UNIT-2

Syllabus

Process and CPU Scheduling : Process concepts- the process, process states, process control block, Threads, process scheduling- Scheduling queues, schedulers, context switch, preemptive scheduling, dispatcher, scheduling criteria, scheduling algorithms, multiprocessor scheduling, real time scheduling, Thread scheduling, case studies Linux, Windows.

Process Coordination- Process synchronization, the critical- section problem, Peterson's Solution, synchronization Hardware, semaphores, classic problems of synchronization, monitors, Case studies Linux, Windows.

1.1 Process

Differences between process and program

1.2 Processstates

1.3 Process controlblock

1.4. Threads:

Thread states

Differences between process and thread

Multithreading

Types of threads

- **userthreads**
- **kernal threads**

Multithreadingmodels

- **many to manyrelationship.**
- **many to onerelationship.**
- **one to onerelationship.**

Differences between user level threads and kernel level threads

1.5. Processscheduling

Scheduling queues

- **jobqueue**
- **ready queue**
- **device queue**

Types ofschedulers

- **Long termscheduler:**
- **Short termscheduler:**
- **Medium term scheduler**

Schedulingcriteria

1.6 .CPU schedulingalgorithms

1.First come first servedscheduling(FCFS)

2)Shortest job first scheduling (SJF)

3)Shortest remaining time first (SRTF)

4)Round robin scheduling algorithm(RR)

5)Priority scheduling

6) Multilevel queuescheduling

7) Multilevel feedbackqueues

1.7. Threadscheduling

contention scope

- **process contentionscope**
- **system contentionscope**

1.8. Multiple – Processorscheduling

1)Approaches to multiple-processor scheduling

a)Asymmetric multiprocessing

b)symmetricmultiprocessing

2)processoraffinity

a)soft affinity

b)hard affinity

3)Loadbalancing

push migration

pull migration

4)Multicore processors

5)Virtualization andscheduling

1.9.Real timescheduling

1.10.ProcessCoordination

The critical section problem

1.11.Peterson solution

1.12.Synchronization hardware

1.13.Semaphores

1.14.Classic problems ofsyncronization

- **Bounded-bufferproblem**
- **The readers-writersproblem**
- **Dining philosophers problem**

1.15.Monitors

- **Solution to producer consumer problem usingmonitors**
- **Solution to dining philososphers problem usingmonitors**
- **Resuming processes with in amonitor**

1.1 Process

A process is a program at the time of execution.

Differences between Process and Program

Process	Program
Process is a dynamic object	Program is a static object
Process is sequence of instruction execution	Program is a sequence of instructions
Process loaded in to main memory	Program loaded into secondary storage devices
Time span of process is limited	Time span of program is unlimited
Process is a active entity	Program is a passive entity

1.2 ProcessStates

When a process executed, it changes the state, generally the state of process is determined by the current activity of the process. Each process may be in one of the following states:

1. New : The process is beingcreated.
2. Running : The process is beingexecuted.
3. Waiting : The process is waiting for some event tooccur.
4. Ready : The process is waiting to be assigned to a processor.
5. Terminated : The Process has finishedexecution.

Only one process can be running in any processor at any time, But many process may be in ready and waiting states. The ready processes are loaded into a “ready queue”.

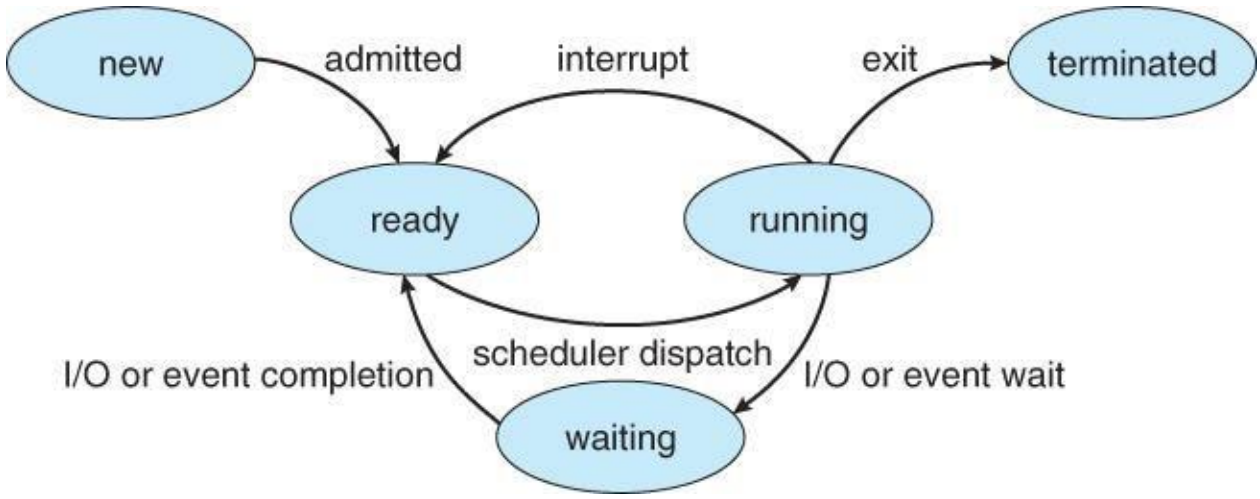


Diagram of process state

a)New ->Ready : OS creates process and prepares the process to be executed,thenOSmoved the process into readyqueue.

b)Ready->Running : OS selects one of the Jobs from ready Queue and move themfrom ready to Running.

c)Running->Terminated : When the Execution of a process has Completed, OSterminatesthatprocess from running state. Sometimes OS terminates the process for someother reasons including Time exceeded, memory unavailable, access violation, protection Error, I/O failure and soon.

d)Running->Ready : When the time slot of the processor expired (or) If the processorreceivedanyinterrupt signal, the OS shifted Running -> ReadyState.

e)Running -> Waiting : A process is put into the waiting state, if the process need an event occur (or) an I/O Devicerequire.

f)Waiting->Ready : A process in the waiting state is moved to ready state when the eventforwhichit has beenCompleted.

1.3 Process ControlBlock:

Each process is represented in the operating System by a Process Control Block.

It is also called Task Control Block. It contains many pieces of information associated with a specific process.

