrl +F9 in windows. or gcc in Linux.

That's how a program compiles and run.

and give us the output.

---

32. Memory Management in Operating systems.

Our computer mainly consists of 2 components.

i) CPU (Does computations)

ii) Memory. (stores Data).

Memory has different forms and there are certain desirable features from Memory:

i) Memory should be accessable fast. so that access time should be really small.

ii) The cost to store every bit should be very low.

iii) Capacity of the Memory should be High. You should be allowed to store lots of lots of data in the Memory.

From Memory we expect thing.

Access Time ↓ (Low)

Capacity ↑ (High)

Cost ↓ (Low)

All these Three Things do not come together

Ex. Cache Memory has very Low Access Time, High cost, and Low capacity.

Main memory has Middle, and secondary Memory (Like Harddisk) has the Highest access Time, lowest cost and Highest capacity.

Q) So How do we store data in these 3 Memories ??

A) Like a Shopkeeper. Imagine you are a shopkeeper. You are selling some items. You produce many items and put those many items in your store. Which is your secondary Disk. and The items which are generally demanded by the Customer you put them in the shop. and the item which are most demanded by the customers you put them at your front counter. So Front counter is like your Cache Memory.

Abhishek Sharma Notes.
Ph: 6290903490.

and whenever something is ~~is~~ not in the front counter and some customer is asking for it you go to your shop on your backside and you bring that item. and some item is not in your shop you go to your store. you bring that item from store and deleiver it to the customer.

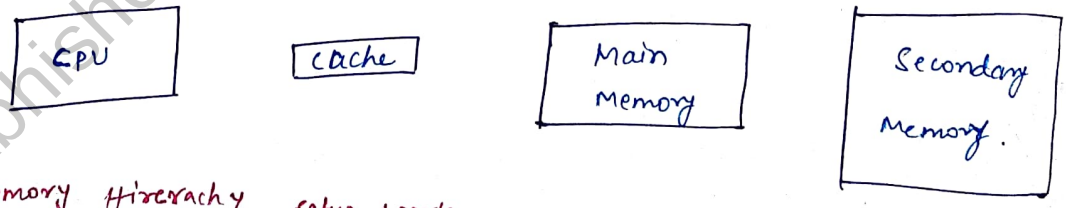
The same idea is Memory Hierarchy.

We put Lowest Access Time Memory closest to the CPU.

Then main memory Then Secondary Memory.

So, cache is your front counter, Main Memory is your shop. ~~Store~~ and Secondary Memory is your store.

Memory Hierarchy.



This Memory Hierarchy setup works really great bcz of the ~~locality~~ Locality of the reference. When we ~~use~~ access an item its very likely that the items near it are going to be access next so the nearby item are brought into cache. and cache misses are generally less.