OS concepts By

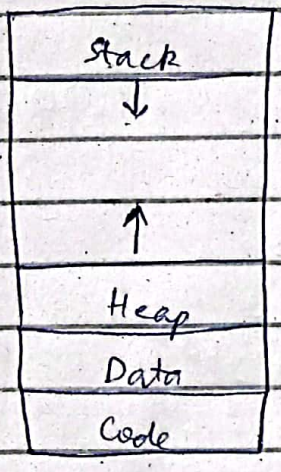
13th Feb, 2024 lecture 1

Process ?

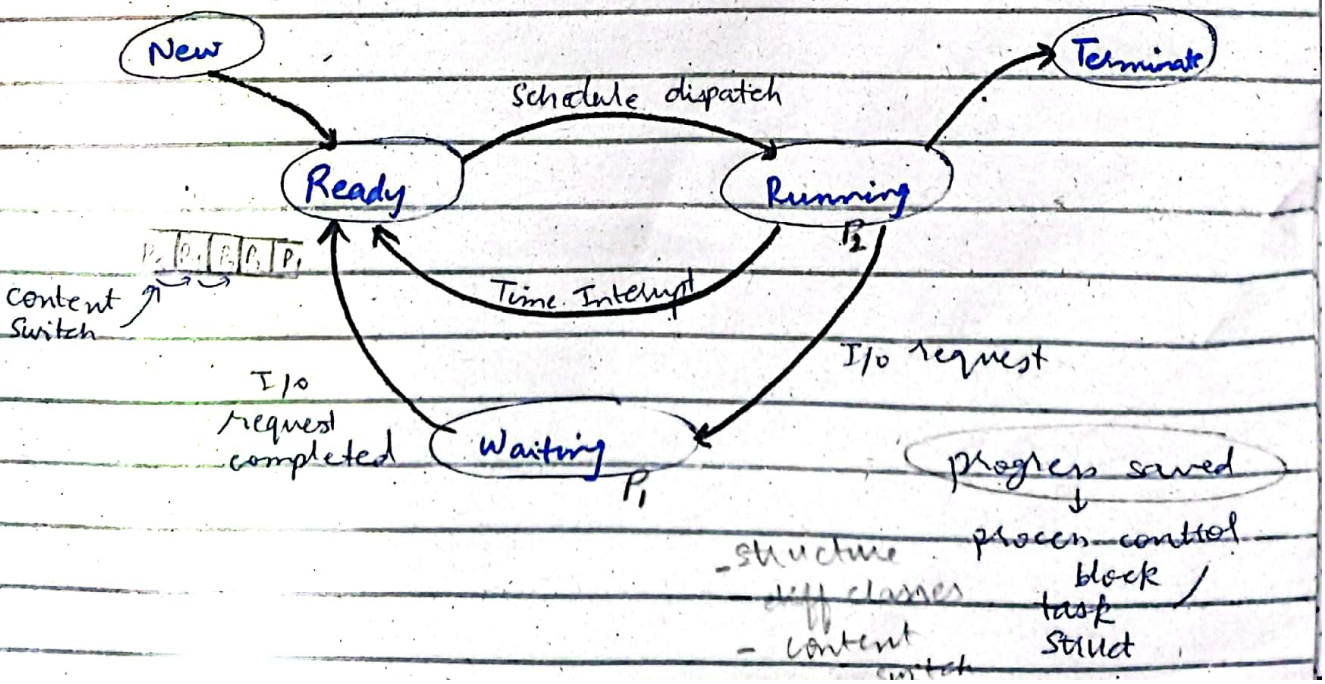
→ Steps

Chapter 3 Process

15th Feb, 2024 lecture 2



Process State Diagram

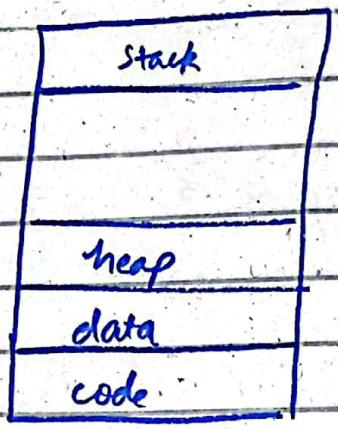


21st Feb, 2024 lecture 3

Fork()? for creating process

```
void main()
{
    int pid, x, y;
    pid = fork();
    printf("pid =", pid);
}
```

p - pid = 3276
c - pid = 0



3 1 = 3

%d x %d = %d

a, y, a*y

wait(NULL);