Process State
Program Counter
CPU Registers
CPU Scheduling Information
Memory – Management Information
Accounting Information
I/O Status Information

Process Contol Block

1. **ProcessState** : The State may be new, ready, running, and waiting,Terminated…
2. **ProgramCounter** : indicates the Address of the next Instruction to beexecuted.
3. **CPUregisters** : registers include accumulators, stack pointers, Generalpurpose Registers….
4. **CPU-SchedulingInfo** : includes a process pointer, pointers to schedulingQueues,otherscheduling parametersetc.
5. **Memory management Info**: includes page tables, segmentation tables, value of base andlimit registers.
6. **AccountingInformation** : includes amount of CPU used, time limits, Jobs(or)Processnumbers.
7. **I/O StatusInformation** : Includes the list of I/O Devices Allocated to theprocesses,listof openfiles.

1.4. Threads:

A process is divide into number of light weight process, each light weight process is said to be a Thread. The Thread has a program counter (Keeps track of which instruction to execute next), registers (holds its current working variables), stack (execution History).

Thread States:

1. bornState : A thread is justcreated.
2. readystate : The thread is waiting forCPU.
3. running : System assigns the processor to thethread.
4. sleep : A sleeping thread becomes ready after the designated sleep timeexpires.
5. dead : The Execution of the threadfinished.

Eg: Word processor.

Typing, Formatting, Spell check, saving are threads.

Differences Between Process and Thread

Process	**Thread**
process takes more time to create.	thread takes less time to create.
it takes more time to complete execution & terminate.	less time to terminate.
execution is very slow.	execution is very fast.
it takes more time to switch b/w two processes.	It takes less time to switch b/w two threads.
Communication b/w two processes is difficult .	communication b/w two threads is easy.
process can't share the same memory area.	threads can share same memory area.
system calls are requested to communicate each other.	system calls are not required.
process is loosely coupled.	threads are tightly coupled.
It requires more resources to execute.	requires few resources to execute.

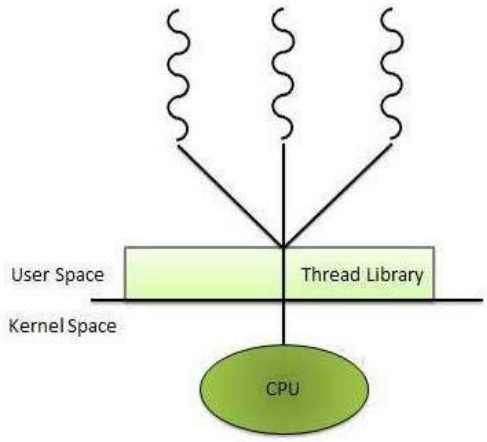
Multithreading

A process is divided into number of smaller tasks each task is called a Thread. Number of Threads with in a Process execute at a time is called Multithreading.

If a program, is multithreaded, even when some portion of it is blocked, the whole program is not blocked.The rest of the program continues working If multiple CPU's are available. Multithreading gives best performance.If we have only a single thread, number of CPU's available, No performance benefits achieved.

Types OfThreads:

1) **User Threads** : Thread creation, scheduling, management happen in user space by Thread Library. user threads are faster to create and manage. If a user thread performs a system call, which blocks it, all the other threads in that process one also automatically blocked, whole process isblocked.



Advantages

- Thread switching does not require Kernel modeprivileges.
- User level thread can run on any operatingsystem.
- Scheduling can be application specific in the user levelthread.
- User level threads are fast to create andmanage.

Disadvantages

- In a typical operating system, most system calls areblocking.
- Multithreaded application cannot take advantage ofmultiprocessing.

2)Kernal Threads: kernel creates, schedules, manages these threads .these threads are slower, manage.If one thread in a process blocked, over all process need not beblocked.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process onmultiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the sameprocess.
- Kernel routines themselves canmultithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the userthreads.
- Transfer of control from one thread to another within same process requires a mode switch tothe Kernel.

Multithreading Models

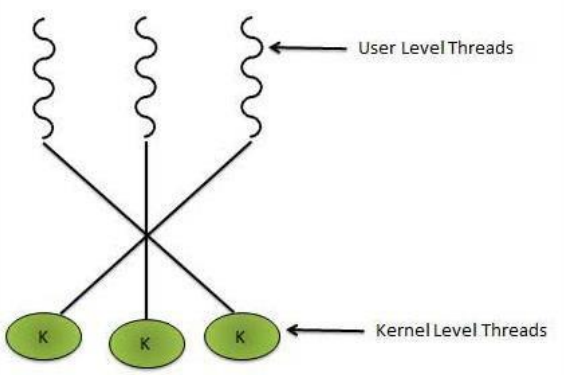
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to manyrelationship.
- Many to onerelationship.
- One to onerelationship.

Many to Many Model

In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.

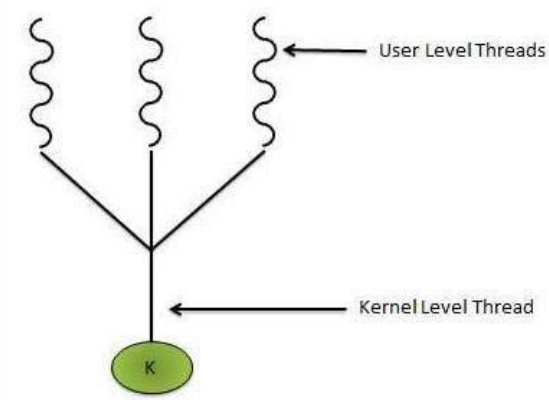
Following diagram shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.



Many to One Model

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

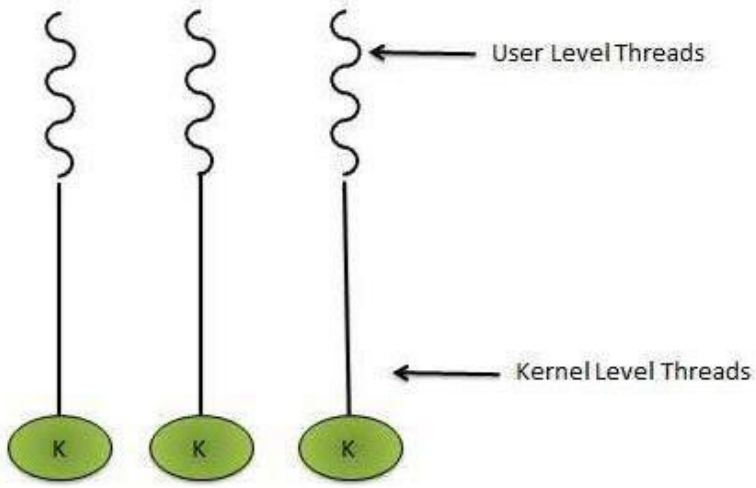
If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



One to One Model

There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Difference between User Level & Kernel Level Thread

S.N.	User Level Threads	Kernel Level Thread
1	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User level thread is generic and can run on any operating system.	Kernel level thread is specific to the operating system.
4	Multi-threaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

1.5. PROCESS SCHEDULING:

CPU is always busy in **Multiprogramming**. Because CPU switches from one job to another job. But in **simple computers** CPU sit idle until the I/O request granted.
scheduling is a important OS function. All resources are scheduled before use.(cpu, memory, devices…..)

