Operating System:

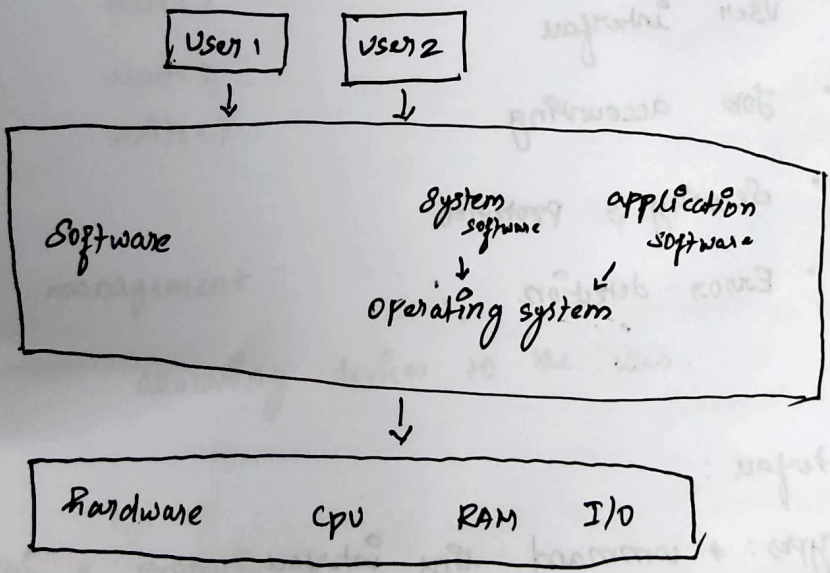
OS is a intermediate b/w user and the system hardware. it is assembly language.

Eg: linux, unix, mac, etc....

Types of Operating System:

- Batch os
- Time shared os
- Android
- iOS

Architecture of OS



Kernel

the process mode is User mode to kernal mode is called context switch.

Kernel is present in a OS which the gets the user requested proces from the user and the proces goes to the Kernal that proces is called System call.

Fuctions of OS:

- memory management
- Prosesor management
- Network management
- File management
- Device management
- User interface
- Job accounting.
- Security & Protection
- Error detection.

User Interface:

Types: * command line interface
* Batch line interface
* GUI

Process management: (Interaction b/w process and processor)

1. load the prgrm from secondary to main memory.
2. Processor checks the free processor
3. Process the execution

memory management:

types of MM:

* Primary * Secondary.

File management:

open()
close()
read()
write()

Device management:

allocating devices to the user.

Security & protection:

System call:

Types:

- Process control
- file management
- device management
- Information maintenance

Process management:

Process:

When a program is converted from high level language and enter into the compilation prouss after that it comes to the active entity that is program execution ~~phase~~ phase(~~passive entity~~) and prouss is a active entity.

Threads:

The thread is a unit of prouss. a program has many no of threads.

Prouss state:

- New
- Ready
- Running/ execution
- Exit / termination
- Waiting.

The prouss has to pass many stages that is called Prouss state

cpu utilization:

Interrupt:

- when the prouss enters into the cpu and if that prouss has some request to occupy a I/o divices it will go to Ilo and the main method prouss should will be in waiting state.

- when the sighen priority prouss comes to the prouss it will give the priority to that prouss.

Prouss memory:

4 section

- Text section(compiled prgm will be stored) [machine language]
- data section(static variable, global variable stored)
- Heap " (dynamic memory allocation)
- Stack " (local variable used in our prgm)

when the prouss goes to the waiting state and again to the running state the prouss should inform the memory to start the execution from the next prouss that called data are stored in heap.

Process control block:

for every execution ds will be maintained by OS that is called PCB.

once the work is over the PCB created for Process will be deleted.

components:

- Process ID
- State
- pointer (address)
- Priority
- Program counter (stores the nxt instruction)
- cpu register (temprory register)
- I/o information
- Accounting information

Scheduling:

- long term Schedule / job scheduler
- Short term Schedule / cpu scheduler
- medium term Schedule

job Scheduler:

New → Ready

Short term Scheduler:

Ready → running.

medium term Scheduler:

when the higher priority prouss enters into the cpu the prouss already there in the cpu was forced to go to the waiting state in that time medium term scheduler is used.

