

Software Application

→ Web Browsers, Games, Microsoft office, task manager etc.

Operating System

→ A trusted software that works as bridge between hardware and user applications.

Hardware (CPU, HDD, etc)

- Resource Management
- Security of Data
- Increasing Performance

Virtualization of CPU → Mid

Virtualization of Memory → Final

Concurrency → Mid

File and storage System → Final.

Concurrency → Working in parallel

1. Process, Process state

2. Context switch, System call

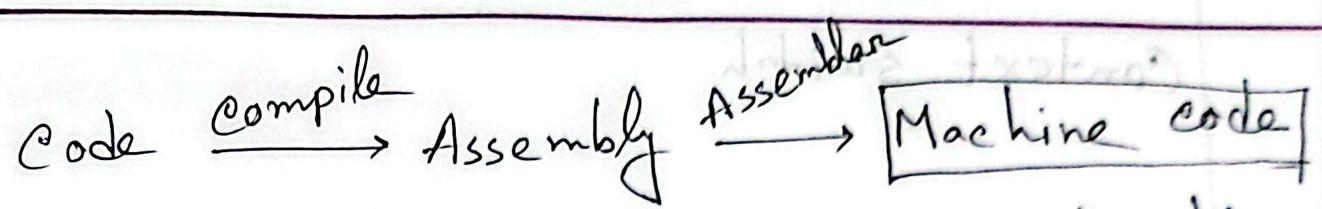
3. Process Creation, (fork, wait, exec
system calls)

4. Scheduling (FCFS, SJF, STCF, RR, MLFQ)

5. Concurrency (Basic idea on threads,
locks, synchronization,
producer-consumer problem)

• function working on specific base address

27/07/20



Statement

Assembly

Machine code

Instruction

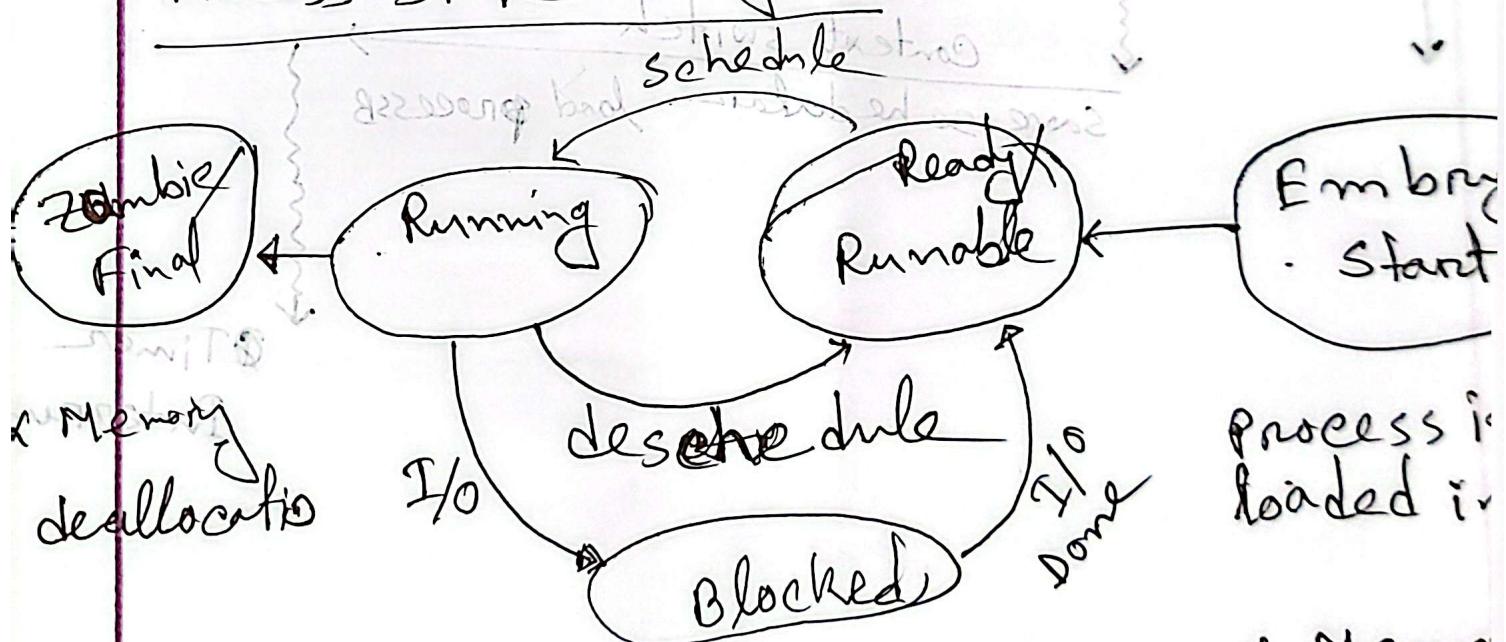
$a = 10$

$\$ t0 = 10$

STORE (a , $\$ t0$)

Registers \rightarrow Computations (Mathematical Logics)

Process State Diagram

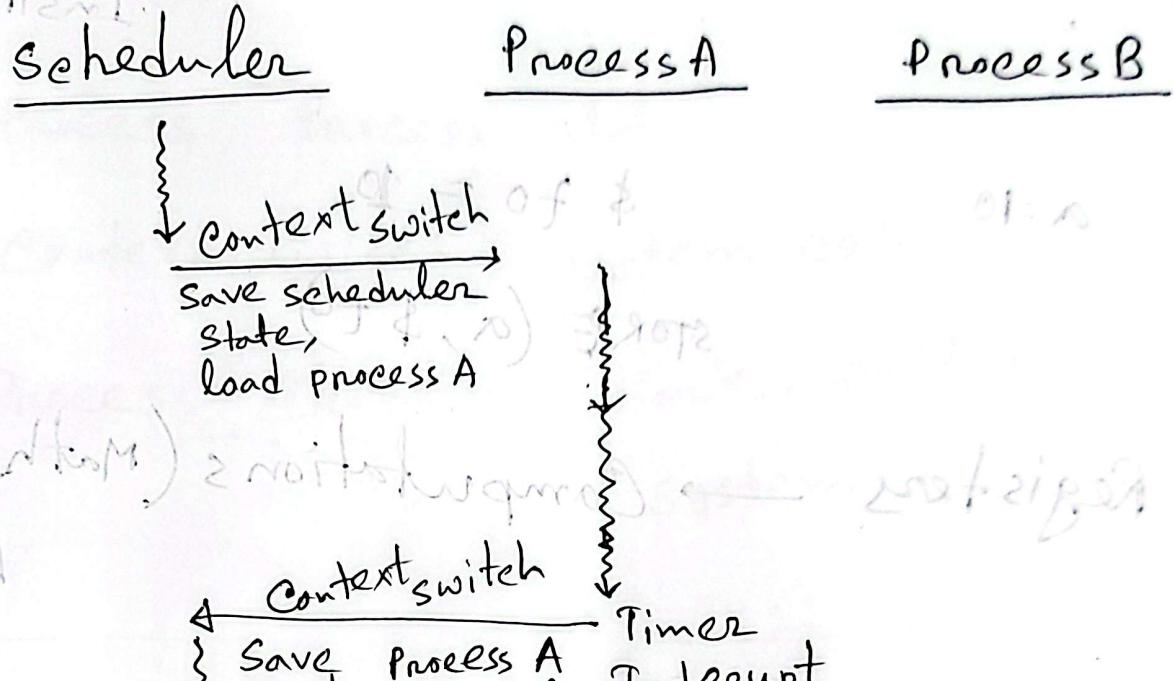


process is loaded in

* Memo

A1

Context switch



Chapter-5

Process Creation

`fork` → Creates an exact copy of the calling process, process id, memory & file descriptions are different.

`wait` → can wait for a process or child processes.

`exec` → Used to overwrite the contents of child process to convert it into a new specific process.

`fork` → Used to create a copy of the current process. It creates a copy of the current process and then creates a new process. Both processes share the same memory space.