↳ child 1st
Parent 2nd

execlp

Process Scheduling
Queues

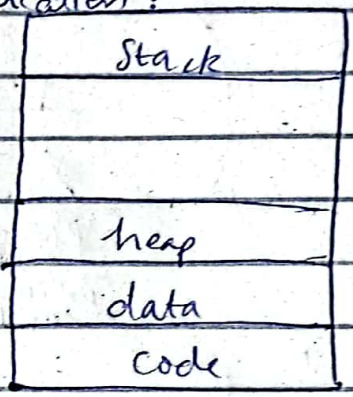
Scheduler

22nd Feb, 2024 Lecture 4

Inter Process Communication

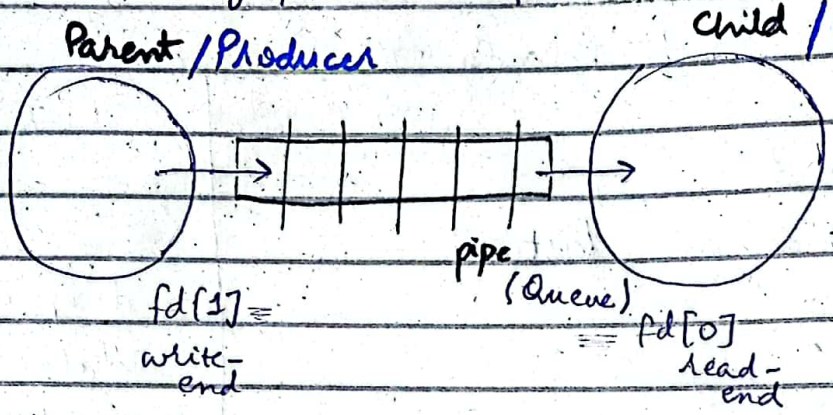
Ways for Inter Process communication:

- Pipes. ✓
- Named pipe ✓
- Shared memory (fastest)
- Msg Queues ✓
- Sockets. (fast) → logical identifier




```
int main()  [combo: ipaddress & port #]
{
  int x = 10, y = 0, pid;
    pid = fork();
  if (pid == 0)
  {
     x = 20;
     y = y+1;
  }
  else
  {  Wait(NULL);
     x = x+10;
     y = y+1;
  }
  cout << " x = " << x << endl;
  cout << " y = " << y << endl;
}
```

Pipe (✗ duplicate)
(descriptive of pipes ✓ duplicate)



file descriptor
↳ fd

```
int (pid == 0)
{
    char buffer [10];
    Read (fd[0], rbuffer,
      Size of rbuffer);
}
else
{
    char buffer [10];
    Write (fd[1], buffer,
      Size of buffer);
}
```

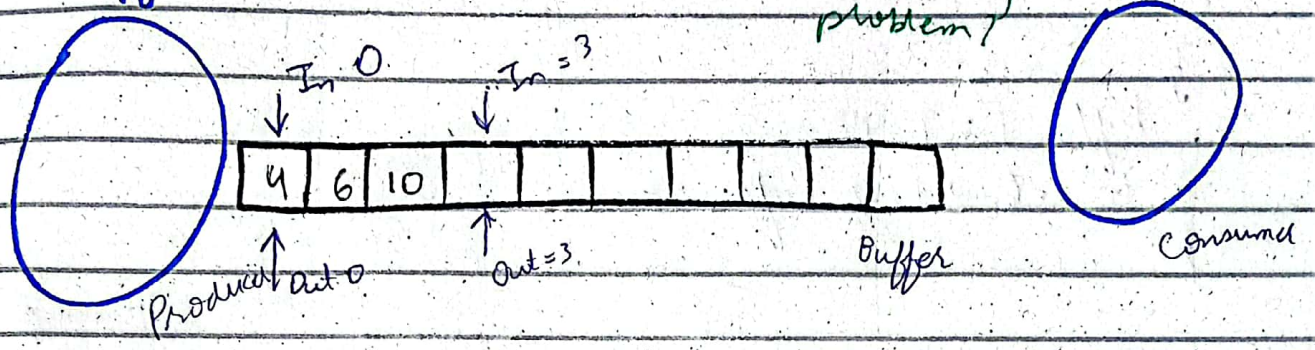
```
int fd[2];
pipe (fd);
→ Write (fd[1],
buffer, size of buffer)
→ Read (fd[0],
rbuffer, size of rbuffer)
```

28th Feb, 2024 Lecture 5

Producer - Consumer Problem

What is consumer problem?