SCHEDULING QUEUES: people live in rooms. Process are present in rooms knows as queues.

Theseare3types
1. job queue: when processes enter the system, they are put into a **job queue**, which consists all processes in the system. Processes in the job queue reside on mass storage and await the allocation of mainmemory.

2.ready queue: if a process is present in main memory and is ready to be allocated to cpu for execution, is kept in **readyqueue.**

3. device queue: if a process is present in waiting state (or) waiting for an i/o event to complete is said to bein device queue.
(or)
The processes waiting for a particular I/O device is called device queue.

Schedulers : There are 3 schedulers

1. Long termscheduler.
2. Medium termscheduler
3. Short termscheduler.

Scheduler duties :

- maintains thequeue.
- Select the process from queues assign toCPU.

Types of schedulers

1. **Long termscheduler:**
   select the jobs from the job pool and loaded these jobs into main memory (ready queue).
   Long term scheduler is also called job scheduler.
2. **Short termscheduler:**
   select the process from ready queue, and allocates it to the cpu.
   If a process requires an I/O device, which is not present available then process enters device queue.
   short term scheduler maintains ready queue, device queue. Also called as cpu scheduler.
3. **Medium term scheduler**: if process request an I/O device in the middle of the execution, then the process removed from the main memory and loaded into the waiting queue. When the I/O operation completed, then the job moved from waiting queue to ready queue. These two operations performed by medium termscheduler.

Comparison between Scheduler

S.N.	Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch: Assume, main memory contains more than one process. If cpu is executing a process, if time expires or if a high priority process enters into main memory, then the scheduler saves information about current process in the PCB and switches to execute the another process. The concept of moving CPU by scheduler from one process to other process is known as context switch.

Non-Preemptive Scheduling: **CPU** is assigned to one process, CPU do not release until the completition of that process. The CPU will assigned to some other process only after the previous process has finished.

Preemptive scheduling: here CPU can release the processes even in the middle of the execution. CPU received a signal from process p2. OS compares the priorities of p1 ,p2. If p1>p2, CPU continues the execution of p1. If p1<p2 CPU preempt p1 and assigned to p2.

Dispatcher: The main job of dispatcher is switching the cpu from one process to another process. Dispatcher connects the cpu to the process selected by the short term scheduler.

Dispatcher latency: The time it takes by the dispatcher to stop one process and start another process is known as dispatcher latency. If the dispatcher latency is increasing, then the degree of multiprogramming decreases.

SCHEDULING CRITERIA;

1. **Throughput**: how many jobs are completed by the cpu with in a timeperiod.
2. **Turn around time** : The time interval between the submission of the process and time of the completion is turn aroundtime.
   TAT = Waiting time in ready queue + executing time + waiting time in waiting queue for I/O.
3. **Waiting time**: The time spent by the process to wait for cpu to beallocated.
4. **Response time**: Time duration between the submission and firstresponse.
5. **Cpu Utilization**: CPU is costly device, it must be kept as busy aspossible.
   Eg: CPU efficiency is 90% means it is busy for 90 units, 10 units idle.

1.6 .CPU SCHEDULINGALGORITHMS:

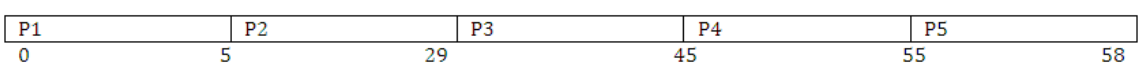
1.First come First served scheduling: (FCFS): The process that request the CPU first is holds the cpu first. If a process request the cpu then it is loaded into the ready queue, connect CPU to thatprocess.

Consider the following set of processes that arrive at time 0, the length of the cpu burst time given in milli seconds.

burst time is the time, required the cpu to execute that job, it is in milli seconds.

Process	Burst time(milliseconds)
P1	5
P2	24
P3	16
P4	10
P5	3

Chart:



Average turn around time:

Turn around time= Finished time- arrival time

Turn around time for p1= 5-0=5.

Turn around time for p2 =29-0=29

Turn around time for p3=45-0=45

Turn around time for p4=55-0=55

Turn around time for p5= 58-0=58

Average turn around time= (5+29++45+55+58/5) = 187/5 =37.5 millisecounds

Average waiting time:

waiting time= starting time- arrival time

Waiting time for p1=0

Waiting time for p2=5-0=5

Waiting time for p3=29-0=29

Waiting time for p4=45-0=45

Waiting time for p5=55-0=55

Average waiting time= 0+5+29+45+55/5 = 125/5 = 25 ms.

Average Response Time :

Formula : First Response - Arrival Time

Response Time for P1 =0

Response Time for P2 => 5-0 = 5

Response Time for P3 => 29-0 = 29

Response Time for P4 => 45-0 = 45

Response Time for P5 => 55-0 = 55

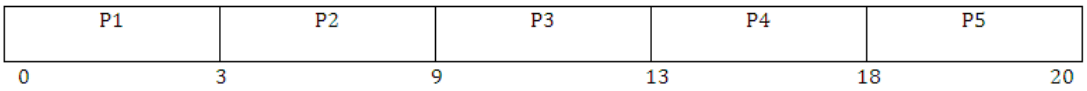
Average Response Time => (0+5+29+45+55)/5 =>25ms

1)First Come FirstServe:

It is Non Primitive Scheduling Algorithm.

PROCESS	BURST TIME	ARRIVAL TIME
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8

Process arrived in the order P1, P2, P3, P4, P5.
P1 arrived at 0 ms.
P2 arrived at 2 ms.
P3 arrived at 4 ms.
P4 arrived at 6 ms.
P5 arrived at 8 ms.



Average Turn Around Time :

Formula : Turn around Time = Finish Time - Arrival Time

Turn Around Time for P1 => 3-0 = 3
Turn Around Time for P2 => 9-2 = 7
Turn Around Time for P3 => 13-4 = 9
Turn Around Time for P4 => 18-6 = 12
Average Turn Around Time => (3+7+9+12+12)/5 =>43/5 = 8.50 ms.