Memory X

```
int a = fork(); // a=6  
if (a == 0) {  
    printf("Child");  
} else {  
    printf("Parent");  
}
```

P1.C Pid = 5

1. Calls fork [From A]

2. OS creates an exact copy of running process

Existence of B starts here

A

2.1

Memory Y

3. return value of fork is set

3.1 0 for the child

process

3.2 pid of child process for the calling process

```
int a = fork(); // a=6  
if (a == 0) {  
    printf("Child");  
} else {  
    printf("Parent");  
}
```

Child process

Pid = 6

4. Return from fork

[Both for A & B]

Process B run first or (first)

Both process go by run into

Overview of Process Creation

`fork (void) {`

~~↳ stores from parent~~
Copy memory of parent to child

initialize pid for child

set return value of child, a0=0;

~~↳ stores from parent~~

~~↳ a0 → register of child proc~~

return pid; // pid → pid of child

~~↳ stores from parent~~

`a = fork ()`

↓ Assembly

1. Call fork
2. execute fork → child process created
3. return from fork.

↳ Parent returns from fork
child returns from fork

True || fun()

↳ Does not execute

False || fun()

↳ will be executed

blinds not big, will be

True & & fun() and no big blinds

blinds big & big will be

big

False && fun()

↳ will be executed

↳ will not be executed

() short ()
blinds

short blinds

big blinds & short blinds

big blinds & short blinds

not many

big blinds

not many

big blinds

```
if (fork() == fork()) {
```

↓ compilation

1. \$t0 = fork()
2. \$t0 == 0;

```
$t1 = fork()
```

```
$t2 = $t0 || $t1
```

```
if $t2 == 0;
```

jump to label B

jump to label A

```
If (fork() && fork()) {
```

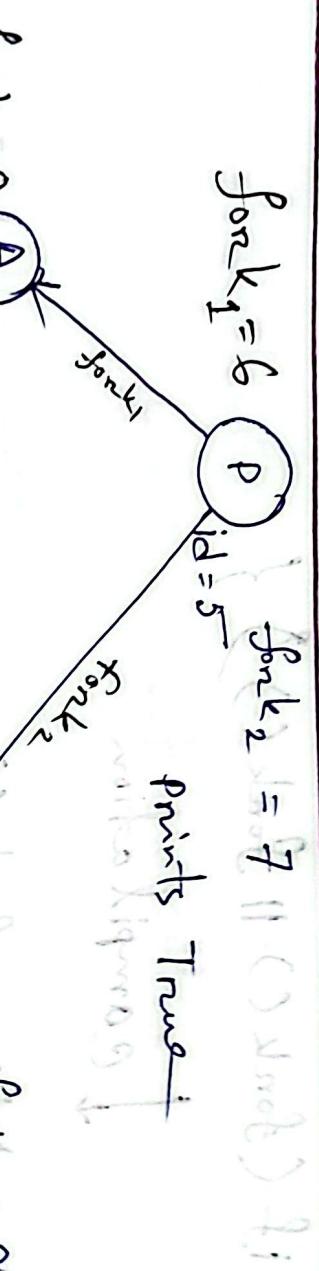
```
printf("True\n");
```

```
} else {
```

```
printf("False\n");
```

}

Draw the process tree
and determine the output.



$$\text{funk}_i = 0 \quad (\text{A})$$

$$\begin{aligned} f_{\text{rank}_1} &= 0 \\ f_{\text{rank}_2} &= 0 \end{aligned}$$

Prints False

Output:

True
False
False

2. Head of government

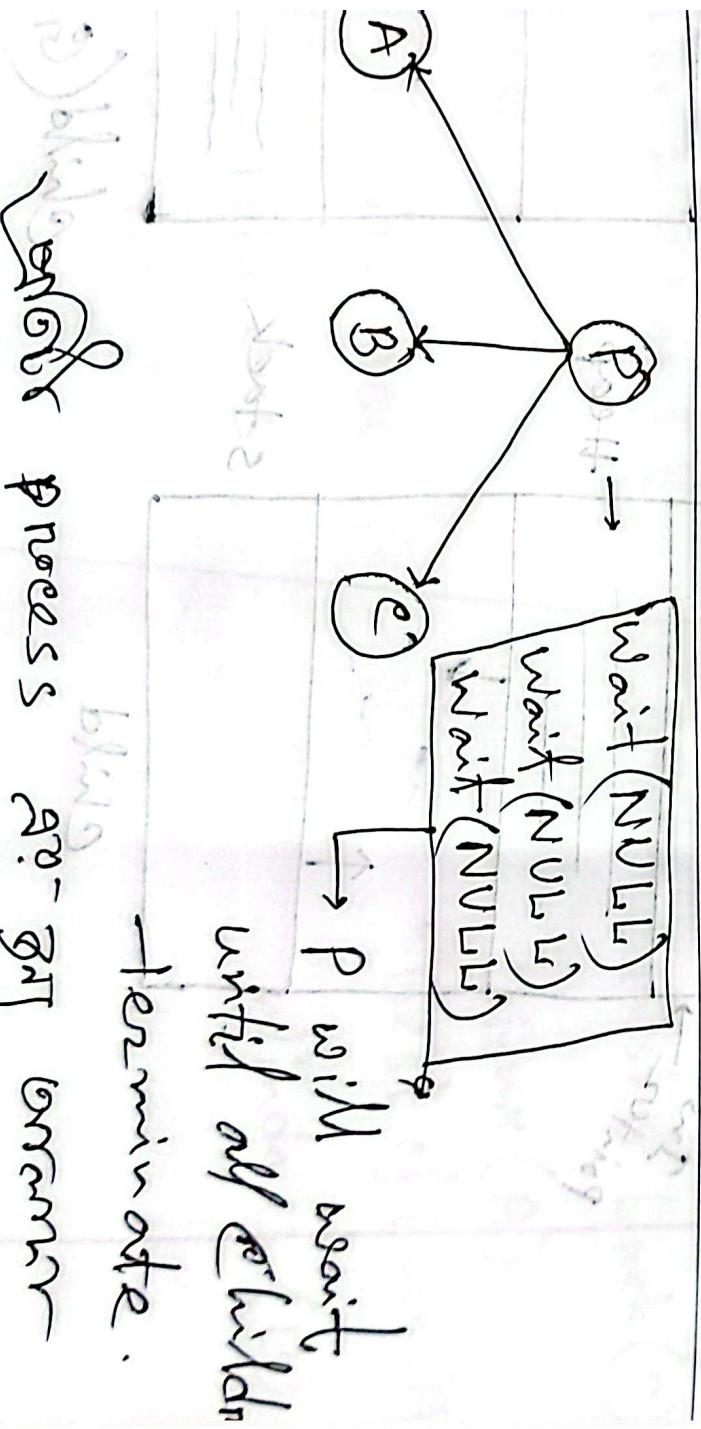
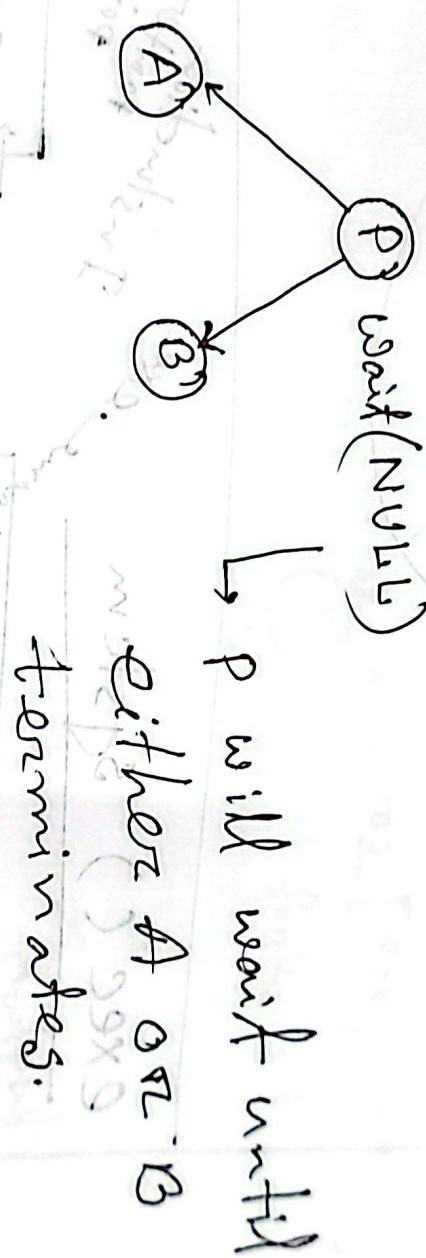
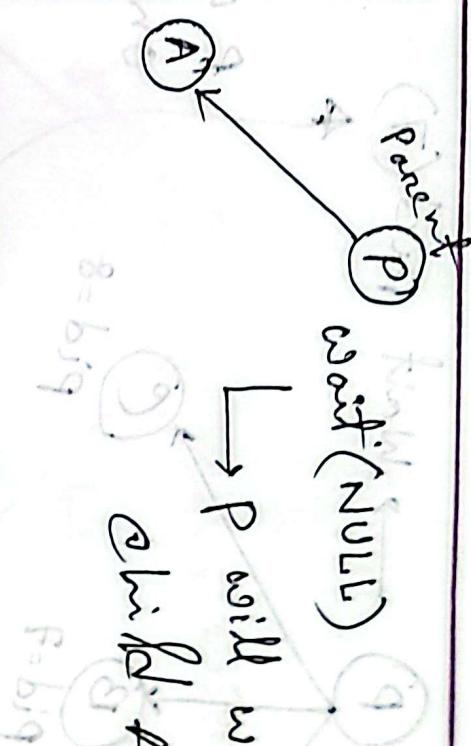
22 (cont'd) & 22 (cont'd) 88

Brown & Lowne, New York.