How to solve shared memory problem?



pointing to item to be produced

```
while (In == out)
{
    // busy loop
}
```

```
buffer [in] = item
in = in + 1
```

```
buffer [in] = item
in = in + 1
```

```
item = buffer [out]
out = out + 1
```

```
in = (in + 1) % buffersize
```

↳ queue ended / circular queue

```
While ((in+1) % buffersize == out)
{
    // busy loop
}
```

```
While (1)
{
  while ((in + 1) % buffersize == out)
    {
    }
      buffer [in] = item
      in = (in + 1) % buffersize
}
```

In

```
While (1)
{
  while (In == out)
    {
    }
      item = buffer [out]
      out = out + 1
}
```

Out

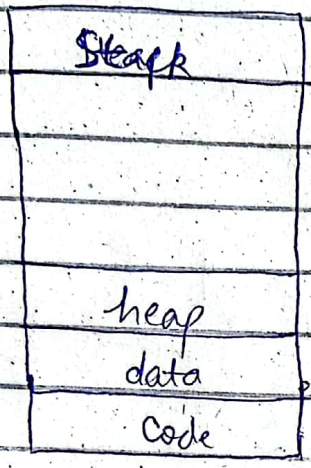
Shared Memory

*ptr+8

29th Feb, 2024 Lecture 6

1 ques must from threads

Multithreading



Stack different for every thread.

Why threads?

→ easier.

→ light weight.

→ can be manged under single process.

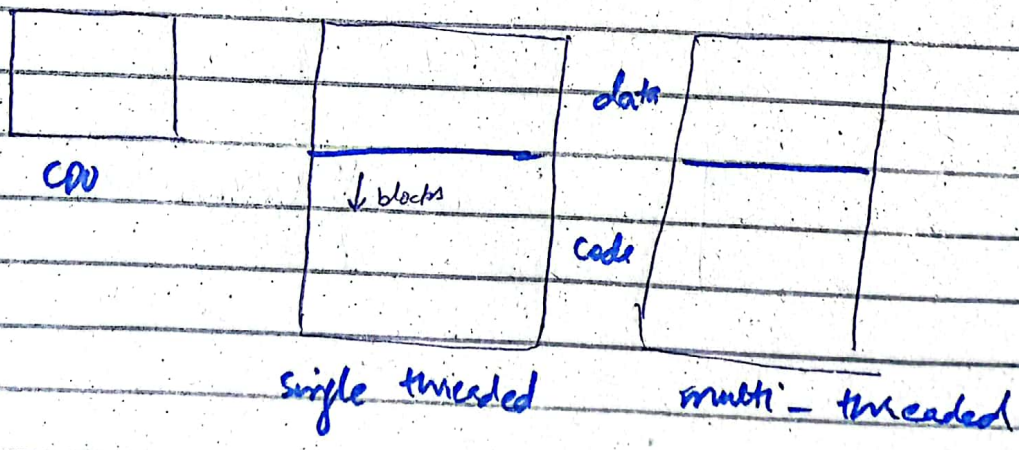
→ can be created easily.

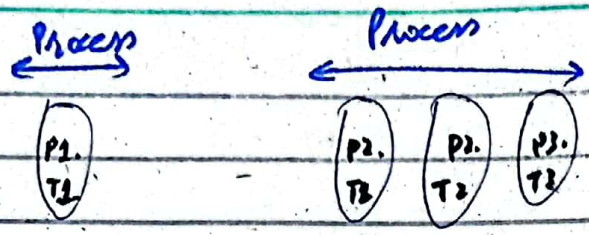
→ communication easy.