Average Response Time :

Formula : Response Time = First Response - Arrival Time

Response Time of P1 = 0
Response Time of P2 => 3-2 = 1
Response Time of P3 => 9-4 = 5
Response Time of P4 => 13-6 = 7
Response Time of P5 => 18-8 =10
Average Response Time => (0+1+5+7+10)/5 => 23/5 = 4.6 ms

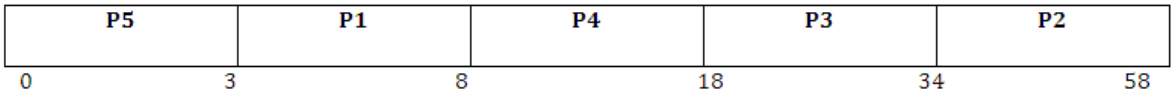
advantages : Easy to Implement, Simple.

disadvantage : Average waiting time is very high.

2)Shortest Job First Scheduling (SJF):

Which process having the smallest CPU burst time, CPU is assigned to that process . If two process having the same CPU burst time, FCFS is used.

PROCESS	CPU BURST TIME
P1	5
P2	24
P3	16
P4	10
P5	3



P5 having the least CPU burst time (3ms). CPU assigned to that (P5). After completion of P5 short term scheduler search for nest (P1).....

Average Waiting Time :

Formula = Staring Time - Arrival Time
waitting Time for P1 => 3-0 = 0
waiting Time for P2 => 34-0 = 34
waiting Time for P3 => 18-0 = 18
waiting Time for P4 =>8-0=8
waiting time for P5=0
Average waiting time => (3+34+18+8+0)/5 => 63/5 =12.6 ms

Average Turn Around Time :

Formula = Finish Time - Arrival Time

Turn Around Time for P1 => 8-0 = 8
Turn Around for P2 => 58-0 = 58
Turn Around for P3 => 34-0 = 34
Turn Around Time for P4 => 18-0 = 18
Turn Around Time for P5 => 3-0 = 3
Average Turn around time => (8+58+34+18+3)/5 => 121/5 = 24.2 ms

Average Response Time :

Formula : First Response - Arrival Time

First Response time for P1 =>3-0 = 3
First Response time for P2 => 34-0 = 34
First Response time for P3 => 18-0 = 18
First Response time for P4 => 8-0 = 8
First Response time for P5 = 0
Average Response Time => (3+34+18+8+0)/5 => 63/5 = 12.6 ms

SJF is Non primitive scheduling algorithm

Advantages : **Least average waiting time**
Least average turn around time
Least average response time
Average waiting time (FCFS) = 25 ms
Average waiting time (SJF) = 12.6 ms
50% time saved in SJF.
Disadvantages:

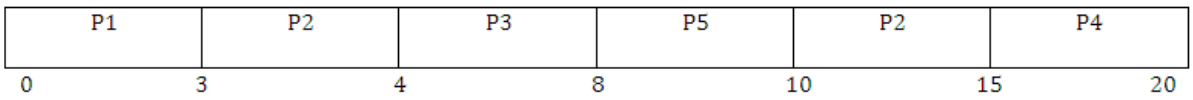
- knowing the length of the next CPU burst time isdifficult.
- Aging (Big Jobs are waiting for long time for CPU)

3)Shortest Remaining Time First (SRTF):

This is primitive scheduling algorithm.

Short term scheduler always chooses the process that has term shortest remaining time. When a new process joins the ready queue , short term scheduler compare the remaining time of executing process and new process. If the new process has the least CPU burst time, The scheduler selects that job and connect to CPU. Otherwise continue the old process.

PROCESS	BURST TIME	ARRIVAL TIME
P1	3	0
P2	6	2
P3	4	4
P4	5	6
P5	2	8



P1 arrives at time 0, P1 executing First , P2 arrives at time 2. Compare P1 remaining time and P2 (3-2 = 1) and 6. So, continue P1 after P1, executing P2, at time 4, P3 arrives, compare P2 remaining time (6-1=5) and 4 (4<5) .So, executing P3 at time 6, P4 arrives. Compare P3 remaining time and P4 (4-2=2) and 5 (2<5). So, continue P3 , after P3, ready queue consisting P5 is the least out of three. So execute P5, next P2, P4.

FORMULA : Finish time - Arrival Time

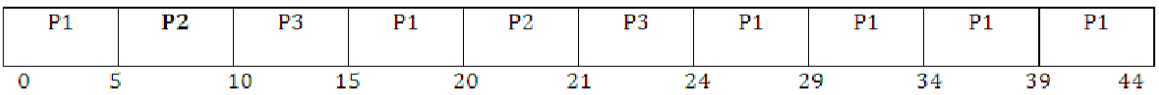
Finish Time for P1 => 3-0 = 3
Finish Time for P2 => 15-2 = 13
Finish Time for P3 => 8-4 =4
Finish Time for P4 => 20-6 = 14
Finish Time for P5 => 10-8 = 2

Average Turn around time => 36/5 = 7.2 ms.

4)ROUND ROBIN SCHEDULING ALGORITHM :

It is designed especially for time sharing systems. Here CPU switches between the processes. When the time quantum expired, the CPU switched to another job. A small unit of time, called a time quantum or time slice. A time quantum is generally from 10 to 100 ms. The time quantum is generally depending on OS. Here ready queue is a circular queue. CPU scheduler picks the first process from ready queue, sets timer to interrupt after one time quantum and dispatches the process.

PROCESS	BURST TIME
P1	30
P2	6
P3	8



AVERAGE WAITING TIME :

Waiting time for P1 => 0+(15-5)+(24-20) => 0+10+4 = 14
Waiting time for P2 => 5+(20-10) => 5+10 = 15
Waiting time for P3 => 10+(21-15) => 10+6 = 16
Average waiting time => (14+15+16)/3 = 15 ms.

AVERAGE TURN AROUND TIME :

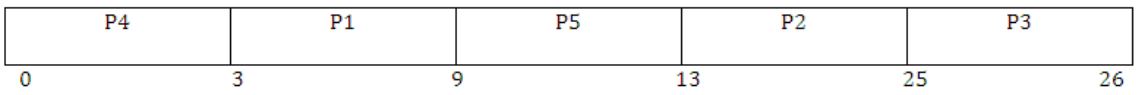
FORMULA : Turn around time = Finished time - Arrival Time

Turn around time for P1 => 44-0 = 44
Turn around time for P2 => 21-0 = 21
Turn around time for P3 => 24-0 = 24
Average turn around time => (44+21+24)/3 = 29.66 ms

5)PRIORITY SCHEDULING :

PROCESS	BURST TIME	PRIORITY
P1	6	2
P2	12	4
P3	1	5
P4	3	1
P5	4	3

P4 has the highest priority. Allocate the CPU to process P4 first next P1, P5, P2, P3.



