Threads :

Thread: Thread is a basic unit of CPU utilization (OR) Smallest execution unit. It comprises of thread ID, a program counter, a register set and a stack.

Single Threaded	Multithreaded
* Able to service only one client at a time. & a client might have to wait for long time for its request to be serviced.	* Enhance processing capabilities on multicore systems. * Performs CPU intensive tasks in parallel across multiple computing cores.

→ Remote Process Call (RPC):
Thread plays important role in RPC systems. RPCs provide communication mechanism to allow interprocess communication. RPC servers are multithreaded

→ Benefits of Multithreading:

① Responsiveness:
Multithreading is an interactive application. If a time consuming operation is performed on a single thread, application remains responsive.

② Resource sharing:
Process share resources → shared memory / Message passing
Threads share memory & resources of the process to which they belong by default.

③ Economy:
Since thread share resources by default it is more economical to create and context-switch threads.

④ Scalability:
Multithreading benefits more to multiprocessing architecture where threads may be running in parallel on different processing cores

→ Multicore

* Multiple
whether
called

* Multithre
of mult

Data P

* Data divid
computing
single t
perforne

Eg:
1 Sorting
2 Word cou
3 Searchi

* More spe
executio

* Parallelis

→ Multicore Programming :=

* Multiple computing cores on a single chip, whether across CPU chips or within CPU chips called Multicore or multiprocessor systems.

* Multithreaded programming → more efficient use of multicore systems and improved concurrency.
  i.e: threads can run in parallel.

Parallelism

Data Parallelism

* Data divides accross parallel computing nodes but single task is performed

Eg:
1. Sorting
2. Word count
3. Searching

* More speed b/c of one execution thread
* Parallelism is proportional to input data size.

Task Parallelism

* Data divides across parallel compiling nodes with multiple divided Tasks (Processes)

Eg
1. Matrix mulliplication (x, +)
2. Gaming
3. Banking System.

* Less speed as each will execute a different thread.
* Parallelism is proportional to no. of independent tasks.

→ Chalenges for programmer for better utilization of Multicore programming.

① Dividing tasks: Concurrent tasks are seperated to run on different processors.

② Balance: Balance work distributed to each processor.

③ Data Splitting: Data accessed and manipulated by the tasks must be divided to run on seperate cores.

④ Data Dependency: Programmers must ensure that the execution of the dependent tasks must be synchronized.

⑤ Testing & Debugging: Testing & debugging programs running in parallel is difficult.

→ Amdahl's Law:

" Identifies speedup in latency of the execution tasks at fixed work load by adding additional computing cores to an application. "

$$\text{speedup} \leq \frac{1}{S + (1-S)/N}$$

S = serial execution portion.

N = processing cores.

* Fork → divide - Join Models:
  - If problem is small, solve directly, using sequential algorithm.
  - Else divide tasks into subtasks on multiple parallel thread, having one master thread.

→ Multithreading Models:

① User thread:
- Threads provided by the user level.
- User threads are supported above the kernel
- User threads are managed without kernel support.

② Kernel thread:
- Threads provided at the kernel level.
- Kernel threads are supported and managed directly by the OS.

* Multithreading Models establish the relationships between the user thread & kernel thread.

① Many - to - One Model:

- Many User level threads mapped to single kernel level thread.
- Only one thread can access the kernel at a time
- Multiple threads are unable to run in parallel on multicore systems.
- Entire process will block, if a thread makes a blocking system call.

Disadvantage: inability to take advantage of multiple processing cores.

② One - to - One Model:

- Each User level thread maps to Kernel level thread.
- More concurrent because process can run even if single thread makes a blocking system call.
- Multiple threads can run in parallel on multicore systems.

Disadvantage:

- Creating a user thread requires creating the corresponding kernel thread.
- Overhead of creating kernel threads burdens performance of application.

72766 = 35
2048

③ Many-to-Many Model:

- Multiplexes many user-level threads to equal or smaller number of kernel-level threads.
- Multiple user threads can be created as per requirement.
- Multiple kernel threads corresponding to user thread can run in parallel
- Kernel can schedule another user thread for execution even if one thread makes a blocking system call.

→ Examples:

- Green thread (For Solaris systems & early versions of JAVA) used Many-to-One model.
- Linux along with WOS implement One-to-One model
- Solaris OS supported Two-level-Model (Many-to-Many Model).

→ Thread Libraries :-

A thread library provides the programmer with an API for creating and managing threads

- Three main thread libraries:

① POSIX Pthreads :
- May be provided either as user-lever or kernel-level library. (Linux & Unix)

② Windows Thread:
Kernel level library on Windows System (Windows)

③ JAVA Thread API:
Allows threads to be created & managed directly in JAVA programs. (JVM).