→ no need for pipe (can be used but without pipe easier to handle)

7th March, 2024 Lecture 8

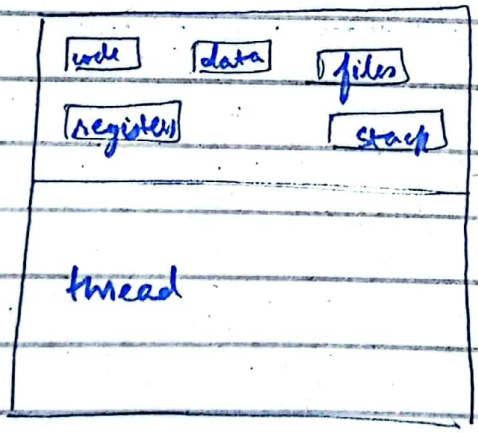
Threads and Thread usage





Schedulable Entities
We can select one of them & run.

```
function 1()                    thread 1
{
}
function 2()
{
}
main()
{
thread_create (function 1)
thread_create (function 2)
thread_create (function 1)
}
```

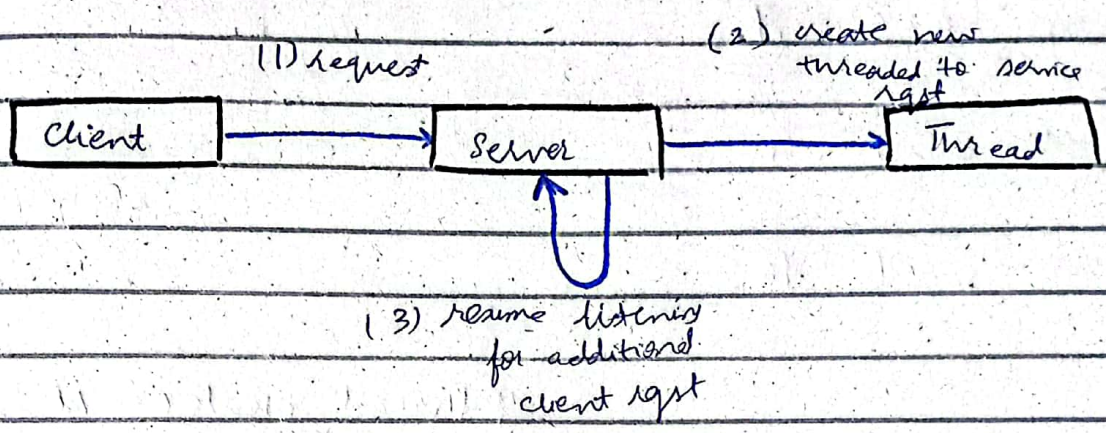


thread → separate stack

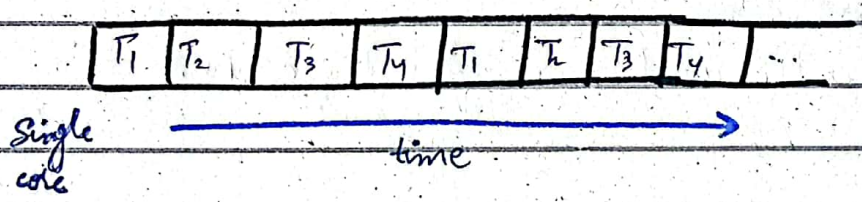
code, data, files, heap

shared in multithreaded process.

