23/07/2025

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

Operating System → A trusted software that wor as bridge between hardwa and user applications.

- → Resource Managem
- → Security of Data
- → Increasing Perform

Hardware (CPU, HDD, etc)

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, MLF
5. Concurrency (Basic idea on threads, locks, syncronization, producer ~~procedures~~ : consumer prob

27/07/20

Code $\xrightarrow{\text{compile}}$ Assembly $\xrightarrow{\text{Assembler}}$ Machine code

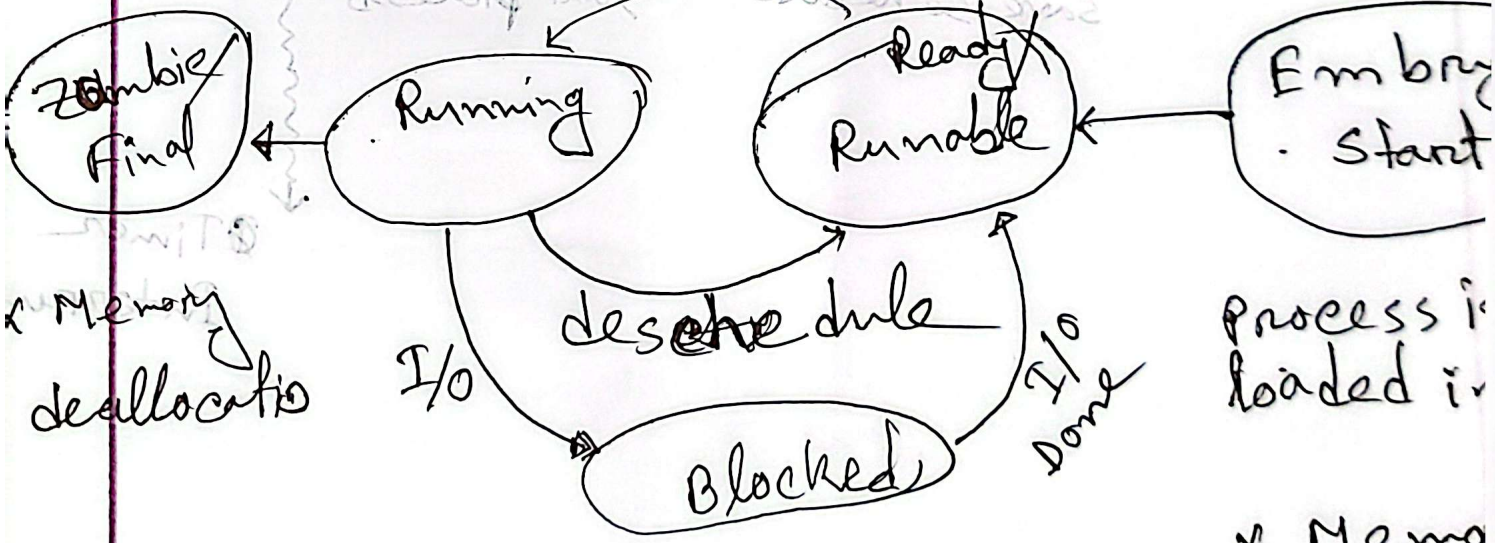
Instruction

a = 10. $ t0 = 10

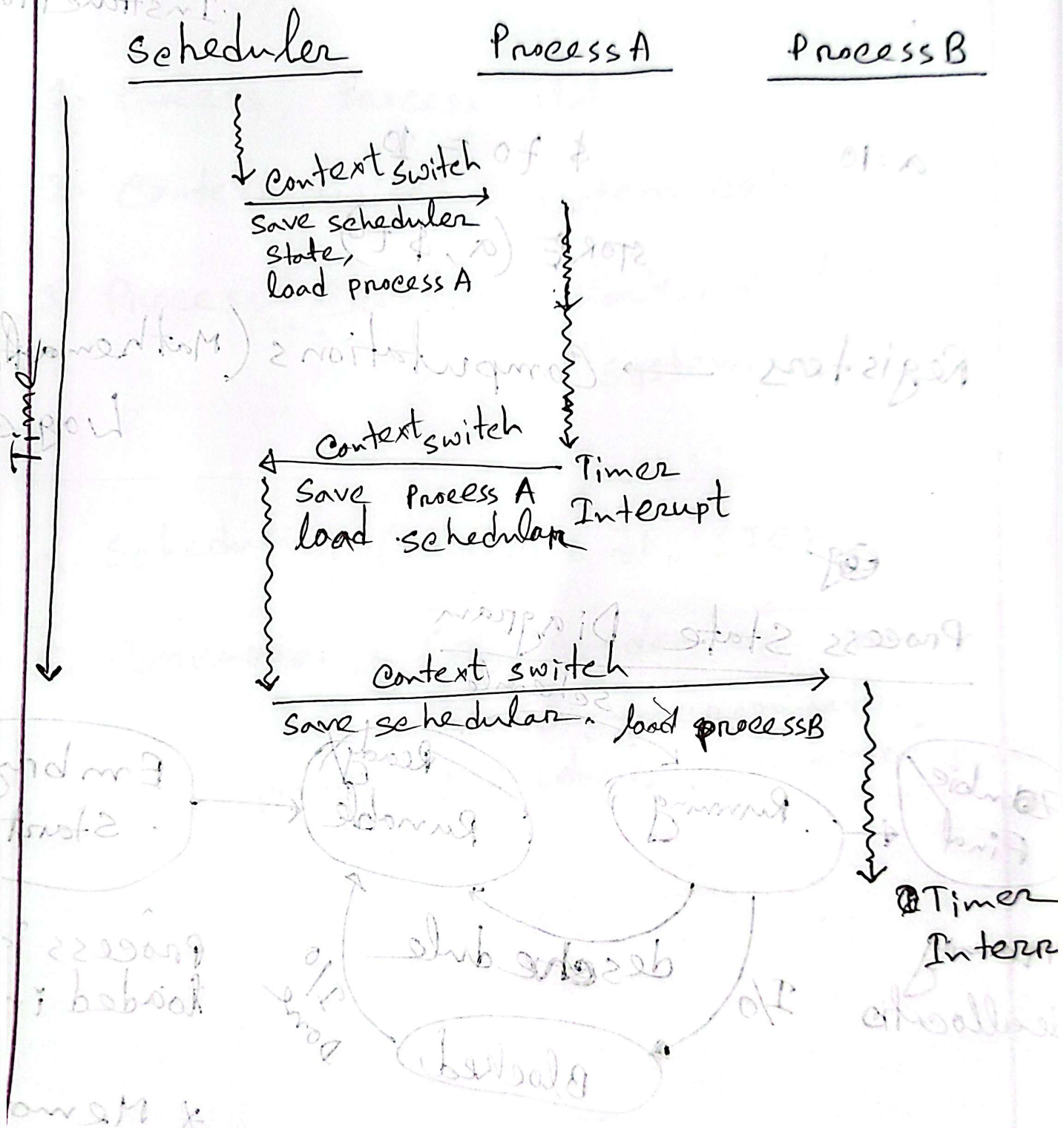
STORE (a, $t0)

Registers ⟶ Computations (Mathematic
Logica

Process State Diagram



Context switch



Chapter-5

rocess Creation

_k → Creates an exact copy of the calling process, process id, memory & file descriptors are different.

it → can wait for a process or child process.

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