46

new & old strings

03/08/2025



wait call.

longer process go on forever

blowing

process go on forever

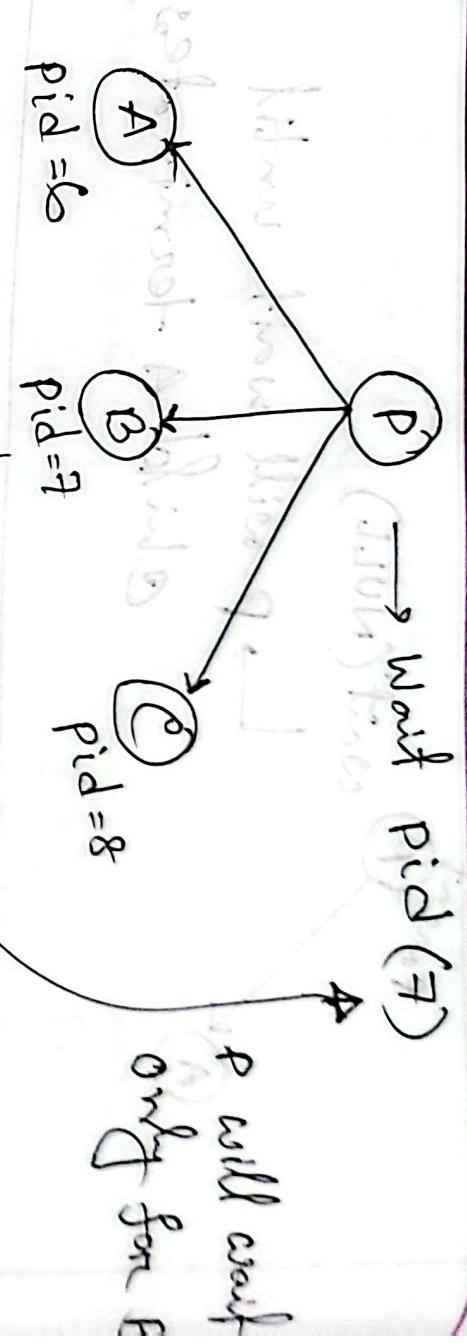
wait call.

longer process go on forever

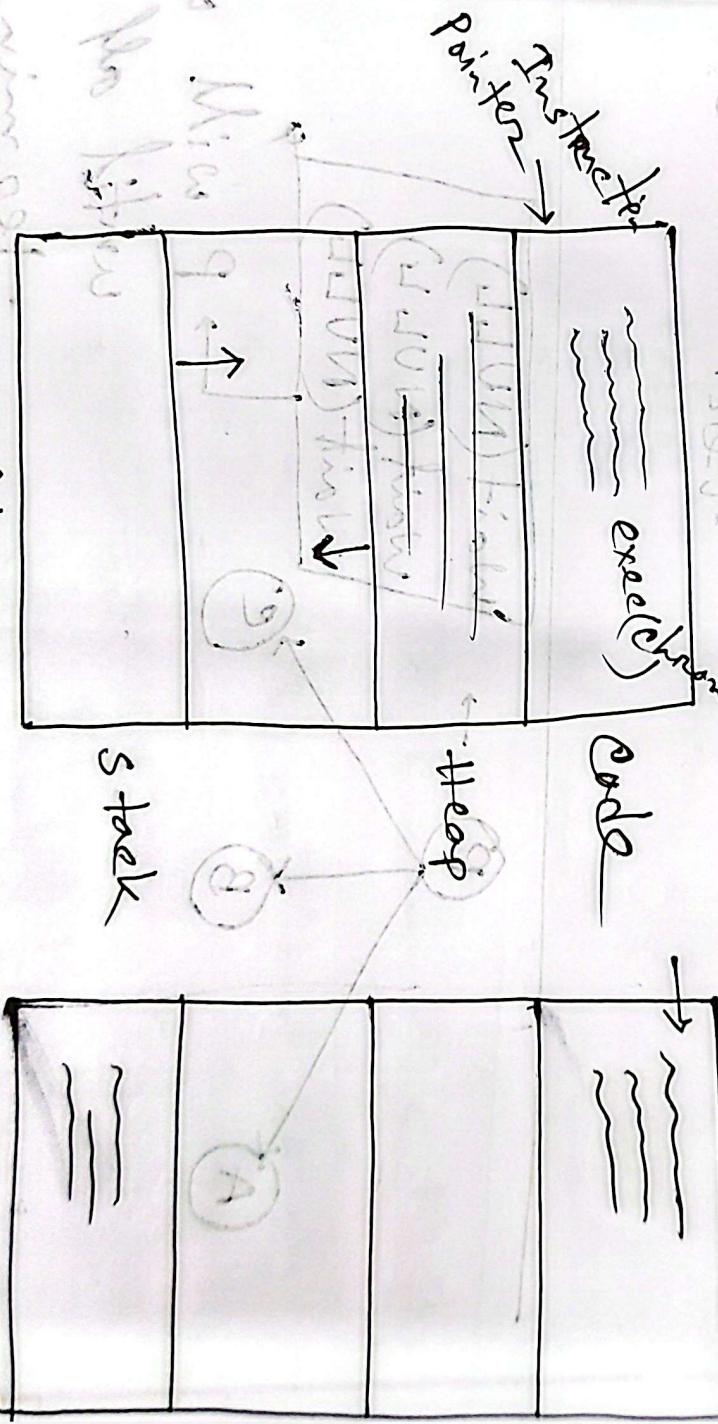
blowing

process go on forever

wait call.



exec() system call
Instruction pointer points to first instruction



none
The -gB 220014
Child(Go back to main)
Main

Q 5 243

(A) (B) (C)

$c_1 = 1502$
 $fork_2 = 1503$

$c_1 = 0$ (A)
 $id = 1502$ $c_1 = 619$
 $fork_2$

$c_1 = 1502$ [Copy from
 $fork_2 = 0$
 $id = 1503$

$c_1 = 0$ (B)
 $id = 1502$ $c_1 = 619$
 $f = 619$
 $fork_2$

Output

Linux

Best of 1

Linux (1)

Best of 1

Windows

Spring - 28 Mid

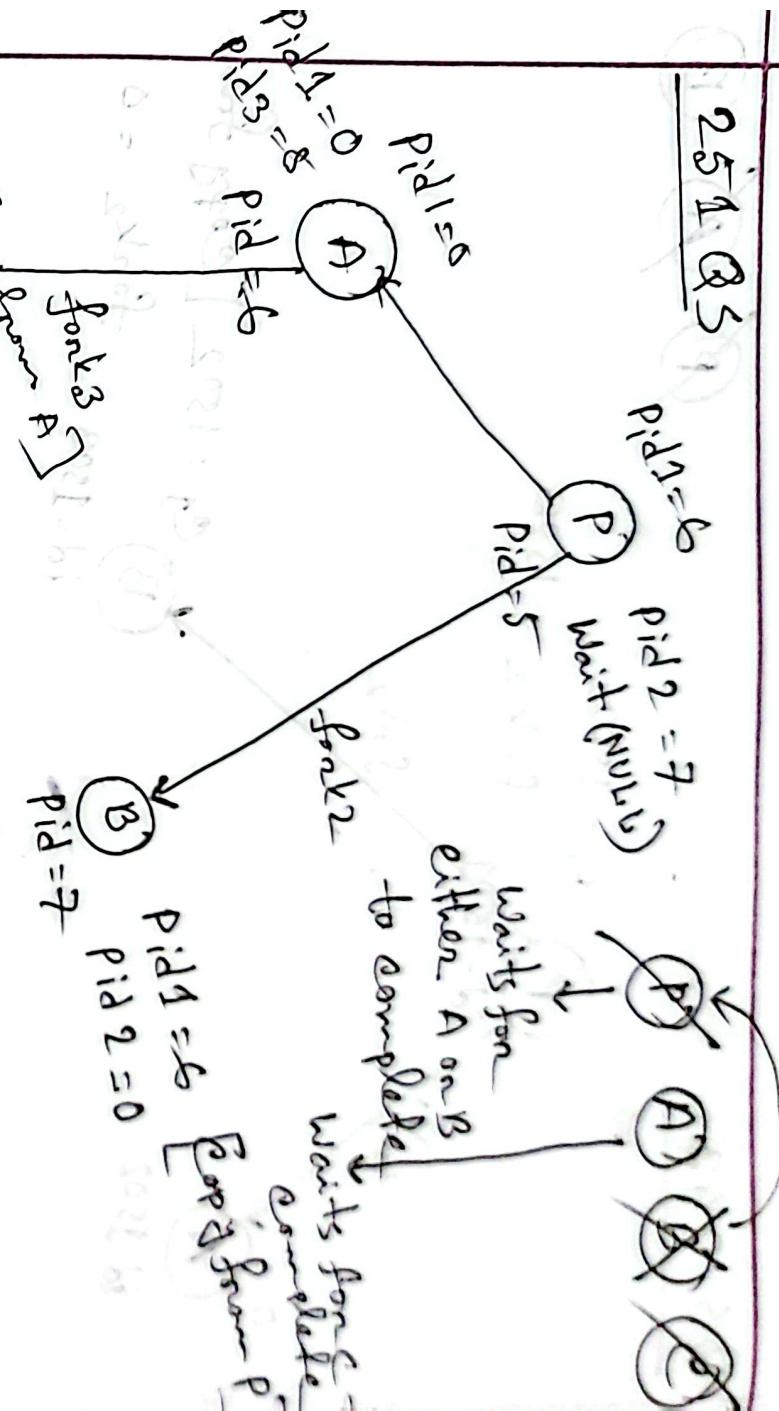
CT - 1 Syllabus:
Lecture - 2 slide full
practical

Fall - 24 CT - 1 QEs

Fall - 24 Mid QEs

Spring - 28 Mid

251 Q5



Output:

(P) Hornerx destroyed

(B) Hornerx destroyed: still

(P) At least one hornerx?