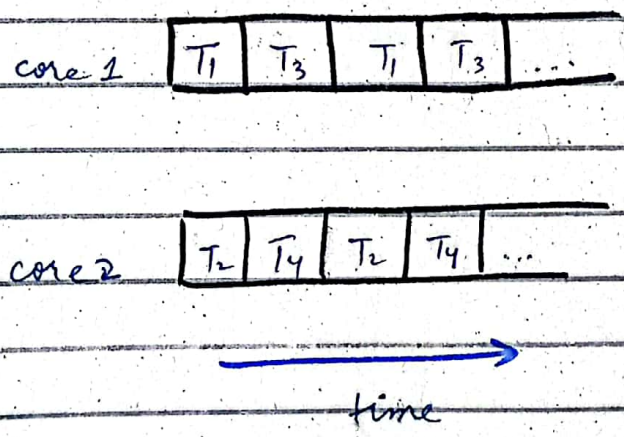
Multithreaded Server Architecture



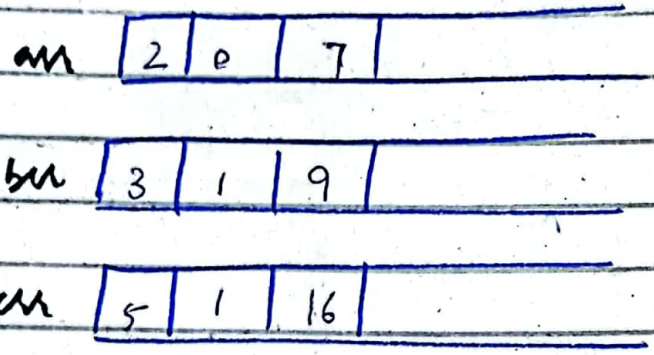
Concurrent Execution on a single-core system



Multi-core



13th March, 2024 Lecture 9



pthread_create(&tid1, NULL, thread1, NULL
Pthread_join(tid1, NULL);

Threading support

User level libraries

Library creates and manages threads.

User level implementation

Kernel

Kernel creats and manages threads

Kernel level implementa

- Many-to-one model. * (1 ques)
- One-to-one model.
- Many-to-Many model.

14th March, 2024 Lecture 10

Chapter 5 Synchronization

Threads / Programm

```
Pthread_create (&tid1, NULL, inc, NULL);
Pthread_create (&tid2, NULL, dec, NULL);
Pthread_join (tid1, NULL);
Pthread_join (tid2, NULL);

void *inc (void *arg)
{
    for (int n=0; n<1000; n++)
        count = count + 1;
}

void *dec (void (*arg)
{
    for (int n=0; n=1000; n--)
        count = count - 1;
}
```

Running in preempted simultaneously

21st March, 2024 **lecture 12**

Synchronization Hardware

→ Test and Set

→ Swap

XCHG

```
Test and Set (bool *target)
{
    bool rv = *target;
    *target = TRUE;
    return (rv);
}
```

lock | T...

False

do {

while (TestAndSet(&lock))
{
// busy loop
}
// C.S
lock = FALSE;
} while (1);

do {

while (TestAndSet(&lock))
{
// busy loop
}
// C.S
lock = FALSE;
} while (1);

entry section

exit section

Swap

```
Swap (bool *a , bool *b)
{
bool temp ;
    temp = *a ;
    *a = *b;
    *b = temp;
}
```

T_0 a = 1	a = 1	a = 1	a = 1
T_1 TAS( )	TAS( )	TAS( )	TAS( )

Lock | False

Thread 0

```
do{
    Key = True
while ( key == True)
{
    swap (key, lock);
}
// C.S
    lock = False ;
}
while (1);
```

Thread 1

```
do {
    Key = True
While ( key == True)
{   entry section
    swap ( key, lock);
}
// C.S
    lock = False;
}
while(1);
```

exit section

→ Cannot hold bounded waiting

Even if you are in busy loop CPU cycles will be consumed and will go wasted.

→ Issue with
- Test&set
- Swap (XCHG)

signal → if nothing blocked increments value of s by 1

Signal unblocks any blocked process

Semaphores → every semaphore have a queue in sema queue

S = 1

wait → decrements value by 1

Th0 — Thread 0 / P_0

Th1 — Thread 1 / P_1

Th2 — Thread 2 / P_2

```
wait (s);          wait(s);          wait (s);
// C.S             // C.S            // C.S
signal (s);        signal(s);        signal (s);
```

21th March, 2024 **Lecture 13**

P_0	P_1	P_2	P_3
wait (S)	wait (S)	wait (S)	wait (S)
//acces printer	//acces printer	//acces printer	//acces printer
signal (S)	signal (S)	signal (S)	signal (S)

Writer 2

Reader ✓

→ any Writer come when any reader reading block writer

Writer 1 | DATA SET | Reader 1

Reader 2

→ When any writer writing don't allow any reader or writer

Semaphore Mutex = 1
wrt = 1

int readcount = 0

```
do {
    wait (wrt);
    // writing is performed
    signal (wrt);
} while (1);
```