AVERAGE WAITING TIME :

Waiting time for P1 => 3-0 3
Waiting time for P2 => 13-0 = 13
Waiting time for P3 => 25-0 = 25
Waiting time for P4 => 0
Waiting time for P5 => 9-0 =9

Average waiting time => (3+13+25+0+9)/5 = 10 ms

AVERAGE TURN AROUND TIME :

Turn around time for P1 => 9-0 = 9
Turn around time for P2 => 25-0 = 25
Turn around time for P3 => 26-0 = 26
Turn around time for P4 => 3-0 = 3
Turn around time for P5 => 13-0 = 13

Average Turn around time => (9+25+26+3+13)/5 = 15.2 ms

disadvantage : Starvation

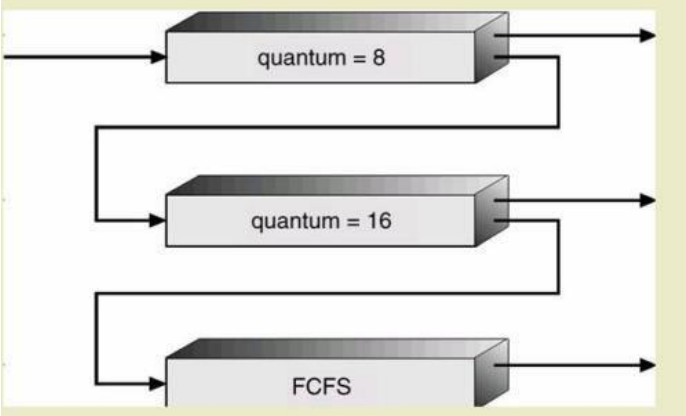
Starvation means only high priority process are executing,but low priority process are waiting for the CPU for the longest period of the time.

6) Multilevel QueueScheduling:

ready queue is partitioned into number of ready queues.Each ready queue is capable to load same type of jobs.Each ready queue has its own scheduling algorithm.ready queue is partitioned into 4 ready queues.one ready queue for system processes,one ready queue for background processes,one ready queue for foreground processes,another ready queue for student processes. No student process run unless system,foreground,background process were all empty.
each queue gets a certain portion of the cpu time ,which it can then schedule among its various processes.

7) Multilevel FeedbackQueues:

This algorithm allows a process to move between the queues.if a process uses too much cpu time,it will be moved to next low priority queue.a process that waits too long in a lower priority queue may be moved to a higher priority queue.It prevents starvation.



E.G: It has 3 queues(Q0,Q1,Q2),Scheduler first executes all process in Q0.only when Q0 is empty will it execute Q1.The process in Q2 will only be executed ,if Q0,Q1 are empty.high priority queue is Q0,low priority queue is Q2.
A process entering the ready queue is put in Q0.A process in Q0 is given a time quantum of 8 ms.if it does not finish with in this time,moved to tail of Q1.if Q0 is empty,the process at the head of the queue

Q1 is given a time quantum of 16 ms.if it does not complete,put into Q2.Q2 processes are run on FCFS basis.but these processes are run only when Q0,Q1 are empty.
This algorithm gives highest priorities to any process with a CPU burst of 8 ms (or) less.process that need more than 8 but less than 24 ms are also served quickly.long processes automatically sink to Q2,are served in FCFS.

1.7. ThreadScheduling

Kernel-level threads scheduled by the Operating system.user level threads managed by a thread library.To run a CPU,user level threads must ultimately be mapped to an associated kernel level thread.

Contention scope:

Defines whether a thread is to contend for processing resources relative to other threads with in the same process (or) relative to other threads within the same system.

a)Process contentionscope:

Competition for the CPU takes palce among threads belonging to the same process.

b)System contentionscope:

Completion for the CPU takes place among all threads in the system.

1.8. Multiple – processorscheduling:

When multiple processes are available,then the scheduling gets more complicated ,because there is more than one CPU which must be kept busy and in effective use at all times.
Load sharing resolves around balancing the load between multiple processors.multi processor systems may be heterogeneous(It contains different kinds of CPU's) (or) Homogeneous(all the same kind of CPU).

1)Approaches to multiple-processor scheduling

a)Asymmetricmultiprocessing

One processor is the master,controlling all activities and running all kernel code,while the other runs only user code.

b)Symmetric multiprocessing:

Each processor schedules its own job.each processor may have its own private queue of ready processes**.**

2)ProcessorAffinity

successive memory accesses by the process are often satisfied in cache memory.what happens if the process migrates to another processor.the contents of cache memory must be invalidated for the first processor,cache for the second processor must be repopulated.most Symmetric multi processor systems try to avoid migration of processes from one processor to another processor,keep a process running on the same processor.this is called processor affinity.

a)Softaffinity:

Soft affinity occurs when the sysyem attempts to keep processes on the same processor but makes no guarantees.

b)Hardaffinity:

Process specifies that it is not to be moved between processors.

3)Loadbalancing:

One processor wont be sitting idle while another is overloaded.
Balancing can be achived through push migration or pull migration.

Push migration:
Push migration involves a separate process that runs periodically(e.g every 200 ms) and moves processes from heavily loaded processors onto less loaded processors.
Pull migration:
Pull migration involves idle processors taking processes from the ready queues of the other processors.

4)Multicoreprocessors:

A multi-core processor is a single computing component with 2 or more independent actual processing units(cores),which are the units that read and execute program instructions.
When a processor acceses memory,it spends some time waiting for the data to become available.this is known as memory stall.To remedy this,design multithreaded processor cores in which 2 or more hardware threads are assigned to each core.If one thread stall while waiting for memory,the core can switch to another thread.

From operating system perspective,each hardware thread appears as a logical processor that is availableto run a software thread.thus on a dual threaded,dual core system,four logical processors are presented to the operatingsystem.

5) irtualization andscheduling:

In virtualization,single-CPU system,frequently acts like a multiprocessor system.the virtualization software presents one or more virtual CPU's to each of the virtual machines running on the system and then schedules the use of the physical CPU's among the virtual machines.most virtual environments have one host OS,many guest OS.The host OS creates and manages the virtual machines,each virtual machine has a guest OS installed and applications running within that guest.

1.9.Real timescheduling:
Real time scheduling is generally used in the case of multimedia operating systems.Here multiple processes compete for the CPU.How to schedule processes A,B,C so that each one meets its deadlines.
The general tendency is to make them pre-emptable,so that a process in danger of missing its deadline can preempt another process.when this process sends its frame,the preempted process can continue from where it had left off.here throughput is not so significant.Important is that tasks start and end as per their deadlines.