→ Implicit Threading :

To address the difficulties of multicore programming transfer the creation & management of threading from application developers to compilers and run-time libraries. It is called Implicit threading.

Alternative approaches of designing multithreaded programs for better utilization of multicore processors through implicit threading.

① Thread Pool ② Open MP ③ Grand Central

① Thread Pools:
- Create a number of place them into a
- When a server rec to the available is completed, thre more work.

Benefits:
* Servicing a reques waiting to cre
* Limits the nu that cannot su
- Creation of

② Open MP:
- A set of c programs wr provides su memory en

③ GCD:
- Is combin a run-time to identify

① Thread Pools:

- Create a number of threads at process startup and place them into a pool, where they wait for work. When a server receives a request, it passes the request to the available thread for service. Once service is completed, thread returns to pool and wait for more work.

Benefits:

* Servicing a request with an existing thread is faster than waiting to create a thread.
* Limits the number of threads. Good for systems that cannot support a large number of concurrent threads.

- Creation of threads depends on processor's specifications.

② Open MP:

- A set of compiler directives as well as an API for programs written in C, C++ or FORTRAN that provides support for parallel programming in shared memory environment.

③ GCD:

- Is combination of extensions to C, an API, and a run-time library that allows application developers to identify sections of code to run in parallel.

→ Signal Handling:

- To notify a process that a particular event occured.

Signal generated → delivered to process → handled.

- Signal handling is easy in single-threaded program, as signal are delivered to the process.
- However, delivering signals is complicated in multithreaded programs.
  - Deliver to the thread to which the signal applies
  - Deliver to every thread in process.
  - Deliver to certain thread in process.
  - Assign a specific thread to receive all signals for the process.

CPU Scheduling

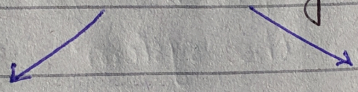
→ CPU-I/O Burst Cycle:

- Process execution consist of a cycle of CPU execution & I/O execution.
- CPU burst is followed by I/O burst.

→ Processes:

- CPU bound: Require most of time on CPU
- I/O bound: Require most of time on I/O devices.

CPU Scheduling



Non-Premptive

- Rigid
- Can not interrupt b/w execution.
- CPU allocated till process terminates or switches to waiting state
- No context switching hence no overhead of switching.

Preemptive

- Flexible
- Can interrupt b/w execution
- CPU allocated for limited time
- Overhead of switching processes and maintaining ready queue.

* Burst time / Execution time / Running time:
Time process requires for running on the CPU

* Waiting time: Time spend by a process in ready state waiting for CPU.

* Arrival Time: When a process enters ready state

* Exit Time: When a process completes execution and exit from the system.

* Turn Around Time: Total time spend by a process in the system.

$$T.A.T = E.T - A.T = B.T + W.T$$

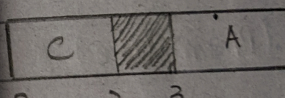
* Response time: Time between which a process enters ready queue and get scheduled on the CPU for the first time.

→ Criteria for Scheduling Algorithm:
- Avg. waiting time → Min
- Avg. response time → Min
- CPU utilization → Max
- Throughput → Max
(No. of process executing per unit time)

→ First Come First
- No starvation
- Implemented u
- Non-preemptive

Pid	A.T
A	3
B	5
C	0
D	5
E	4

Gannit Chart:



Avg. TAT =

Avg. W.T =

* Convoy effec
long

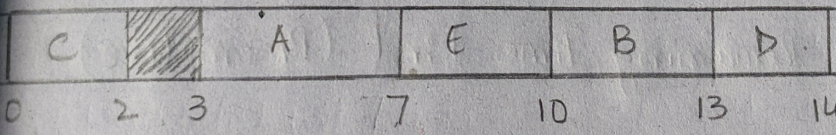
* Starvation:
is biased

→ First Come First Serve Algorithm:

- No starvation
- Can serve from convoy effect.
- Implemented using queue.
- Non-premptive.

Pid	A.T	B.T	T.A.T (C.T - A.T)	W.T = TAT - BT
A	3	4	7-3 = 4	0
B	5	3	13-5 = 8	5
C	0	2	2-0 = 2	0
D	5	1	14-5 = 9	8
E	4	3	10-4 = 6	3

Gannt Chart:



Avg. TAT = 5.8

Avg. W.T = 3.2

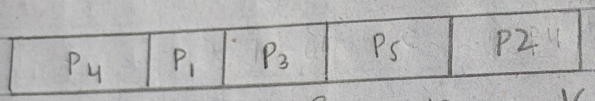
* Convoy effect: Smaller process have to wait for long time for bigger process to release CPU.