→ Writer

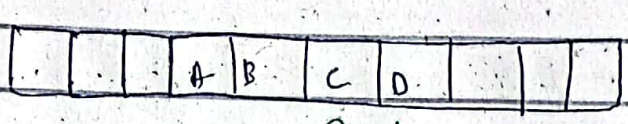
```
do {
    wait (Mutex);
    read count ++;
    if (read count == 1)
        wait (wrt);
    signal (mutex);
    // reading performed
    wait (Mutex);
    readcount --;
    if (read count == 0)
        signal (wrt);
    signal (Mutex);
} while (1);
```

Reader

28th March, 2024 Lecture 14

Bounded buffer problem

Empty = 6
Full = 4



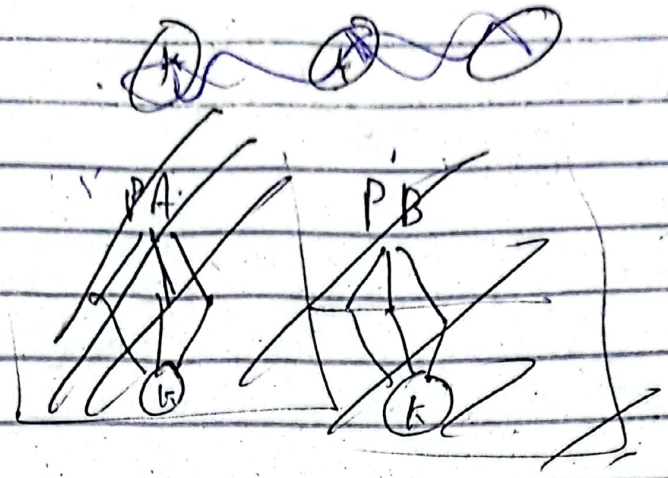
Buffer

Producer thread

```
do{
    wait(empty);
    wait(Mutex);
    //
    Signal(Mutex);
    Signal(Full);
}while(1)
```

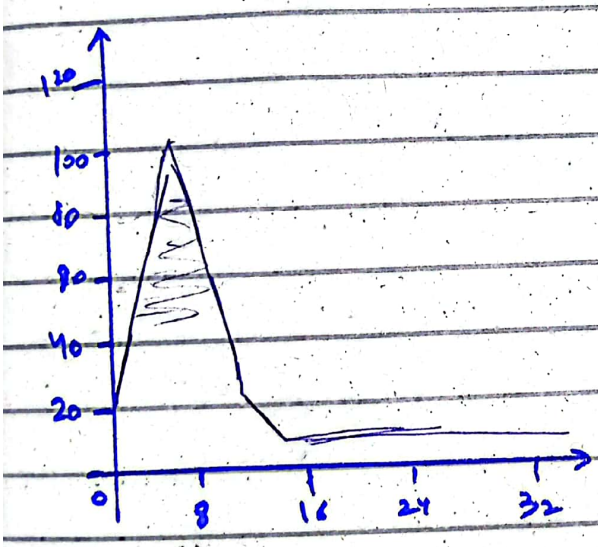
Consumer thread

```
do{
    wait(full);
    wait(Mutex);
    //
    Signal(Mutex);
    Signal(empty);
}while(1)
```



17th April, 2024 **Lecture 15**

Chapter 6 "CPU Scheduling"

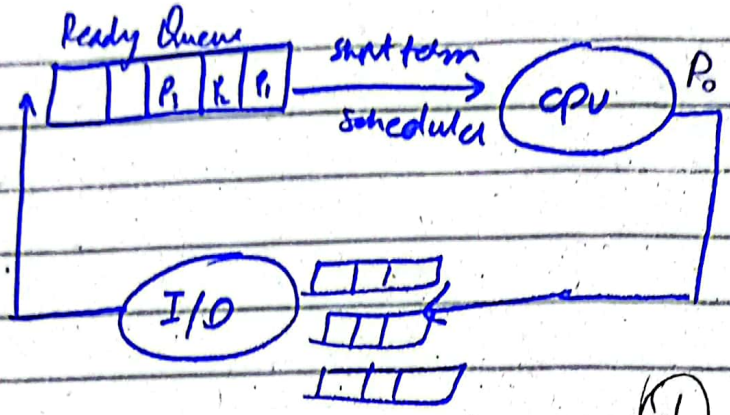


load
store
I/O request
} CPU Burst

I/O request
I/O complete
} I/O Burst

load
store
} CPU Burst

I/O request
I/O request
I/O complete
} I/O Burst



preempt

(1) Running to waiting state

Scheduler need to take decision

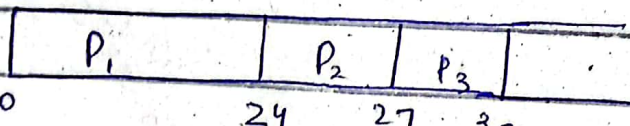
(2) Waiting to Ready state needs b/c of priority (to take decisions)