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

```
int a = fork(); //a=0
if (a==0)
    printf("child
else
    printf("Parent
```

child process Pid = 6

3. return value of fork is set

3.1 0 for the child process

3.2 pid of child process for the calling process

4. Return from fork [Both for (A) & (B)]

[Process B run হওয়ার আগে কোন
Both process কে কখন run করাবে]

30/07/20

Overview of Process Creation

```
fork (void) {
        Copy memory of parent to child
        initialize pid for child
        set return value of child, a0=0;
                                // a0 → register of child proc
        return pid ; // pid → pid of child
}
```

`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

True && fun()

↳ will be executed

False && fun()

↳ will not be executed

```
if (fork() || fork()) {
```

↓ compilation

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

   $t1 = fork()

   $t2 = $t0 || $t1

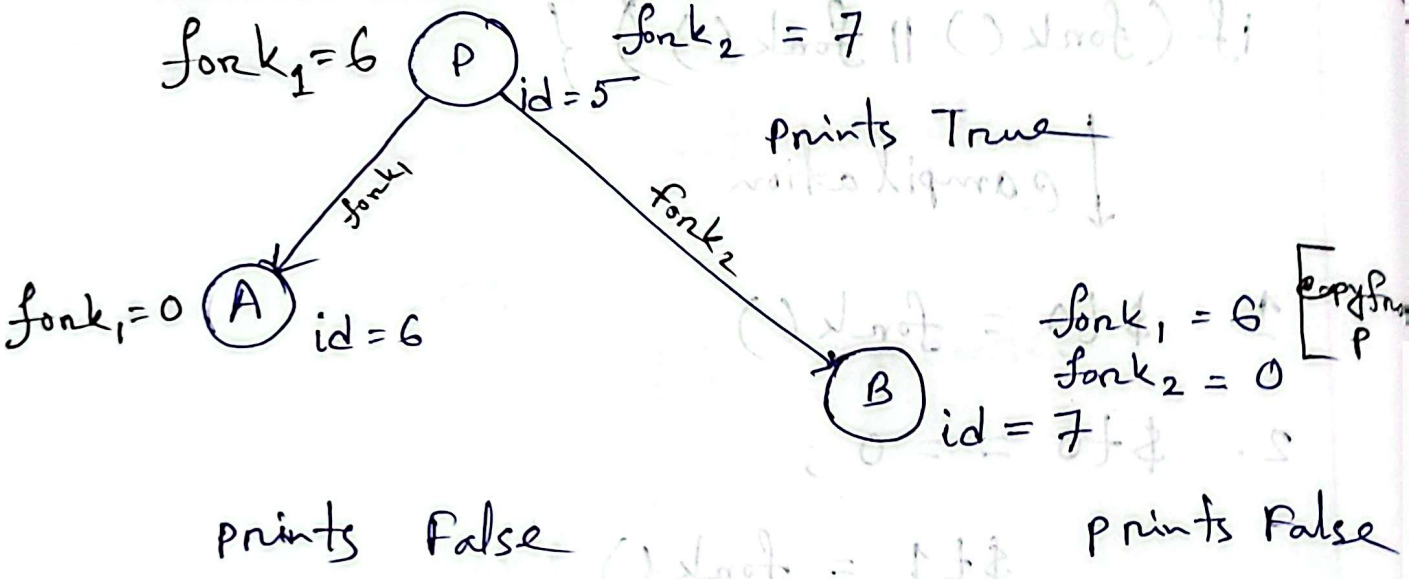
   if $t2 == 0;