Shortus:

A mechanism to store and restore the state on content of cpu in pcb so that a prouss can be resumed from the same point at the later point - multitasking.

communication:

From the p_1 to p_2 it communicates to know the priority that is it is done between 1 prouss and another prouss is called inter prouss communication.

the communication within the process is called intra process communication.

mapping:

- one – one (if the thread is user mode and it requests to the kernal mode
- one – many
- many – one

Scheduling algorithm:

* FCFS –

SA

/ \

Priemtive Non Priemtive

catagears

Non priemtive algos :

continuous process it will not give a chance to other process until it finishes the whole process

Arrival time (the time at which the process arrives in the ready queue)

completion time (the time at which " " completes its execution)

Burst time (the time required by a process for cpu execution)

turn around time (time difference b/w completion time and the arrival time)

TAT = completion time - Arrival time

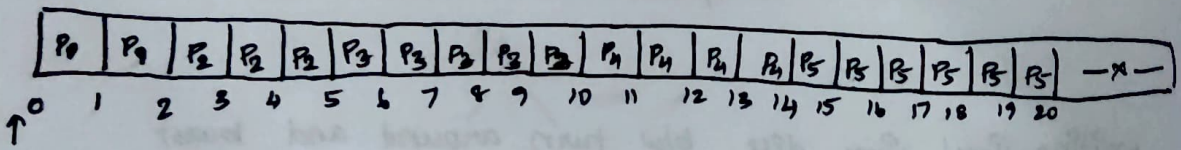
waiting time (time diff b/w turn around and burst time)

waiting time = Turn around time - burst time.

Eg:

consider the following process with the burst time (cpu execution time). calculate Average wait time and average turn around time.

Pid	Arrival time	Burst time / cpu execution time
P_1	0	2
P_2	1	3
P_3	2	5
P_4	3	4
P_5	4	6



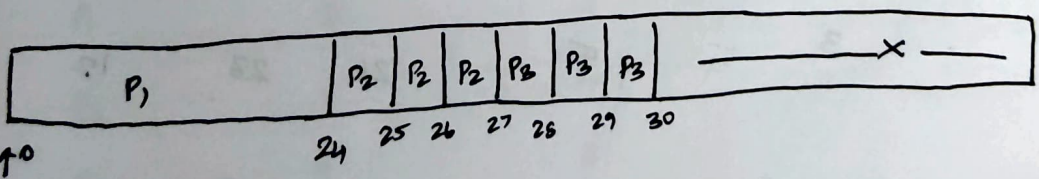
completion time	Turn around time	waiting time
P_1 - 2	$P_1 = 2 - 0 = 2$	0
P_2 - 5	$P_2 = 5 - 1 = 4$	1
P_3 - 10	$P_3 = 10 - 2 = 8$	3
P_4 - 14	$P_4 = 14 - 3 = 11$	6
P_5 - 20	$P_5 = 20 - 4 = 16$	10
	41	20

Average waiting time = 4

ATT = 8.2

In the same order with the arival time 0 and given burst time. calculate the average waiting time and average turn around time.

Pid	Arrival time	Burst time	CT	TT	WT
P_1	0	24	24	24	0
P_2	0	3	27	27	24
P_3	0	3	30	30	27
				81	51



Average TT = 27

AWT = 17

define to find the true good :

$$= \frac{\text{Total no of job}}{\text{total time taken}}$$

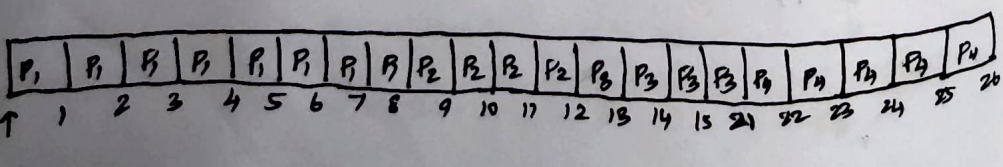
$$= \frac{3}{30} = 0.1$$

0.1 jobs per time unit

consider the process P_1, P_2, P_3, P_4 given in the table, arrives for execution in the same order with given arrival time & burst time

Pid	arrival time	burst time	CT	TT	WT
P_1	0	8	8	8	0
P_2	1	4	12	11	7
P_3	2	9	21	19	10
P_4	3	5	26	23	12

calculate average wait time, average turn around time and true good.



AWT = 15.25

ATT = 8.75

True good = $\frac{4}{26}$ = 0.1538

jobs per unit second.