(C) Hornerx destroyed

Scheduling

FCFS

6/05

01

02

Embryo

Resource allocation

Tarrival

Ready

Deschedule

Blocked

Schedule

Running

limit

Tcompletion

Zombie final

03

04 05 06 07

turn around time

08 09 10 11

unit

Response Time

$$= T_{\text{firstrun}} - T_{\text{arrival}}$$

Turn around time = $T_{\text{completion}} - T_{\text{arrival}}$

00

01

02

03

04

05

06

07

08

09

10

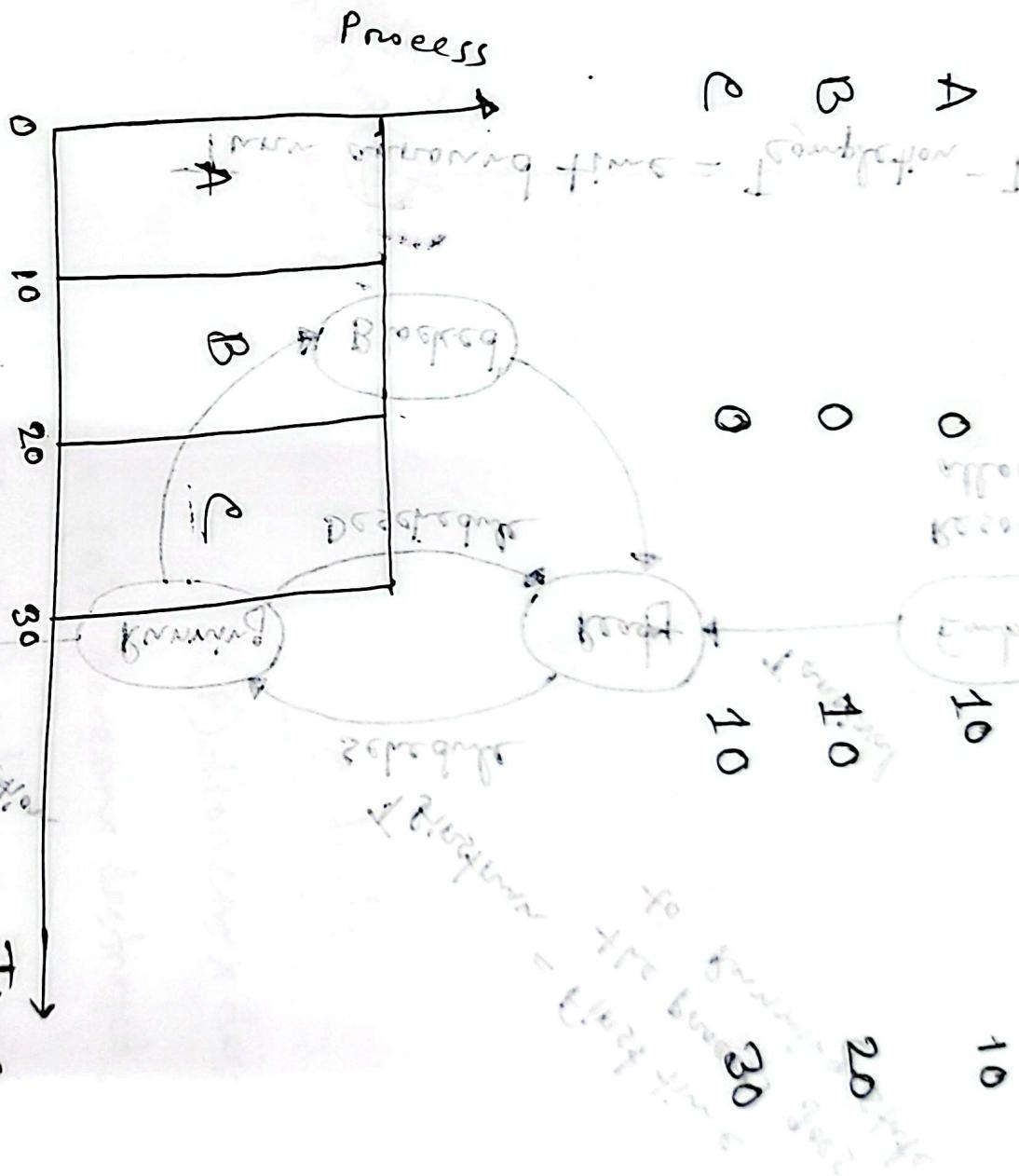
11

12

6/05

1801

<u>Process</u>	<u>Arrival</u>	<u>Duration / Burst Time</u>	<u>Completion</u>
P	0	10	10



一〇

۲۹

2

$$= \downarrow \text{frequency} - \downarrow \text{period}$$

frequency

11

11

12

80

三

6

fine

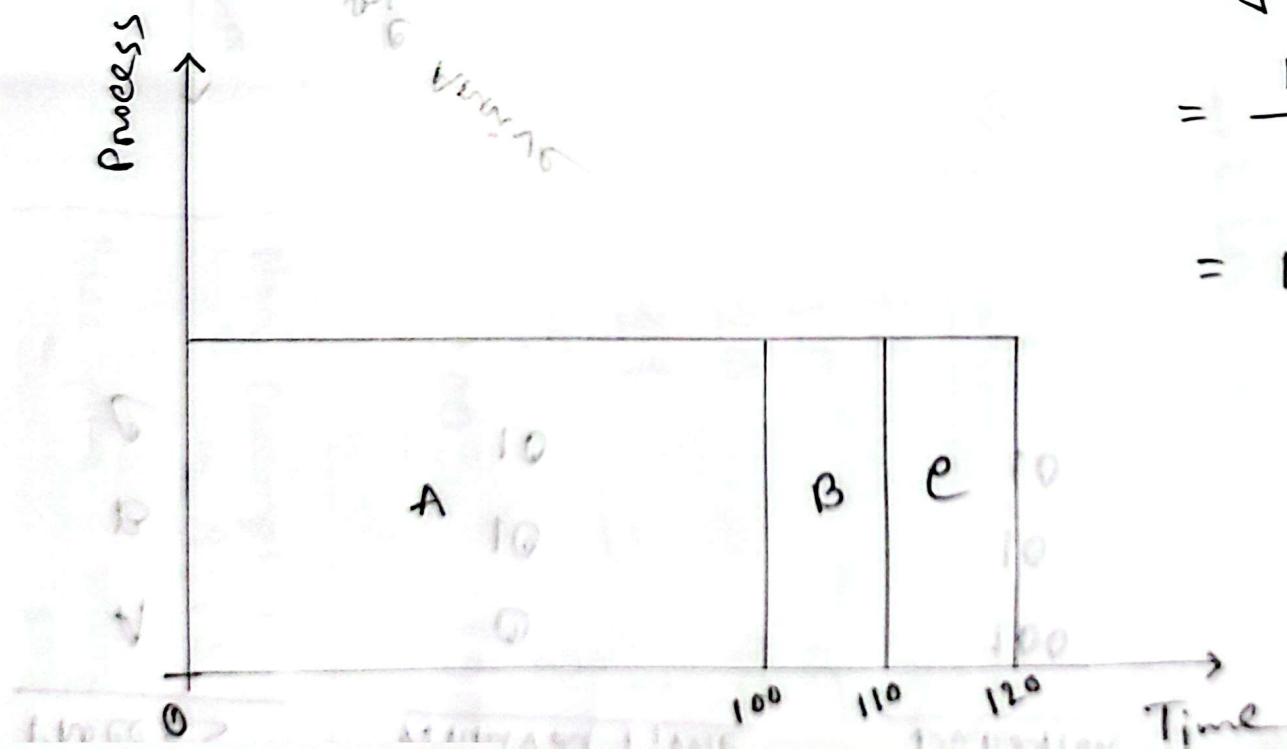
ANd

<u>Process</u>	<u>Arrival</u>	<u>Duration/Burst Time</u>	<u>Tcompletion</u>	<u>Turnaround time</u>
A	0	100	100	100
B	10	10	110	110
C	0	10	120	120