   . jump to label B

   jump to label A

```
If (fork() && fork()) {
    printf("True\n");
} else {
    printf("False\n");
}
```

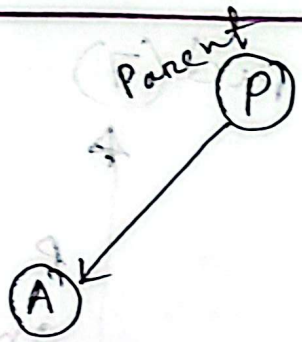
Draw the process tree and determine the output.



output:

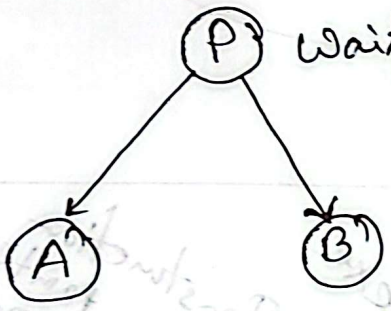
True
False
False

03/08/2025



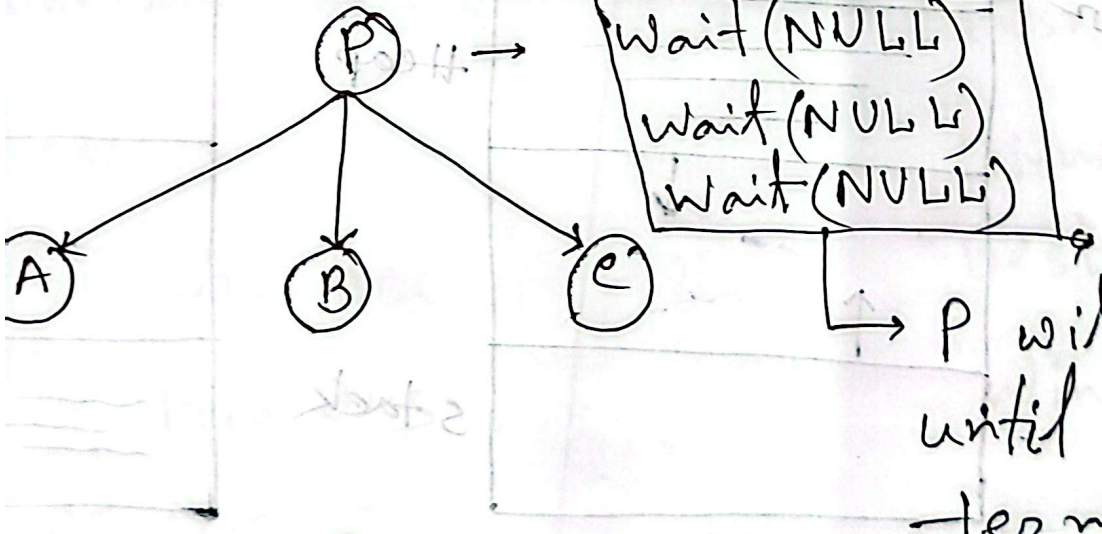
wait(NULL)