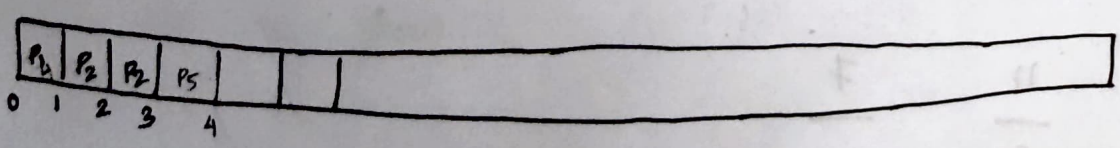
* Shortest job first:

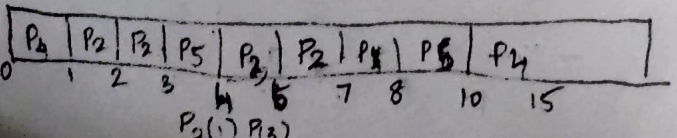
The prouss which has loco burst time will occupy the cpu first

2) more prouss having same burst time and same it go uses FCFS algorithm.

consider a set of five prouss whose arrival time given below

Pid	arrival time	burst time
P_1	3	1
P_2	1	4
P_3	4	2
P_4	0	6
P_5	2	3





Consider the set of two Process whose arrival time and burst time are given.

Pid	Arrival time	burst time
P_0	0	7
P_1	2	4
P_2	4	1
P_3	5	4

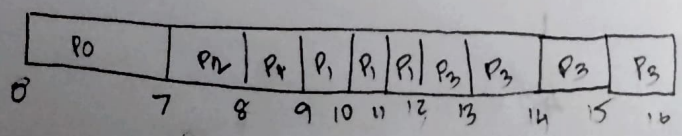
find ATAT and AWT using shortest job first

⇒

i) Non Premetive version.

ii) AT

iii) BT



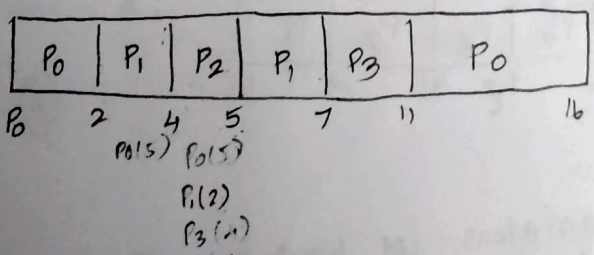
CT	TT	WT
7	7	0
12	10	6
8	4	3
16	11	7
	32	16

ATT = 8 jobs / time units

AWT = 4 jobs / time unit

Pid	Arrival time	burst time	CT	TT	WT
P_0	0	7 - 2	16	16	9
P_1	2	4	7	5	1
P_2	4	1	5	1	0
P_3	5	4	11	6	2
				28	12

i) Preemtive



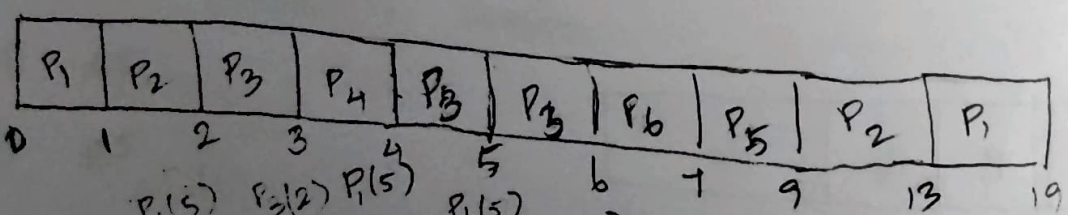
ATT = 1 Per unit

AWT = 3 Per unit

Consider the set of 5 process whose arrival time and burst time are given.

P id	arrival time	burst time	CT	TT	WT
P_1	0	7	19	19	12
P_2	1	5	13	12	7
P_3	2	3	6	4	1
P_4	3	1	4	1	0
P_5	4	2	9	5	3
P_6	5	1	7	2	1
				43	24

Shortest job first, Pumtive:



$P_1(5)$ $P_2(4)$ $P_3(2)$ $P_4(1)$ $P_3(1)$ $P_1(5)$ $P_3(1)$ $P_6(1)$ $P_5(2)$ $P_2(4)$ $P_1(5)$

Unit-2

Process synchronization: action of a process

- Independent process (if it never affects other process)
- cooperative process (action of a process affecting other process)

Eg: Bank.