1.10.Processcoordination:

Process synchronization refers to the idea that multiple processes are to join up orhandshake at a certain point, in order to reach an agreement or commit to a certain sequence of action. Coordination of simultaneous processes to complete a task is known as process synchronization.

The critical section problem

Consider a system ,assume that it consisting of n processes.Each process having a segment of code.This segment of code is said to be critical section.

E.G: Railway Reservation System.

Two persons from different stations want to reserve their tickets,the train number,destination is common,the two persons try to get the reservation at the same time. unfortunately, the available berths are only one,both are trying for that berth.

It is also called the critical section problem.solution is when one process is executing in its critical section,,no other process is to be allowed to execute in its critical section.

The critical section problem is to design a protocol that the processes can use to cooperate.Each process must request permission to enter its critical section.The section of code implementing this request is the **entry section**.The critical section may be followed by an **exit section**.The remaining code is the **remainder section.**

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (1);
```

Figure General structure of a typical process P_i.

A solution to the critical section problem must satisfy the following 3 requirements:

1.mutual exclusion:

Only one process can execute their critical section at any time.

2.Progress:

When no process is executing a critical section for a data,one of the processes wishing to enter a critical section for data will be granted entry.

3.boundedwait:

No process should wait for a resource for infinite amount of time.

Critical section:
The portion in any program that acceses a shared resource is called as critical section (or) critical region.

1.11.Petersonsolution:
Peterson solution is one of the solutions to critical section problem involving two processes.This solution states that when one process is executing its critical section then the other process executes the rest of the code and vice versa.
Peterson solution requires two shared data items:
1)turn: indicates whose turn it is to enter into the critical section.
If turn == i ,then process i is allowed into their criticalsection.
2)flag: indicates when a process wants to enter into critical section.when process i wants to enter their critical section,it sets flag[i] totrue.

Process 0

set flag0 to true
set turn to 1
while(flag1 && turn==1)
do nothing
end do
critical section
set flag0 to false

Process 1

set flag1 to true
set turn to 0
while(flag0 && turn==0)
do nothing
end do
critical section
set flag1 to false

1.12.Synchronizationhardware
In a uniprocessor multiprogrammed system,mutual exclusion can be obtained by disabling the interrupts before the process enters its critical section and enabling them after it has exited the critical section**.**

Disable interrupts
Critical section
Enable interrupts

Once a process is in critical section it cannot be interrupted.This solution cannot be used in multiprocessor environment.since processes run independently on different processors.
In multiprocessor systems, **Testandset** instruction is provided,it completes execution without interruption.Each process when entering their critical section must set **lock**,to prevent other processes

from entering their critical sections simultaneously and must release the lock when exiting their critical sections.

```
repeat
        while testandset(lock)
        do nothing
        critical section
        lock=false
until false
```

a process wants to enter critical section and value of lock is false then **testandset** returns false and the value of lock becomes true.thus for other processes wanting to enter their critical sections **testandset** returns true and the processes do busy waiting until the process exits critical section and sets the value of lock to false.

Swap instruction can also be used for mutual exclusion

```
repeat
        key=true
        repeat
                swap(lock,key)
        until key=false;
                Critical section
        Lock=false
Until false
```

---

lock is global variable initialized to false.each process has a local variable key. A process wants to enter critical section,since the value of lock is false and key is true.

lock=false
key=true

after swap instruction,

lock=true
key=false

now key=false becomes true,process exits repeat-until,and enter into critical section.
When process is in critical section (lock=true),so other processes wanting to enter critical section will have

lock=true
key=true

Hence they will do busy waiting in repeat-until loop until the process exits critical section and sets the value of lock to false.

1.13.Semaphores

A semaphore is an integer variable.semaphore accesses only through two operations.

1)wait: wait operation decrements the count by1.

If the result value is negative,the process executing the wait operation is blocked.

2)signaloperation:

Signal operation increments by 1,if the value is not positive then one of the process blocked in wait operation unblocked.

---

var sem:semaphore
*wait(sem):**sem.count**=**sem.count**-1;*
*if **sem.count**<0 then*
begin
    *add this process in the queue **sem.queue***
    block this process
end;
*signal(sem):**sem.count**=**sem.count**+1;*
*if **sem.count**<=0 then*
begin
    *remove a process p from **sem.queue***
    add process p to ready queue
end;

In binary semaphore count can be 0 or 1.

The value of semaphore is initialized to 1.

```
repeat
    wait(sem);
    critical section
    signal(sem);
until false;
```

First process that executes wait operation will be immediately granted sem.count to 0.

If some other process wants critical section and executes wait() then it is blocked,since value becomes -1.

If the process exits critical section it executes signal().sem.count is incremented by 1.blocked process is removed from queue and added to ready queue.

Problems:

1)Deadlock

Deadlock occurs when multiple processes are blocked.each waiting for a resource that can only be freed by one of the other blocked processes.

2)Starvation

one or more processes gets blocked forever and never get a chance to take their turn in the critical section.

3)Priorityinversion

If low priority process is running ,medium priority processes are waiting for low priority process,high priority processes are waiting for medium priority processes.this is called Priority inversion.

The two most common kinds of semaphores are **counting semaphores** and **binary semaphores**. Counting semaphores represent multiple resources, while binary semaphores, as the name implies, represents two possible states (generally 0 or 1; locked or unlocked).

1.14.Classic problems of syncronization

Bounded-bufferproblem

Two processes share a common ,fixed –size buffer.

Producer puts information into the buffer,consumer takes it out.

The problem arise when the producer wants to put a new item in the buffer,but it is already full.The solution is for the producer has to wait until the consumer has consumed atleast one buffer.similarly if the consumer wants to remove an item from the buffer and sees that the buffer is empty,it goes to sleep until the producer puts something in the buffer and wakes it up.

---

synchronisation problems:

i) we must guard against attempting to write data to the buffer when the buffer is full; ie the producer must wait for an 'empty space'.

ii) we must prevent the consumer from attempting to read data when the buffer is empty; ie, the consumer must wait for 'data available'.

To provide for each of these conditions, we require to employ three semaphores which are defined in the following table:

Semaphore	*Purpose*	*Initial Value*
free	mutual exclusion for buffer access	1
space	space available in buffer	N
data	data available in buffer	0

Producer	*Explanation*
produce item	Application produces data item
wait (space)	If buffer full, wait for space signal
wait (free)	If buffer being used, wait for free signal
add item to buffer	All clear; put item in next buffer slot
signal (free)	Signal that buffer no longer in use
signal (data)	Signal that data has been put in buffer
Consumer	
wait (data)	Wait until at least one item in buffer
wait (free)	If buffer being used, wait for free signal
get item from buffer	All clear; get item from buffer
signal (free)	Signal that buffer no longer in use
signal (space)	Signal that at least one space exists in buffer
consume item	Application specific processing of item.

It should be noted that the counting semaphores *data* and *space* are also used as counters to monitor the occupancy of the buffer and that the *wait* and *signal* operations decrement and increment these counters. Thus, the producer's *signal*(*data*) will increment the count of the number of data items while the consumer's *wait* (*data*) will decrement the count of data items.

The readers-writers problem

A database is to be shared among several concurrent processes.some processes may want only to read the database,some may want to update the database.If two readers access the shared data simultaneously no problem.if a write,some other process access the database simultaneously problem arised.Writes have excusive access to the shared database while writing to the database.This problem is known as readers-writes problem.

First readers-writers problem

No reader be kept waiting unless a writer has already obtained permission to use the shared resource.

Second readers-writes problem:

Once writer is ready,that writer performs its write as soon as possible.

A process wishing to modify the shared data must request the lock in write mode.multiple processes are permitted to concurrently acuire a reader-writer lock in read mode.A reader writer lock in read mode.but only one process may acquire the lock for writing as exclusive access is required for writers.

(i) A proposed solution tackles these issues by assigning higher priorities to the reader processes, as compared to the writer processes.
(ii) When the first reader process accesses the database, it performs a DOWN on the database. This prevents any writing process from accessing the database.
(iii) While this reader is reading the database, if another reader arrives, that reader simply increments a counter RC, which indicates how many readers are currently active.

(iv) Only when the counter RC becomes 0 (which indicates that no reader is active), a writer can write to the database.

An algorithm to implement this functionality is shown in Fig.