Avg Turnaround Time

$$= \frac{100 + 110 + 120}{3}$$

$$= 110 \text{ unit time}$$



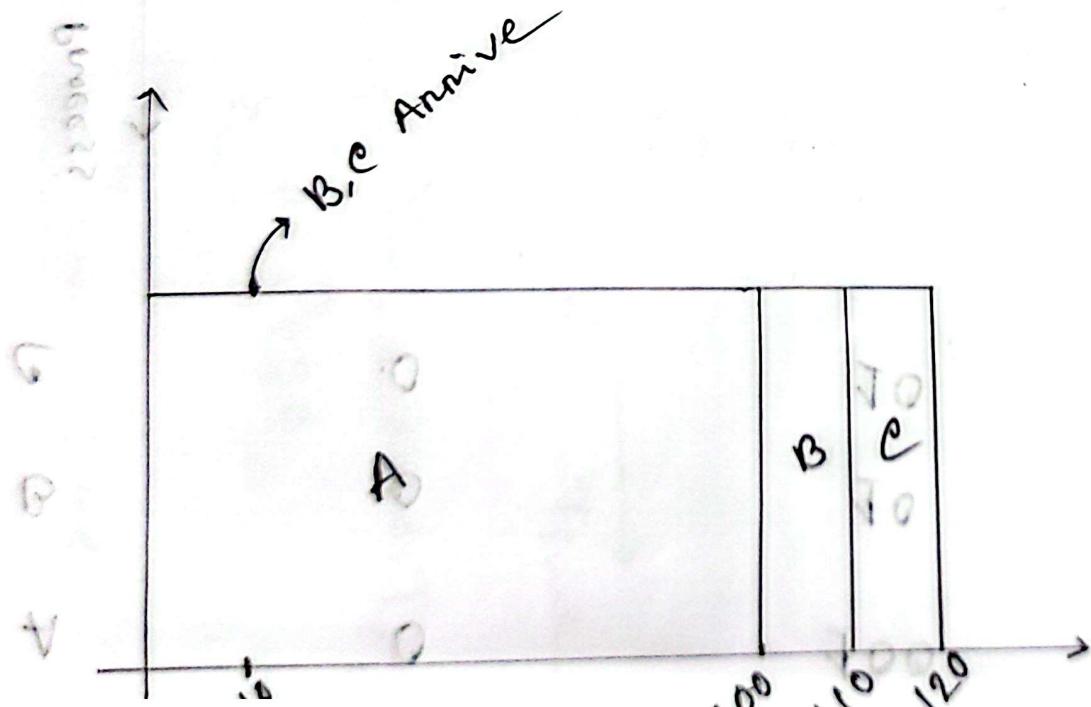
Convoy Effect ***

<u>Process</u>	<u>Arrival Time</u>	<u>Duration</u>	
A	0	100	1100
B	10	10	110
C	10	10	100

= 110 waiting times

= $\frac{1}{3} \times 110$
 $100 + 110 + 150$

avg. processing time



Shortest Time to Completion first (STCF)

- ① At each decision making stage - choose the process with shortest remaining time.

② Decision making stages :-

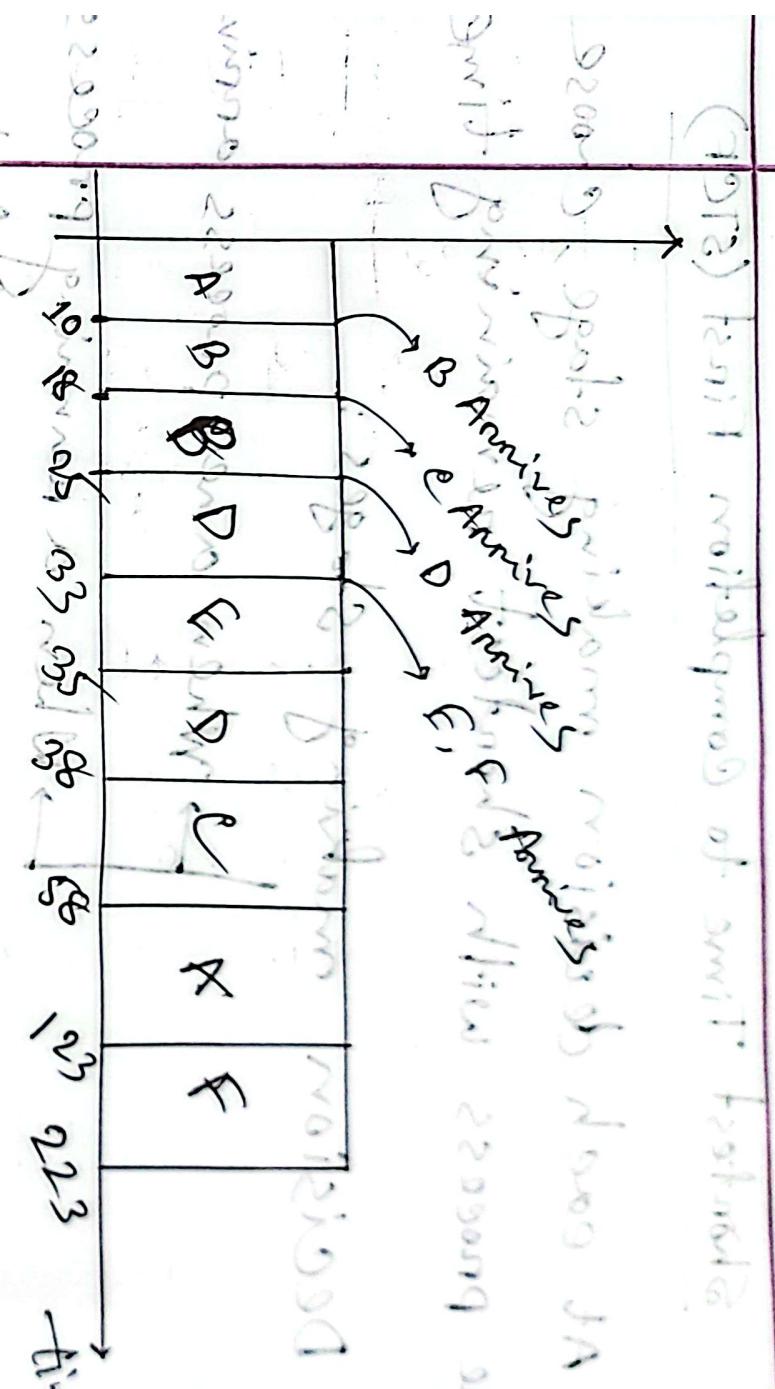
- When a new process arrives.
- When a running process is completed.

<u>Process</u>	<u>Arrival</u>	<u>Duration</u>	<u>Turnaround time</u>
A	0	75	123
B	10	15	15
C	20	40	40
D	25	11	13
E	33	2	2
F	33	100	190
	001		

Any turnaround

Non-preemptive - even
(FIFO, SJF)

Preemptive - shorter turn
time



Process G is completed by time 10.

Process Scheduling - Remaining time

Process	Remaining Time
A	6.5
B	4.5
C	3.5
D	2.5
E	1.5
F	0.5
G	0

Process G is completed by time 10.

Process Scheduling - Remaining time

Process G is completed by time 10.

Process G is completed by time 10.

Round Robin

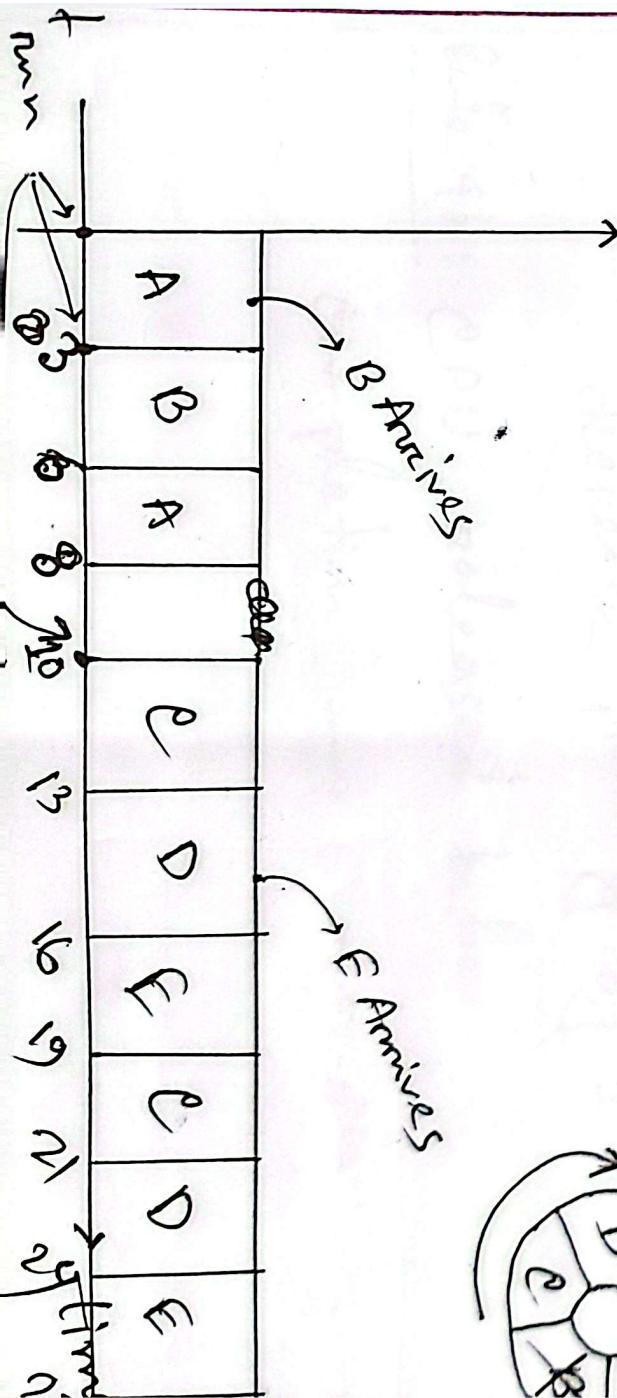
① Improves response time but poor perf in turnaround time.

② Each process gets a fixed time-slice & it is scheduled.

③ After that time slice, RR schedule another process.