↳ p will wait until child A terminates



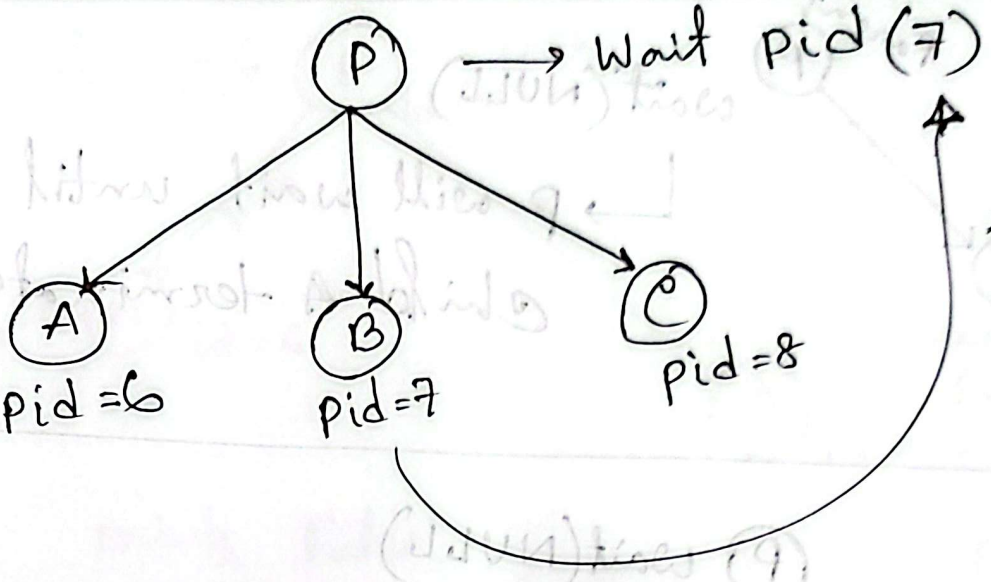
wait(NULL)

↳ p will wait until either A or B terminates.

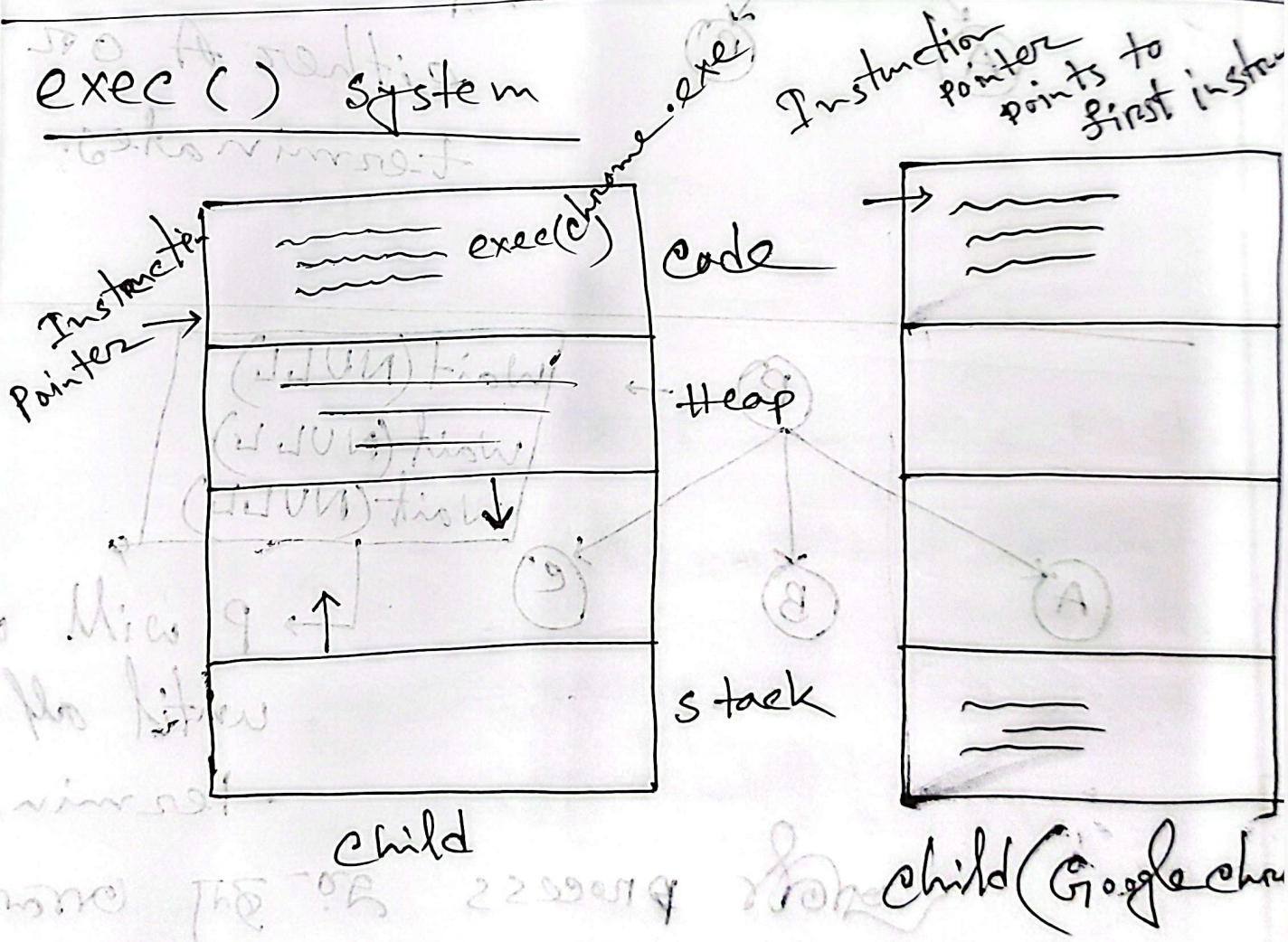


↳ p will wait until all children terminate.