monitoring the shared variable/ process is the job of process synchronization.

int share = 5

P1

```
{ int a = share;
  a++;
  share = a;
}
```

(1)

P2

```
{ int b = share;
  b--;
  share = b;
}
```

(2)

Critical Section: (hold the code of shared variable/ process)

Syntax

```
do
{
    entry section
        critical section
    exit section
        Reminder sect
} while (T);
```

If the we have 2 process p1 and p2 they will share same critical section. and if the one process executes going on if p2 tries to enters onto the execution it will not allow that process. so, that process will not run simultaneously it is called mutal-exclusion.

requirments:

- mutual exclusion (ME) (one process at a time)
- Progress (Prog) (it should not affect the other process and run smoothly)
- Bounded waiting (BW) (fixing the limit)

There are 2 broad catagi for solu:

* Software based solu
* hardware based solu

Software based solu (Peter Sierce algo)

focus on condition what we are going to work

hardware based solu &

focus on counter, lock.

why we need synchronization: process

- for monitoring the shared process
- For not affecting the other processes which

In what basis the solution with to the process synchro is divided into Software based solu and hardware based solu?

Two procss solu: P0

```
do {
    while (turn != i); // entry sec
        CS
    turn = j; // exit sec
        RS
} while(1);
```

0 != 0 -> self loop

if it is false it will enter the execu

if it is true it will wait

its false so it enters the execu

P1

```
do {
    while (turn != 0);
        CS
    turn = 1;
        RS
} while(T);
```

here turn is 1 so it became true so the process P1 will be waiting

consider the following 2 process synchronization solu

P0

Entry: loop while (turn == 1); false it enters CC

CS

exit: turn = 1; ✓ after finishing

P_1 ↓ it goes to P_1

Entry: loop while (turn == 0); false

CS it enters CS

exit: turn = 0; again the loop will go on

The shared variable turn is initialized to 0. which one of the following was true.

a) violates mutal exclusion requirment

b) violates Progress requ

c) violates Bounded wait requ

Peter Science sense solution: (it is software based solu)

P_i

```
do {
    flag[i] = true;        (boolean)
    turn = j;
    while( flag[j] && turn == [j] );
        CS
```

here turn indicates which process should enter the cs

flag indicates whether the pro is ready to enter its cs if flag is true

Note:

- it is a bumble ny algorithm gives chance to the nxt process

Pi

```
do{
entry { flag[i] = true;
        turn = j;
        while(flag[j] && turn == j);
      }
      CS
exit: flag[i] = false;
      RS
} while(true);
```

Pj

```
do{
entry { flag[j] = true;
        turn = i;
        while flag[i] && turn == i;
      }
      CS
exit: flag[j] = false;
      RS
} while(true);
```

Test and set algorithm:

handle 0 and 1

initially the lock set to false

```
boolean testandset(boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```

P1

```
do {
    while (testandset (&lock));
        CS
    lock = FALSE;
        RS
} while (true);
```

P2

```
do {
while (testandset(&lock));
    CS
lock = FALSE;
RS
} while (true);
```

Swap algorithm - (Hardware solu)

initially the lock and key value are false.

```
boolean lock;
individually key;

Void swap (boolean *a, boolean *b)
{
- boolean temp = *a;
    *a = *b;
    *b = temp;
}
while (1) {
    key = True;
    while (key)
        swap (&lock, &key);
            CS
    lock = = False;
```

RS

}

unlock and lock algo

lock and key takes as false ~~as~~ intially

```
boolean lock;
Individual key;
Individual waiting[i];

while(1)
  waiting[i] == True;
  key = TRUE;
  while (waiting(i) && key)
        key = test and set( );
  waiting[i] = false;
     CS
  j = (i+1) %. n;
  while (j != i && ! waiting[jJ));
        j = (j+1) %. n;
    if (j == i)
        lock = FalsE;
    else
        waiting[jJ = False;
        RS
}
```

Semaphore :

It handles the synchronization

* Binary
* counting

2 automatic function

* wait * signal

It holds integer value. It is used to assign instances to thread. so it works in kernal mode. it works on multi programming environment.

```
wait (sem, P)
{
  while (S<=0);
  S--;
}
```

```
signal (sem, V)
{
  S++;
}
```

Deadlock :

Process waits for resources which is already been assigned to some other process.

It runs in a circular wait.

If some process ~~p1~~ ~~hold~~