Process

Process	A	B	C	D	E	time
Arrival	0	2	10	10	15	
Duration	5	3	5	7	6	



midalt bnnor

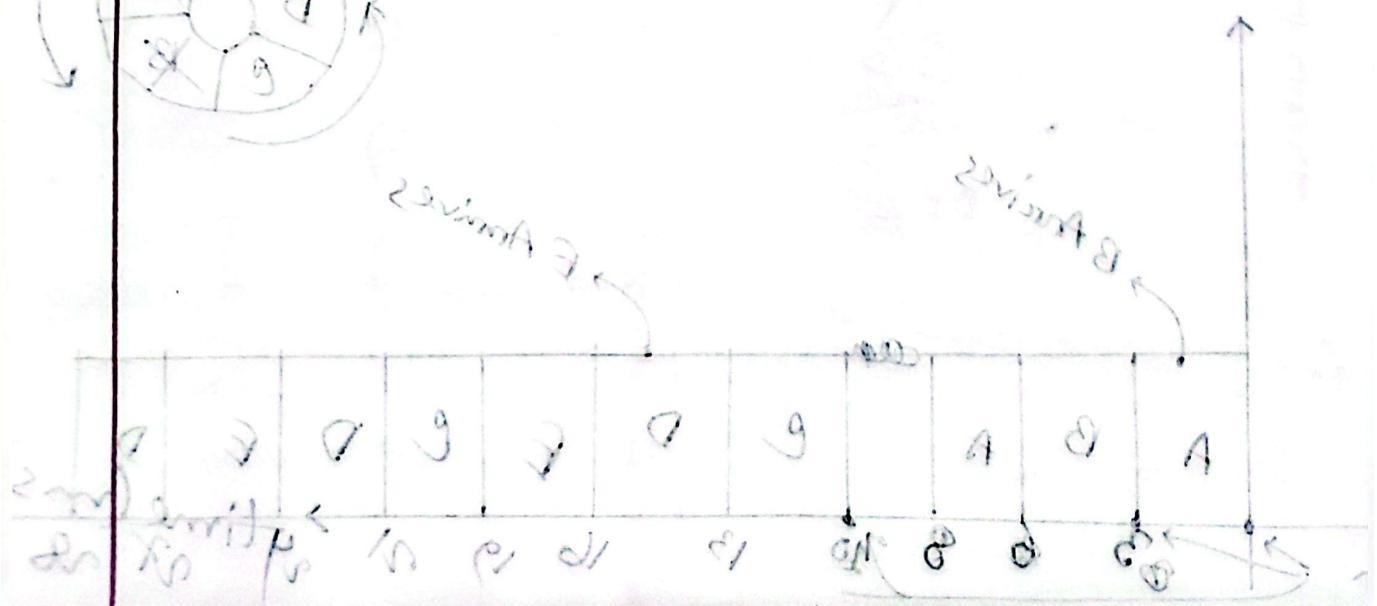
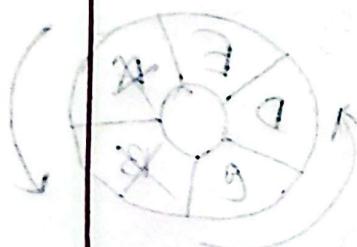
Avg response time

$$= \frac{(0-0) + (3-2) + (10-10) + (13-10) + (16-15)}{5}$$

$$= \frac{1+3+1}{5} = 1 \text{ ms}$$

$$T_{\text{response}} = T_{\text{first run}} - T_{\text{arrival}}$$

$$T_{\text{turnaround}} = T_{\text{completion}} - T_{\text{arrival}}$$

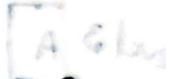


MLFQ

Rule 1: Priority (A) > Priority (B) \rightarrow A runs



changes
Priority



Priority high \rightarrow If CPU is given up before time-slice completion;



Otherwise, A priority lowers down.

Rule 3: Topmost priority at the beginning

Rule 4a: CPU released before time

Rule 4a: CPU used entire time-slice \rightarrow decrease priority by 1.

Rule 4b: CPU released before time-slice completion \rightarrow same priority.

~~problems~~ problems with basic MLFQ

Starvation → (B) Starving & (A)starving → Game theory

Brand → **B**rowsing = **(A)** Browsing
→ **B**rowsing is **3**

rel 3
A

cl2

10

B and C are interactive

process, there are
releasing CPU before

the completion of the model for initiation.

shells - sand. Shells born
egg ("lip").

A Bad Diving Season

old sand and broken up old
soil.

Combination → O_2 + $\text{H}_2 \rightarrow \text{H}_2\text{O}$

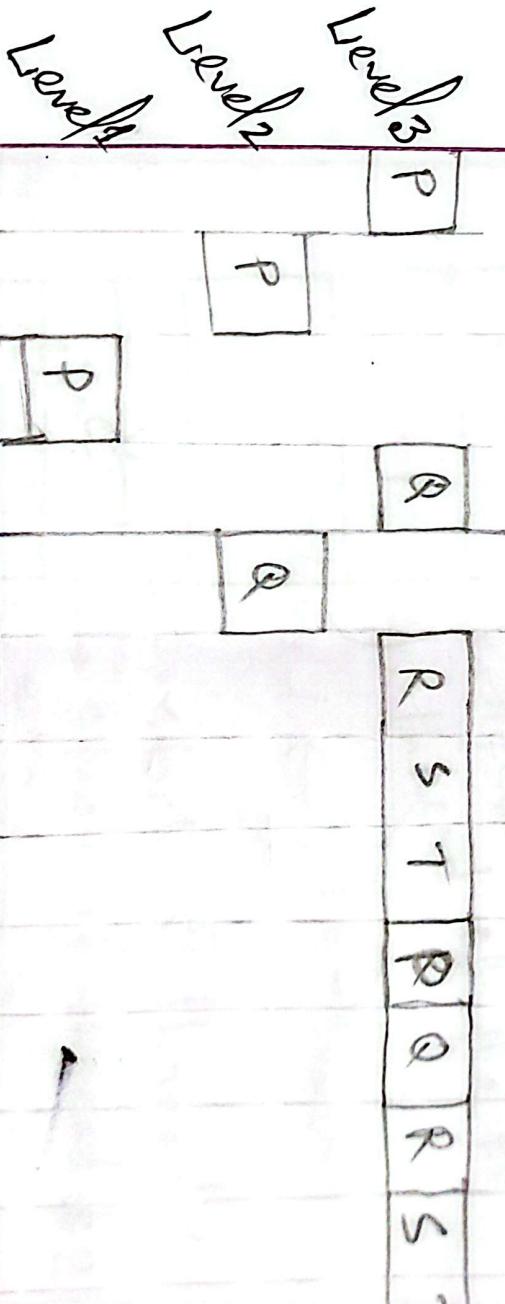
Sol'n: (Starvation)

Rule 5: After some period some jobs go top most priority
(Priority Boosting)

Process	P	Q	R	S	T	U
Arrival Time	0	9	15	15	15	72
Duration(ms)	12	15	6	18	15	3

Priority
Boosting

	R	S	T	R	Q	R	S



0

3 6 9 12 15 18 21 24 27 30 33 36

Q3: P, R, R, S, T, R, R, S, T

Q2: P, R, R, S, T, Q, S, T

Q1: P, Q

17/08/2028

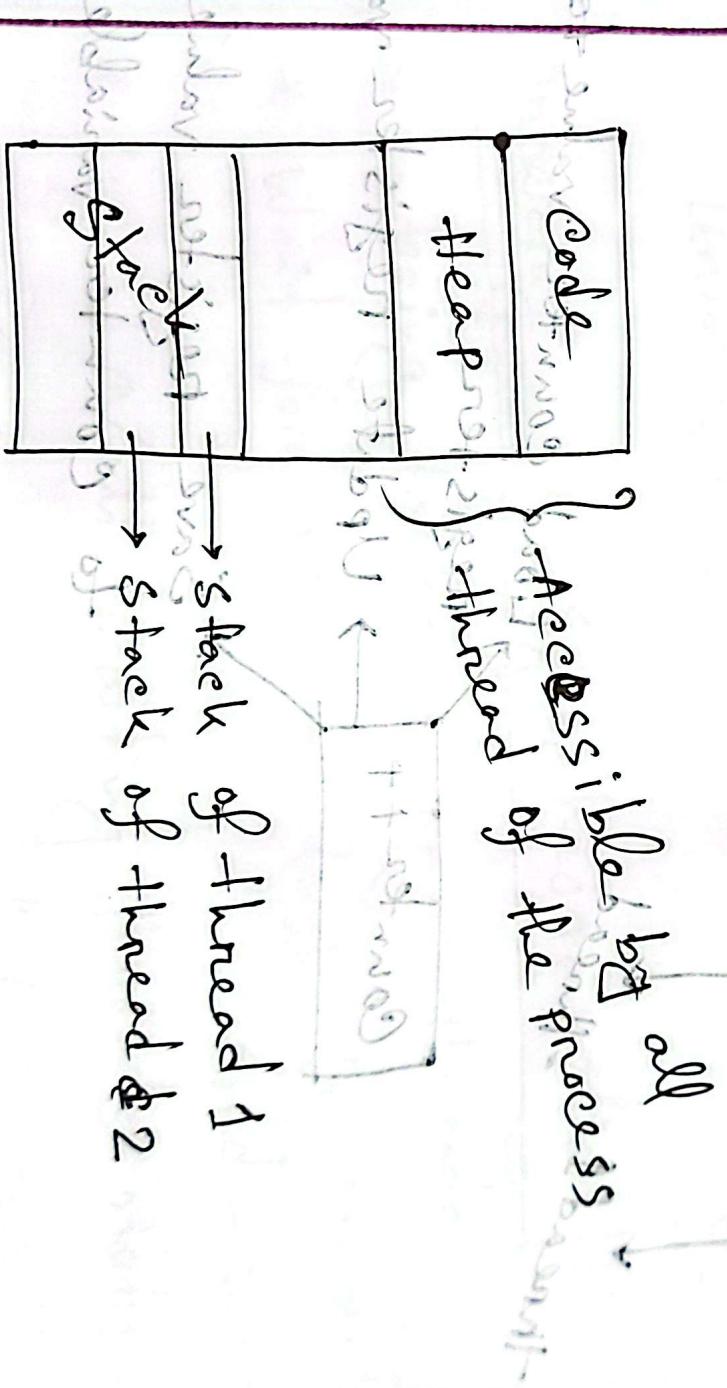
Concerns of

Performing a single task on multiple tasks in parallel and/or maintaining an order.