```
/* Initialize semaphore variables */
integer mutex = 1;                  /* Controls access to RC */
integer DB = 1;                     /* Controls access to the database */
integer RC = 0;                     /* Number of processes reading the database currently */
1. Reader ( )                       /* The algorithm for the reader process */
   Repeat continuously
        DOWN (mutex);               /* Lock (get exclusive access to) the counter RC */
        RC = RC + 1;                /* One more reader */
        If (RC = 1) DOWN (DB);      /* This is the first reader. Lock database for reading */
        UP (mutex);                 /* Release exclusive access to RC */
        Read_Database ( );          /* Read the database */
        DOWN (mutex);               /* Lock (get exclusive access to) the counter RC */
        RC = RC - 1;                /* Reader count less by one now */
        If (RC = 0) UP (DB);        /* This is the last reader. Unlock database */
        UP (mutex);                 /* Release exclusive access to RC */
   End
```

```
2. Writer ( )                          /* The algorithm for the writer process */
   Repeat continuously
            DOWN (DB);                 /* Lock (get exclusive access to) the database */
            Write_Database ( );        /* Read the database */
            UP (DB);                   /* Release exclusive access to the database */
   End
```

Fig. Solution to the readers and writers problem

Clearly, this solution assigns higher priority to the readers, as compared to the writers.

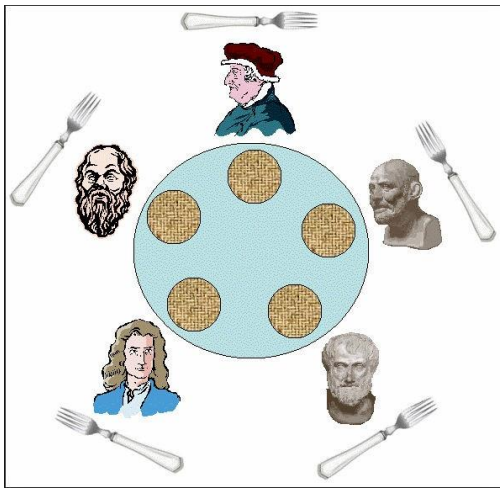
(i) If many readers are active when a writer arrives, the writer must wait until all the readers end their reading jobs.

(ii) Moreover, if a few more readers keep coming in, the writer has to wait until all of them finish reading. This may not always be the best solution, but surely is secure.

Dining Philosophers problem

Five philosophers are seated on 5 chairs across a table.Each philosopher has a plate full of noodles.Each philosopher needs a pair of forks to eat it.There are only 5 forks available all together.there is only one fork between any two plates of noodles.

In order to eat, a philosopher lifts two forks,one to his left and the other to his right.if he is successful in obtaining two forks,he starts eating after some time,he stops eating and keeps both the forks down.



What if all the 5 philosophers decide to eat at the same time ?

All the 5 philosophers would attempt to pick up two forks at the same time. So,none of them succeed.

One simple solution is to represent each fork with a semaphore.a philosopher tries to grab a fork by executing wait() operation on that semaphore.he releases his forks by executing the signal() operation.This solution guarantees that no two neighbours are eating simultaneously.

Suppose all 5 philosophers become hungry simultaneously and each grabs his left fork,he will be delayed forever.

```
1. Main ( )                              /* This is the main task of every philosopher */
   Repeat continuously
            Think ( );                   /* Philosopher is thinking for some time */
            Take_forks ( );              /* Acquire both the forks, or wait if this is not possible */
            Eat ( );                     /* Eat spaghetti */
            Put_forks ( );               /* Put down both the forks on the table */
   End
2. Take_forks ( )
   DOWN (mutex);                         /* Enter critical region (obtain exclusive access to State) */
   State = Hungry;                       /* Signify that the philosopher is hungry */
   Test (Philosopher );                  /* Try to acquire both the forks */
   UP (mutex);                           /* Exit critical region (release exclusive access to State) */
   DOWN (Philosopher);                   /* Block if forks could not be acquired */
3. Put_forks ( )
   DOWN (mutex);                         /* Enter critical region (obtain exclusive access to State) */
   State = Thinking;                     /* Signify that the philosopher has finished eating */
   Test (LEFT );                         /* Check if the left neighbor can now eat */
   Test (RIGHT);                         /* Check if the right neighbor can now eat */
   UP (mutex);                           /* Exit critical region (obtain exclusive access to State) */
4. Test ( )
   If the philosopher is hungry and both his neighbors are not eating
   Then set the State of this philosopher = Eating;
   UP (Philosopher);                     /* Philosopher can be tested by other philosophers */
```

Solution to Dining Philosophers Problem

Several remedies:

1)allow atmost 4 philosophers to be sitting simultaneously at the table.

2)allow a philosopher to pickup his fork only if both forks are available.

3)an odd philosopher picksup first his left fork and then right fork.an even philosopher picksup his right fork and then his left fork.

1.15.Monitors

The disadvantage of semaphore is that it is unstructured construct.wait and signal operations can be scattered in a program and hence debugging becomes difficult.

A monitor is an object that contains both the data and procedures needed to perform allocation of a shared resource.To accomplish resource allocation using monitors,a process must call a **monitor entry routine**.Many processes may want to enter the monitor at the same time.but only one process at a time is allowed to enter.Data inside a monitor may be either global to all routines with in the monitor (or) local to a specific routine.Monitor data is accessible only with in the monitor.There is no way for processes outside the monitor to access monitor data.This is a form of information hiding.

If a process calls a monitor entry routine while no other processes are executing inside the monitor,the process acquires a lock on the monitor and enters it.while a process is in the monitor,other processes may not enter the monitor to acuire the resource.If a process calls a monitor entry routine while the other monitor is locked the monitor makes the calling process wait outside the monitor until the lock on the monitor is released.The process that has the resource will call a monitor entry routine to release the resource.This routine could free the resource and wait for another requesting process to arrive monitor entry routine calls signal to allow one of the waiting processes to enter the monitor and acquire the resource. monitor gives high priority to waiting processes than to newly arriving ones.

Syntax:

```
type monitor_name = monitor
var: variable declarations
procedure entry p1(……)
begin
    ------------
end
procedure entry p2(……..)
begin
       -------------
end

begin
       initialization code
end
```

processes can call procedures p1,p2,p3……They cannot access the local variables of the monitor.

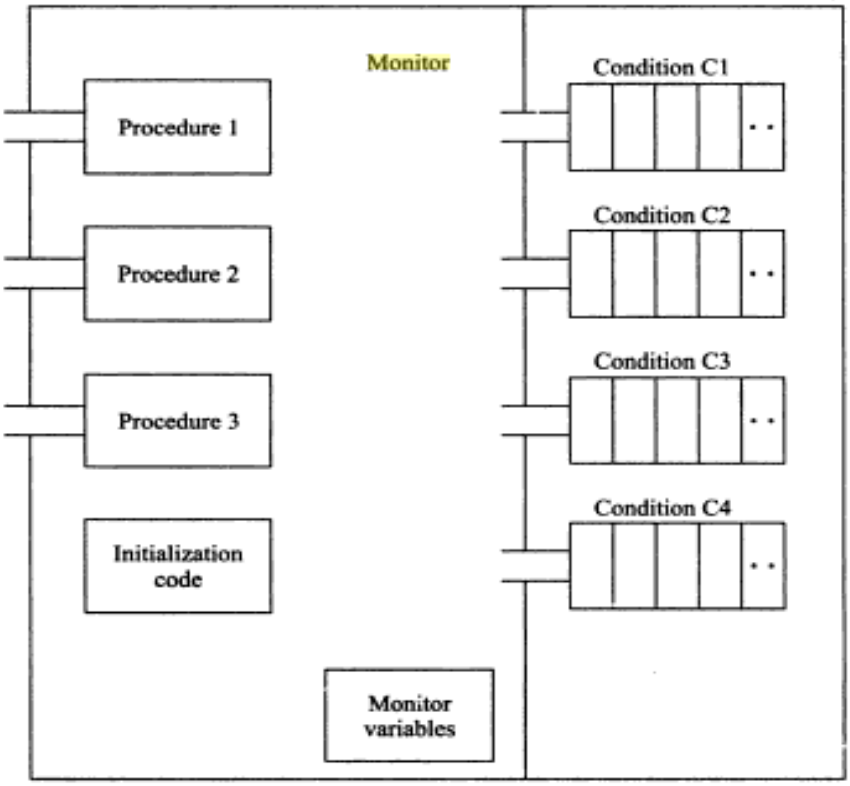


Fig. Structure of a monitor

monitor provides condition variables along with two operations on them i.e. wait and signal.

wait(condition variable)
signal(condition variable)

Every condition variable has an associated queue.A process calling wait on a particular condition variable is placed into the queue associated with that condition variable.A process calling signal on a particular condition variable causes a process waiting on that condition variable to be removed from the queue associated with it. And reenter the monitor.

Solution to Producer consumer problem using monitors:

```
monitor producerconsumer
condition full,empty;
int count;

procedure insert(item)
{
        if(count==MAX)
                wait(full)
          insert_item(item);
        count=count+1;
        if(count==1)
        signal(empty);
}
procedure remove()
{
        if(count==0)
                wait(empty);
        remove_item(item);
        count=count-1;
        if(count==MAX-1)
        signal(full);
}
procedure producer()
{
        producerconsumer.insert(item);
}
procedure consumer()
{
        producerconsumer.remove();
}
```

Condition: Buffer contains atleast one item.

Solution to dining philosophers problem using monitors

```
monitor dp
{
    enum {thinking, hungry, eating} state[5];
    condition self[5];

    void pickup(int i) {
       state[i] = hungry;
       test(i);
       if (state[i] != eating)
          self[i].wait();
    }

    void putdown(int i) {
       state[i] = thinking;
       test((i + 4) % 5);
       test((i + 1) % 5);
    }

      void test(int i) {
         if ((state[(i + 4) % 5] != eating) &&
            (state[i] == hungry) &&
            (state[(i + 1) % 5] != eating)) {
               state[i] = eating;
               self[i].signal();
         }
      }

      void init() {
         for (int i = 0; i < 5; i++)
            state[i] = thinking;
      }
  }
```

Figure A monitor solution to the dining-philosopher problem.

A philosopher may pickup his forks only if both of them are available.A philosopher can eat only if his two neighbours are not eating.some other philosopher can delay himself when he is hungry.

Diningphilosophers.Take_forks( ) : acquires forks ,which may block the process.
Eat noodles( )
Diningphilosophers.put_forks( ): releases the forks.

Resuming processes with in a monitor

If several processes are suspended on condion x and x.signal( ) is executed by some process.then
how do we determine which of the suspended processes should be resumed next ?
solution is FCFS(process that has been waiting the longest is resumed first).In many circumstances,such simple technique is not adequate.alternate solution is to assign priorities and wake up the process with the highest priority.

Resource allocation using monitor

```
boolean inuse=false;
conditionavailable;
                        //conditionvariable
monitorentry void getresource()
{
        if(inuse)       //is resource inuse
        {
                wait(available);        wait until available issignaled
        }
        inuse=true;     //indicate resource is now inuse
}
monitorentry void returnresource()
{
        inuse=false;    //indicate resource is
        notinusesignal(available);  //signal a waiting process
        toproceed
}
```