(3) Running to Ready state

FCFS (First come First Served)

Process	CPU Burst
P_1	24
P_2	3
P_3	3

time



avg waiting time ↓ algo ✓ best

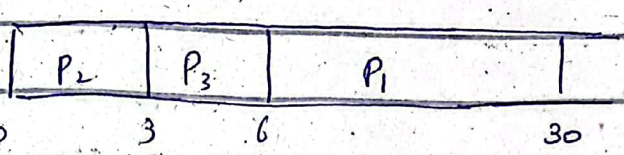
$$= \frac{0+24+27}{3}$$

$$= \frac{51}{3} = 17 \text{ ms}$$

↓

avg waiting time

SJF (Shortest Job First)



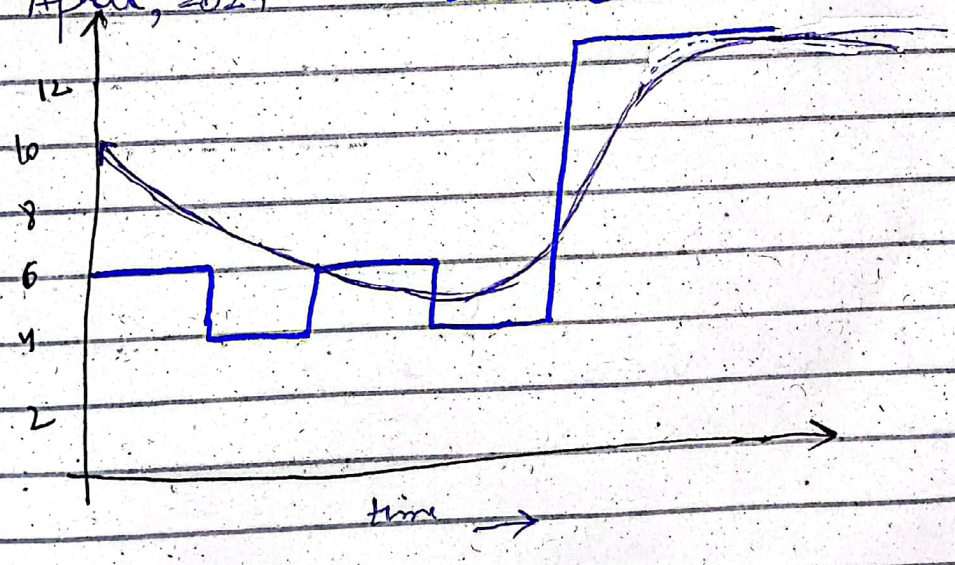
$= \frac{6+0+3}{3} = 3$ ms

Exponential averaging (to predict next CPU Burst of Process).

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

- τ_{n+1} — Predicted value
- t_n — current CPU burst
- τ_n — History values

19th April, 2024 — Lecture 16



$= 0.5 \times 6 + (0.5) \times 10$

$= 3 + 5$

$\tau_{n+1} = 8$

$\tau_{n+2} = 0.5 \times 4 + (0.5) \times 8$

$= 2 + 4$

$\tau_{n+1} = 6$

$T_{n+3} = 0.5 \times 6 + 0.5 \times 6$
$= 3 + 3 = 6$

$T_{n+4} = 0.5 \times 4 + 0.5 \times 6$
$= 2 + 3 = 5$

$T_{n+5} = 0.5 \times 13 + 0.5 \times 5$
$= 6.5 + 2.5$
$= 9$

Exponential Averaging

SRJF (Shortest Remaining Job First)