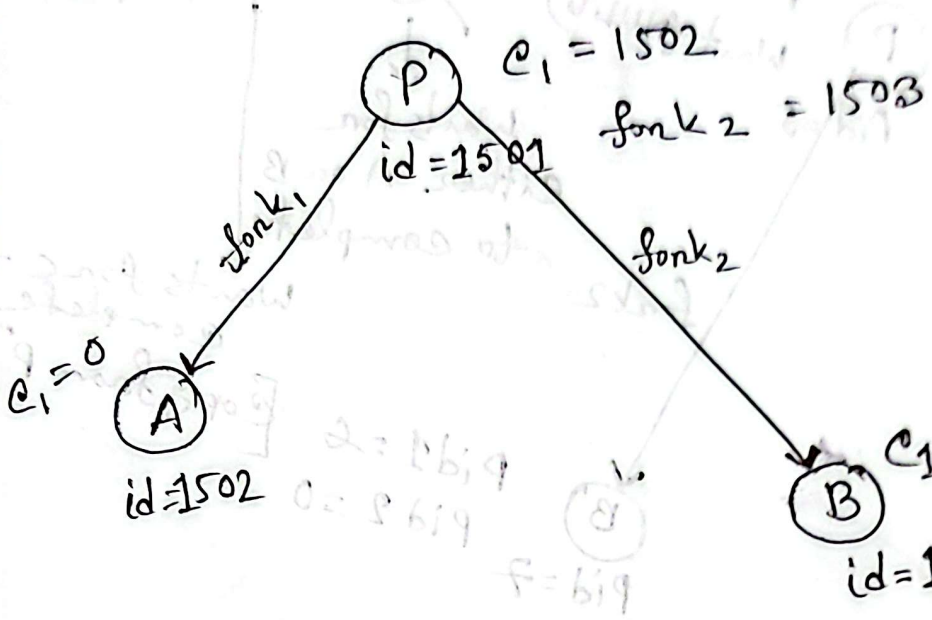
আগের process এর জন্য অবশ্যই wait call.


P
Wait pid (7)
A
B
C
pid =6
pid=7
pid =8
P will wait only for B

exec() system


Instruction pointer points to first instr
exec(chrome.exe)
Instruction Pointer
code
Heap
stack
child
child (Google chr

Qs 243



CT-1 Syllabus:

Leecture-2 slide full

Practice

Fall-24 CT-1 Qes

Fall-24 Mid Qus

Spring-25 Mid "