* Starvation: If process has to wait because processor is biased

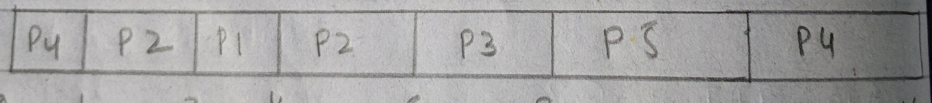
→ Shortest - Job - First (Non - preemptive):

- CPU is assigned to the process having smallest burst time.

Pid	A.T	B.T	T.A.T	W.T
P_1	3	1	7-3=4	3
P_2	1	4	16-1=15	11
P_3	4	2	9-4=5	3
P_4	0	6	6-0=6	0
P_5	2	3	12-2=10	7



→ Shortest - Remaining Time First (Preemptive):



TAT	W.T
4-3=1	0
6-1=5	1
8-4=4	2
16-0=16	10
11-2=9	6

- Gives minimal avg. WT.
- Can not be implemented.
- Standard algo.

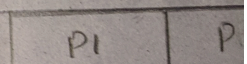
- Processes

→ Priority Alg

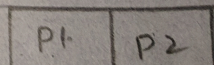
- CPU is alloc highest pri

Non - pree

Pid	AT
P1	0
P2	1
P3	2
P4	3
P5	4



Preemptive



smallest

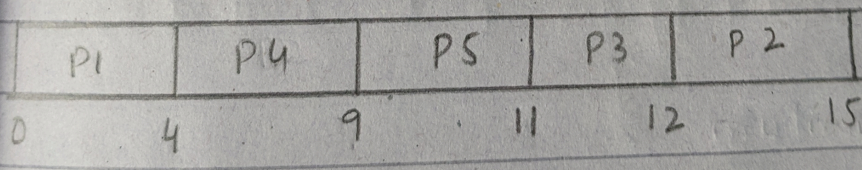
- Processes starved.

→ Priority Algorithm:

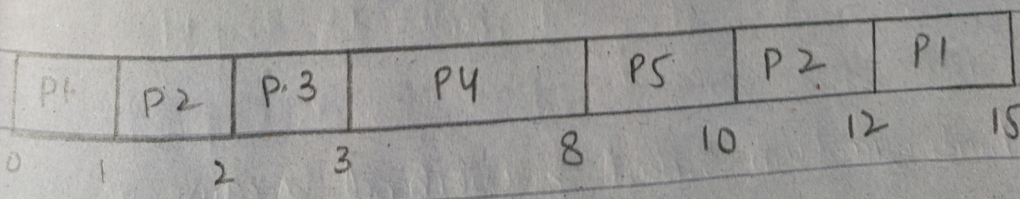
- CPU is allocated to the processes with the highest priority.

Non-preemptive: (Suppose highest number have higher priority.)

Pid	AT	BT	Priority	TAT	WT
P1	0	4	2	4 - 0 = 4	0
P2	1	3	3	15 - 1 = 14	11
P3	2	1	4	12 - 2 = 10	9
P4	3	5	5	9 - 3 = 6	1
P5	4	2	5	11 - 4 = 7	5



Preemptive:



:- low priority processes suffer from starvation.

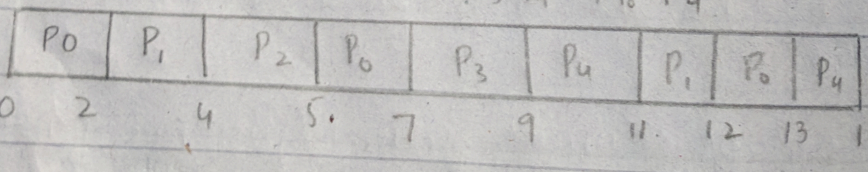
→ Round-Robin Algorithm:

- Fifo variant.
- Assigns equal time slots to each process called Time Quantum.
- Best avg. response time.
- Preemptive algo.

Pid	AT	BT	TAT	W.T
P_0	0	5		
P_1	1	3		
P_2	2	1		
P_3	3	2		
P_4	4	3		

Time Quantum = 2.

P_0 P_1 P_2 P_0 P_3 P_4 P_1 P_0 P_4



- Time quantum small → overhead because of context switching.
- Time quantum large → FIFO

* CPU must ensure
user mode is bel

:- Memory

* Physical Addres
* Logical Addres

* Base Register
* Limit Register

* Address Bindi
- The processe
to be broug
form the
- Mapping
space to

Source Program — Sym
20

* Logical

- At com
bindin
logica
- Howeve
are
- Now
Addre