Thread → basic process

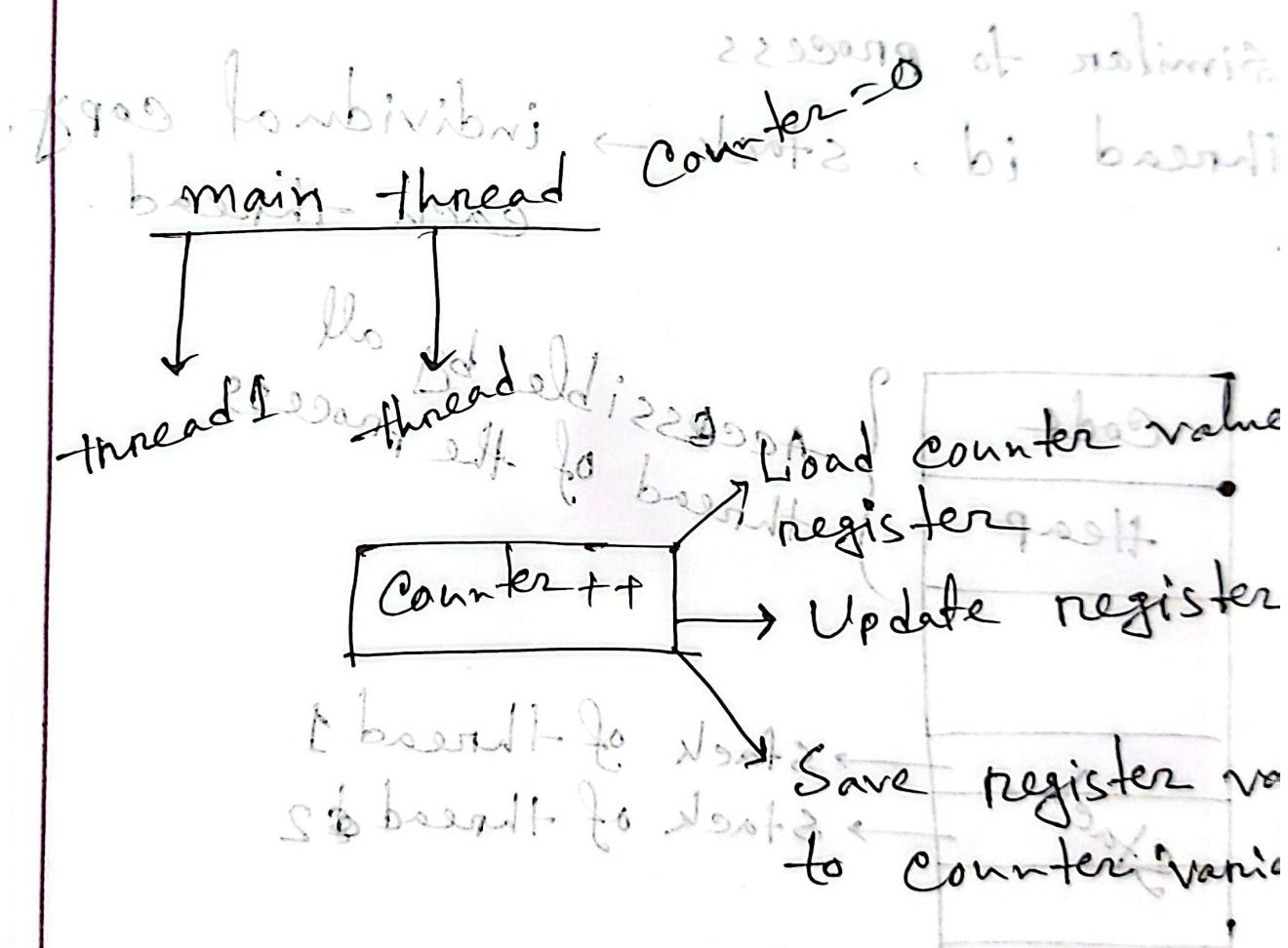
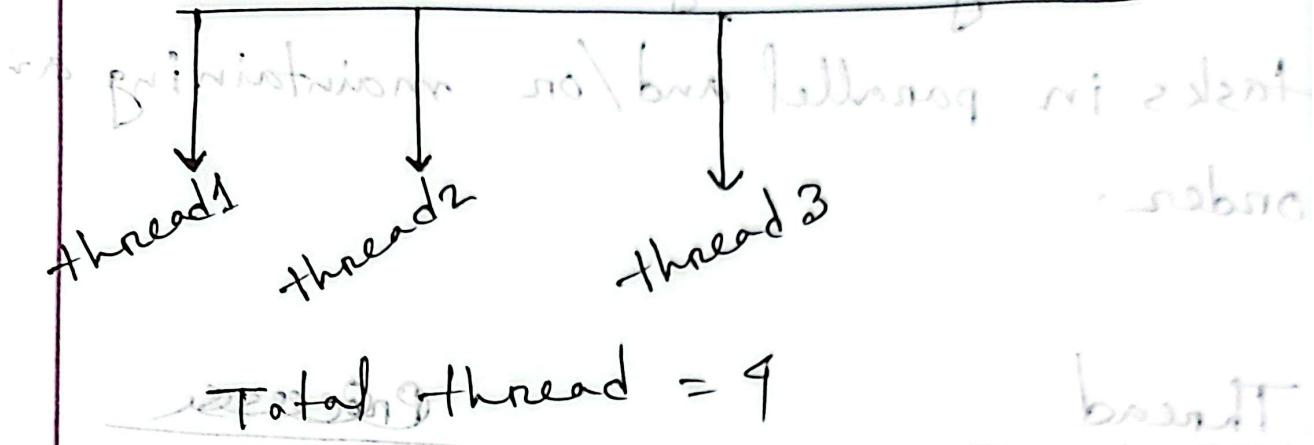
Similar to Process Thread id, stack → individual copy for

Indicates the single back thread.



↳ memory leak

with main thread or process



Mutual exclusion :

If one thread is

Critical section

(using shared data)

Threads must wait for it to complete

→ Achieved with locks / mutex / sem

lock (m); // m is a lock variable

unlock (m);

Ordering on synchronization

* Ordering tasks among threads

* Wait for previous task to complete

cond - n

* Signal all waiting threads not

the task is completed → cond

/ cond-broadcast(),

notifies all waiting threads.

Example of ordering

- ① If produced then can be converted
②

(2) If bus arrives, passengers can get off.
(3) If door is opened, then guests can enter.

③ If door is opened, then guests can enter.

② This room opens to kitchen.

lock (m) ; continue halo fontie
cond - wait (C.m) ;
- . (3). ~~fractoid~~ - bridge

unlock (m) \rightarrow ~~lock~~ \rightarrow ~~lock~~

Producer/Consumer Problem

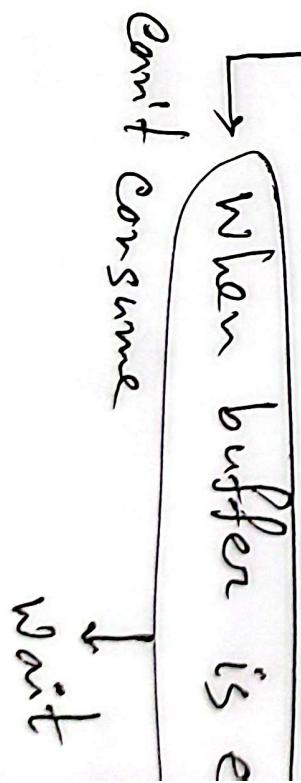
1 producer

↑ consumer
buffer size with size 1.
can't find producer when buffer has
finished to write
producer → empty space

Data item position → when buffer is



consumer →
can consume
→ when buffer has



Implementation: 1 producer, 1 consumer

buffer size 1

so there is 1

```
char mutex; // lock variable  
int max = 1; // buffer size  
int count = 0; // number of elements in buffer
```

Not conditional variable

lock ←
unlock

count ←
count + 1

lock and unlock are done in

sequence

buffer is not used when

count <= 0

first

else
half.
now 2nd

```
void *producer() {
```

```
    void *consumer () {
```

```
        for (int i=1; i<=10)
```

```
            for (int i=1; i<=10)
```

```
                lock(m);
```

```
                lock(m);
```

```
                if (numfull == max)
```

```
                    if (numfull == max)
```

```
                        cond - wait(c, m);
```

```
                        cond - wait(c, m);
```

```
                producer (numfull);
```

```
                consumer (numfull);
```

```
                if (numfull == max)
```

```
                    if (numfull == max)
```

```
                        cond - signal(c);
```

```
                        cond - signal(c);
```

```
}
```

```
}
```

```
}
```

20/08/20

producer 1, consumer 2.