Output

Linux

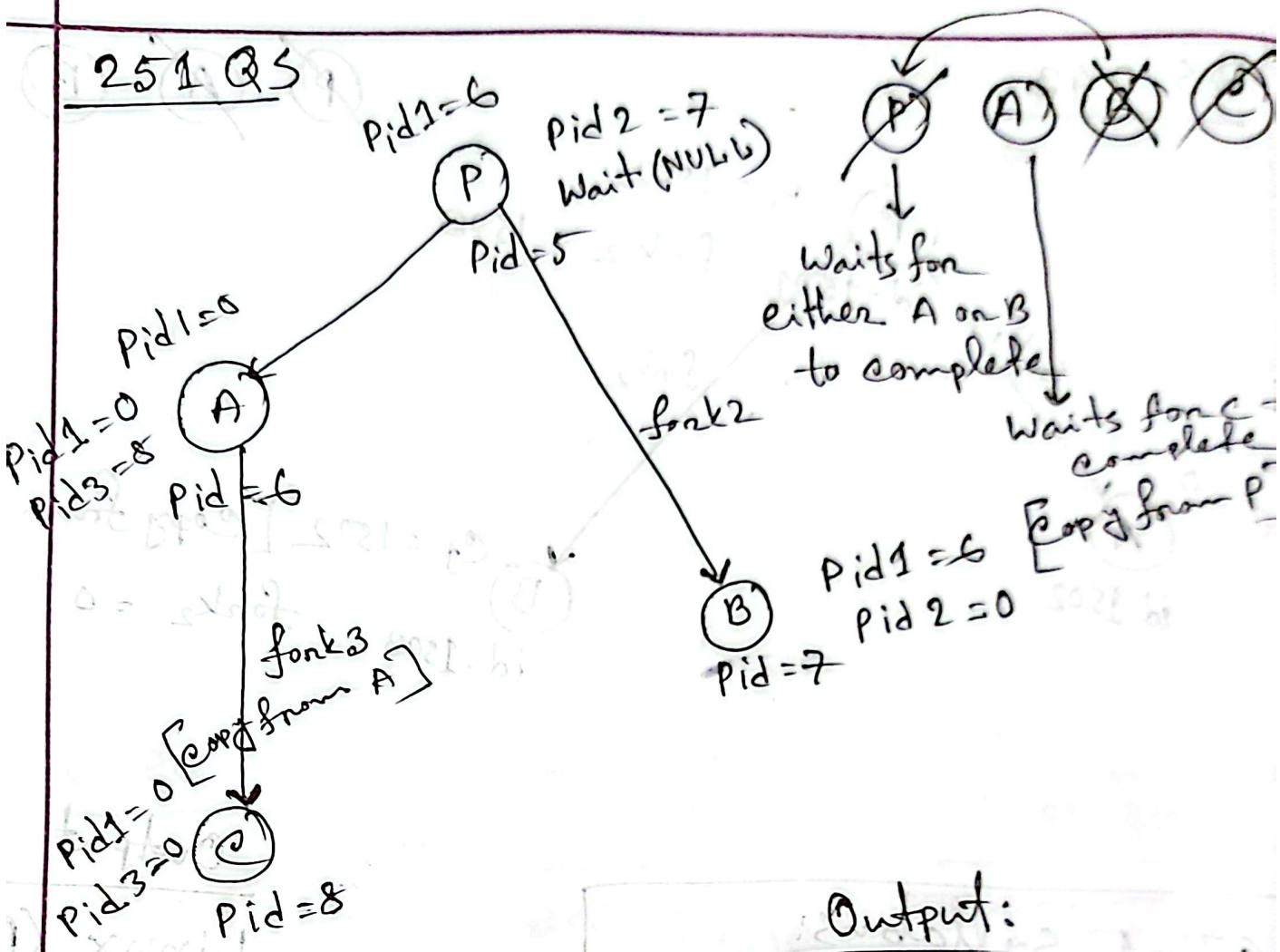
Best of

Linux (

Best of

Windo

251 Q5



Output:

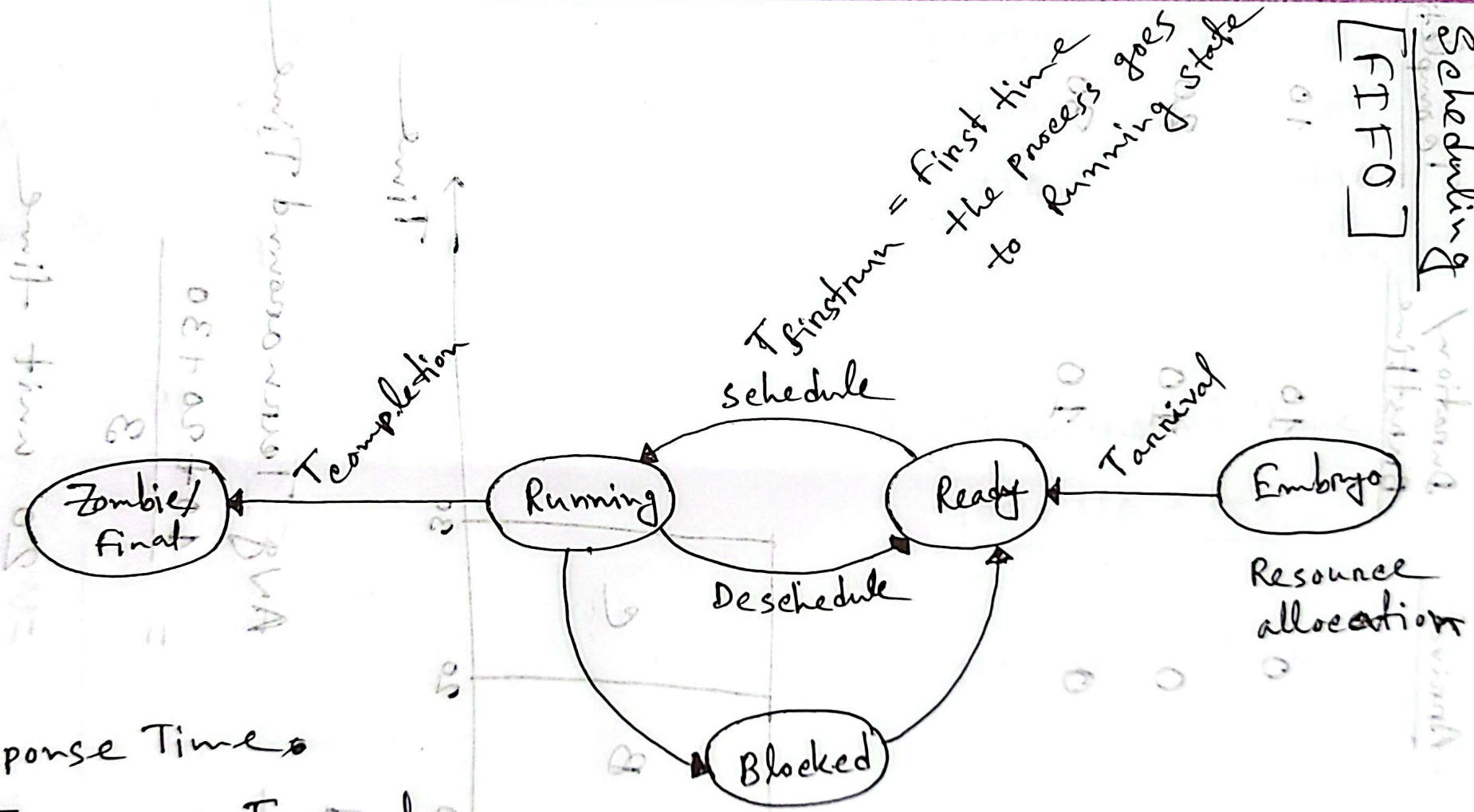
(P) Horcrux destroyed

(B) Horcrux destroyed. Shyt!

(P) At least one Horcrux d

(C) Horcrux destroyed

Scheduling [FIFO]