Priority Scheduling

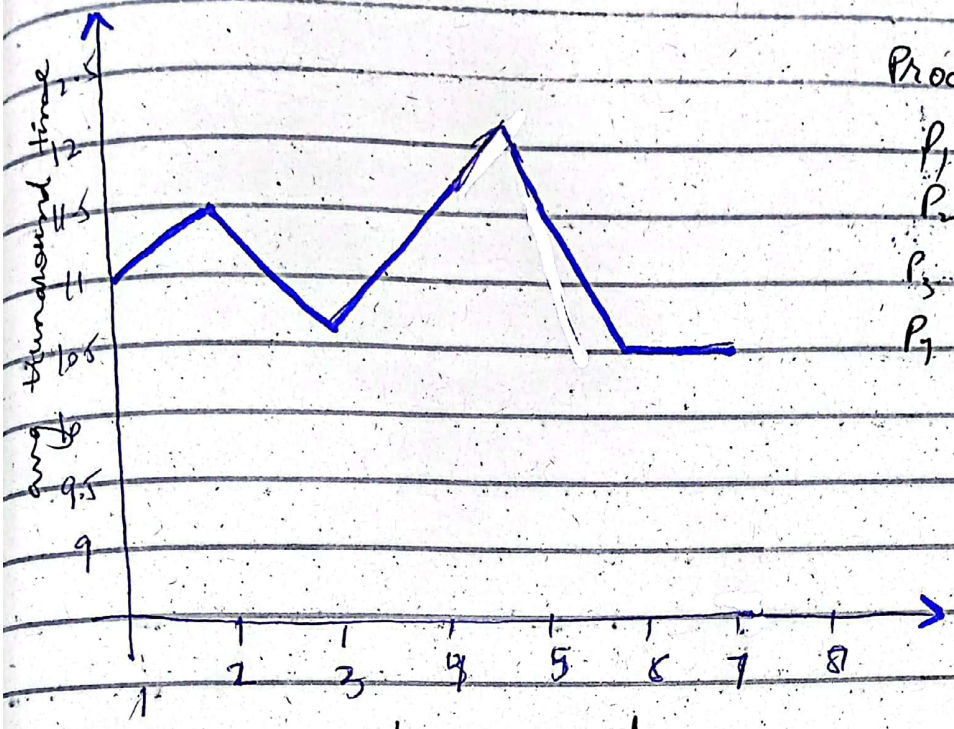
Round Robin (RR)

- CPU scheduling is based on RR.
- Time Quantum given to RR by CPU
- q large = FIFO
- q small =

Context Switch Time

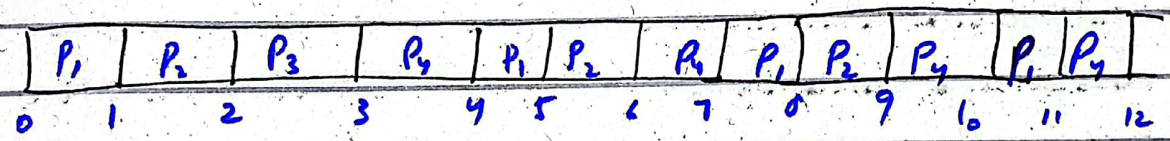
Turnaround time Varies with the Time Quantum

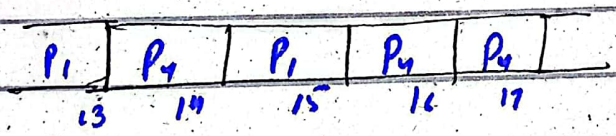
Time Quantum



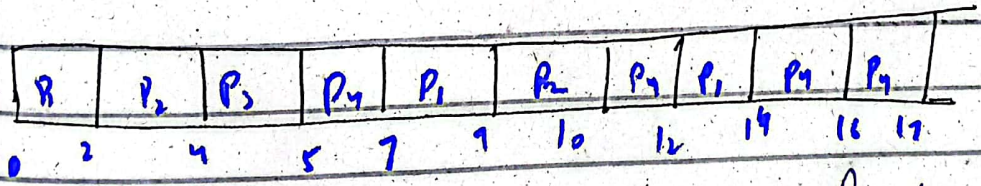
Process	time
P_1	6
P_2	3
P_3	1
P_4	7

$Q=1$





$Q_s=2$



$P_1 = (15-0) = 15$

$P_2 = (9-0) = 9$

$P_3 = (3-0) = 3$

$P_4 = (17-0) = 17$

$$= \frac{15+9+3+17}{4}$$

$$= \frac{44}{4} = 11.0$$