Ready

Running

Blocked

con1, con2, p1, p2

con1

con1 → co

con2, p1

con2

con2 → co

pro1 → pro2
signal (cond)

con2 : without signal
con2 : break block

con2

in use

pro1 → con

* Use different conditional variables for

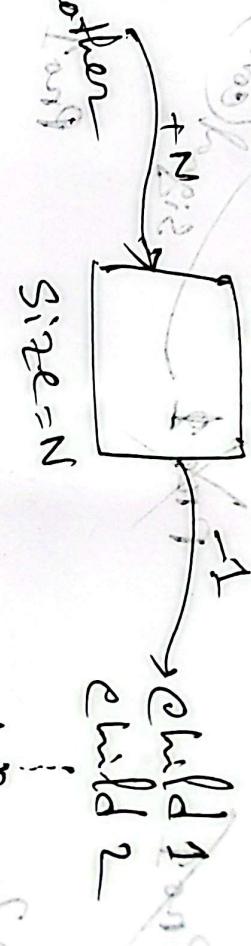
types of threads

producer threads: 3 { Total threads = 4
consumer threads: 4 { Different type

i () not most efficient for
i - frames

* Use while loop when some other thread take resources before you can run after returning from waiting.

problems



child thread Implementation:

Whid (true)

not 2ndarily favoring bonds over stocks

hook m
~~white~~ (can't = 0) } }

} getChocolateFromBox();
count--;

5051-
5051-
5051-
5051-

`eat()` : { () branch - 2nd additional
`unlock(m)` : (optional) lock unregistered

Business - personal - social
Business - personal - social

٢٦٧

void longest(void *args) {

(L.) *bovinum* *lock* (*m.*) *borealis*

Request Court to issue a writ of mandamus.

```
signal (cv->host); // if (quest  
= =  
signal (
```

```
signal (ev_guest) {  
    broadcast  
    enterHouse();  
    unlock(m);  
}
```

م

```

function bus-thread() {
    /* for
       lock(mutex) (or) join */
    bus_arrived = true;
    while (bus_arrived) {
        waiting_count = boarded;
        for (int i = 1; i <= boarded; i++) {
            signal (bus_arrived_cond);
            wait (&passenger_boarded_cond, &
                  bus_arrived); // wait for
            if (bus_arrived == false) { // if bus
                depart(); // has
            }
            unlock (mutex); // - no) leave
        }
        /* do other stuff
           i (no) leave
    }
}

```

Mid 251 - 4(a) : Determining output from concurrent code .

Thread 0 → $K=0$ $i = 2$, $K = 2 \cdot 3$

Thread 1 → signal (P_2)
~~Thread 1 → i=2, k=2, done[2]=0; i=~~

~~Thread 2~~ → signal (e[1])

LOGBOOK

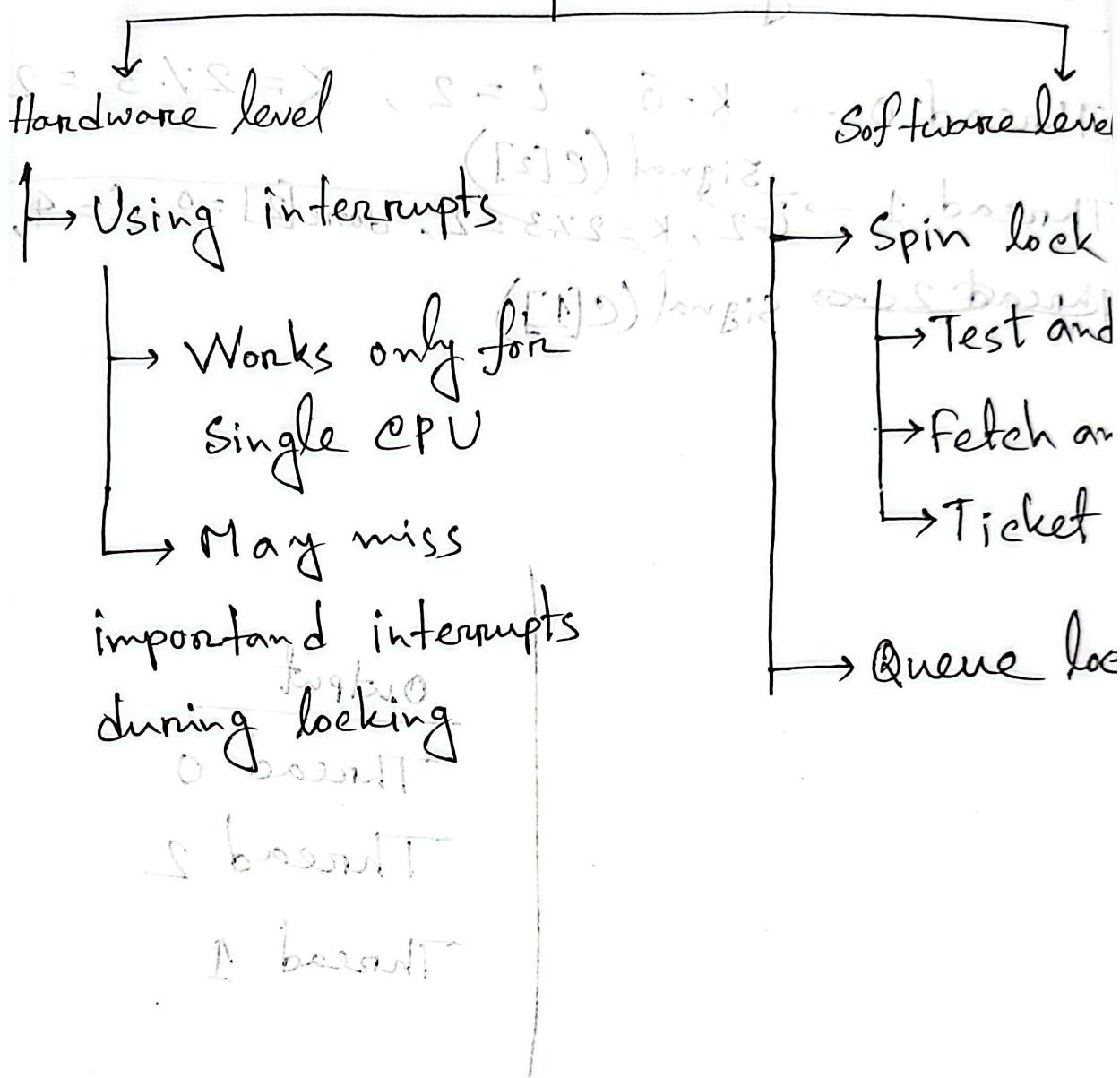
— 1 —

had enough

Output

Thread 1
Thread 2

most two Lock Implementations



Bug in spin lock

spin lock

: (1. - (1. & 0)) & 2. b & !s == 1) while

Thread 1

Thread 2

while (*lock);

(lock * bin) spins here

Desecheduled

o = lock *

- - - - - Desecheduled

*lock = true



Bug: Both threads grabbed the lock

Solⁿ: Test And Set: checks the availability of the lock (if available) atomically with the help of hardware support.

while (testAndSet(&lock, 1) == 1);

 |
 | int old = *lock
 | *lock = 1
 | return old;

| lock available
| unavailable

void acquire(int *lock){

 while(testAndSet(lock, 1) == 1)

 spinWait();

void release(int *lock){

 if(*lock == 1)

 testAndSet(lock, 0);

* lock = 0

Thread 1

lock acquire(&lock);
lock : bus

lock Counter++ : lock test
lock release(&lock);
lock swap(&lock); lock test

lock swap

Compare and swap Spin lock

lock x = lock + 1
if x == lock x
lock swap

2nd problem with spin lock: No quantum
every thread will get lock eventually

```
Thread 1
for (int i=0; i<n; i++) {
    acquire(&
    counter +
    release(&
    lock);
}
```

If Thread 1 is descheduled each time holding the lock, then Thread 2 will hold the lock and will be in starvation

Solution: Ticket lock \rightarrow Guarantees that thread will get the lock eventually.

turn ~~locked~~ \rightarrow The running serial number (

~~turn~~ \rightarrow What's my serial number ticket

i + something

(bad S) reader

i (bad S) writer

i (bad S) reader

new Smith does not have a lock so it can't be read by anyone else. It can be written to by anyone else. So it is not fair.

Process Creation
(fork, wait, exec)

Scheduling

Simulation

Focus on questions

similar to [243 mid]

Conceptual question

similar to 252 CT2

[251 mid]

251 CT1

243 CT-2

- c) Drawback of DAPQ: The OS
change the aging score contin
the background.

writer threads