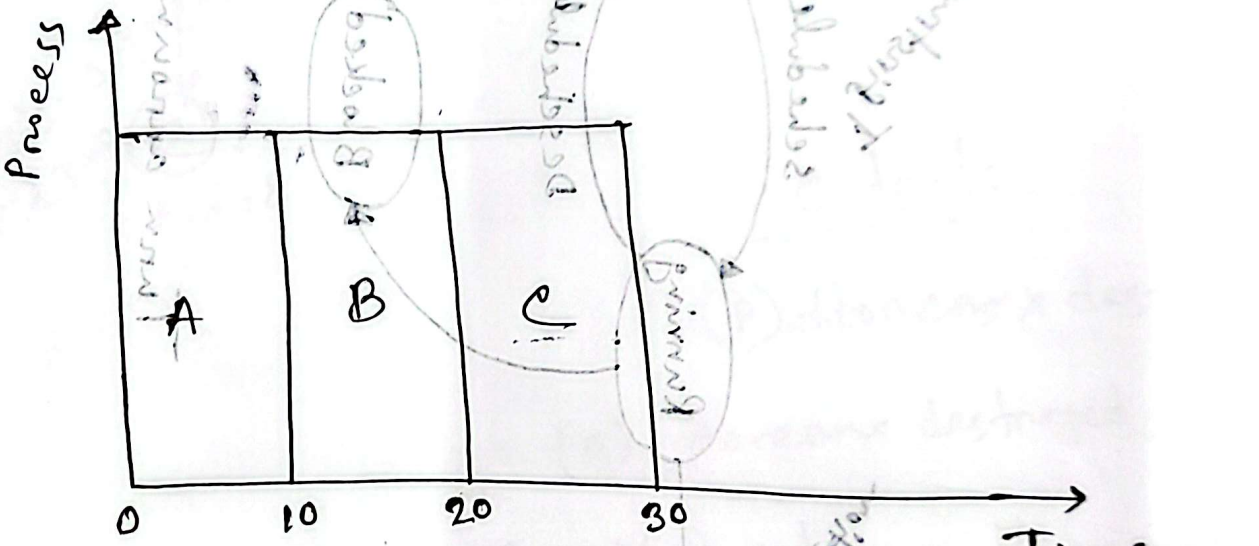


Response Time

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

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

Process	Arrival	Duration/ Bursttime	Tcompletion
A	0	10	10
B	0	10	20
C	0	10	30



Turn around Time
10
20
30

Avg Turnaround Time

$$= \frac{10+20+30}{3}$$

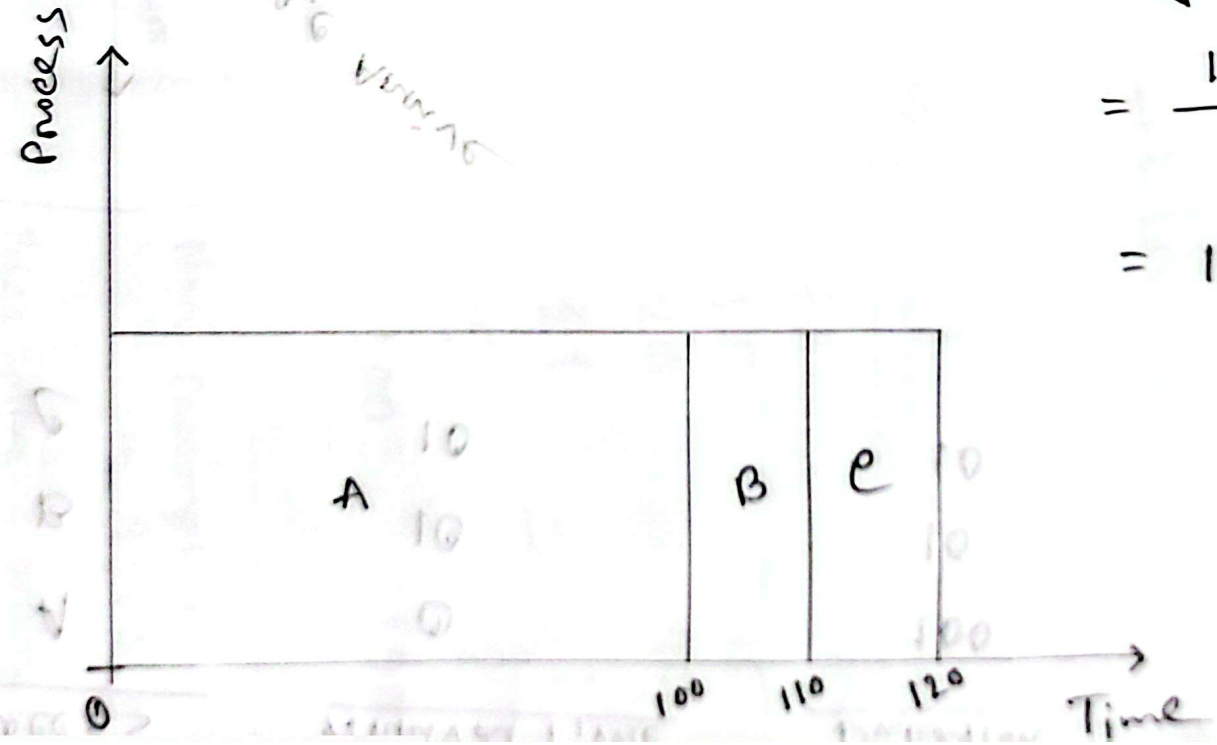
$$= 20 \text{ unit time}$$

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

Avg Turnaround Time

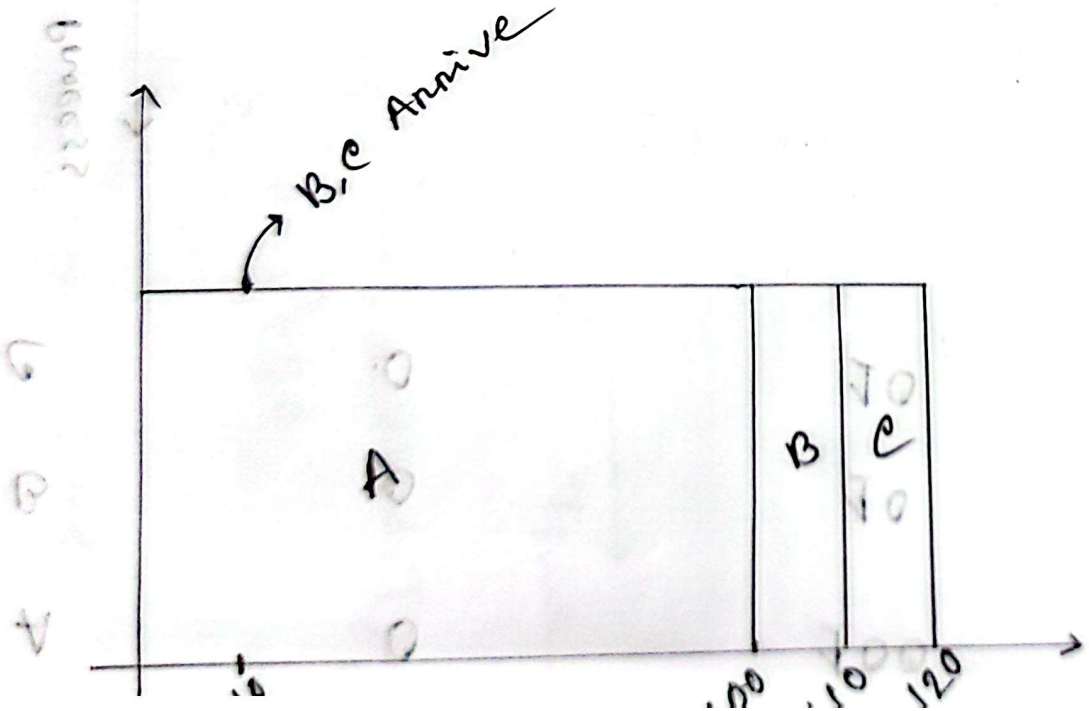
$$= \frac{100 + 110 + 120}{3}$$

= 110 unit time



Convoy Effect ***

Process	Arrival Time	Duration
A	0	100
B	10	10
C	10	10



Shortest Time to Completion First (STCF)

① At each decision making stage, choos
the process with shortest remaining ti

② Decision making stages :

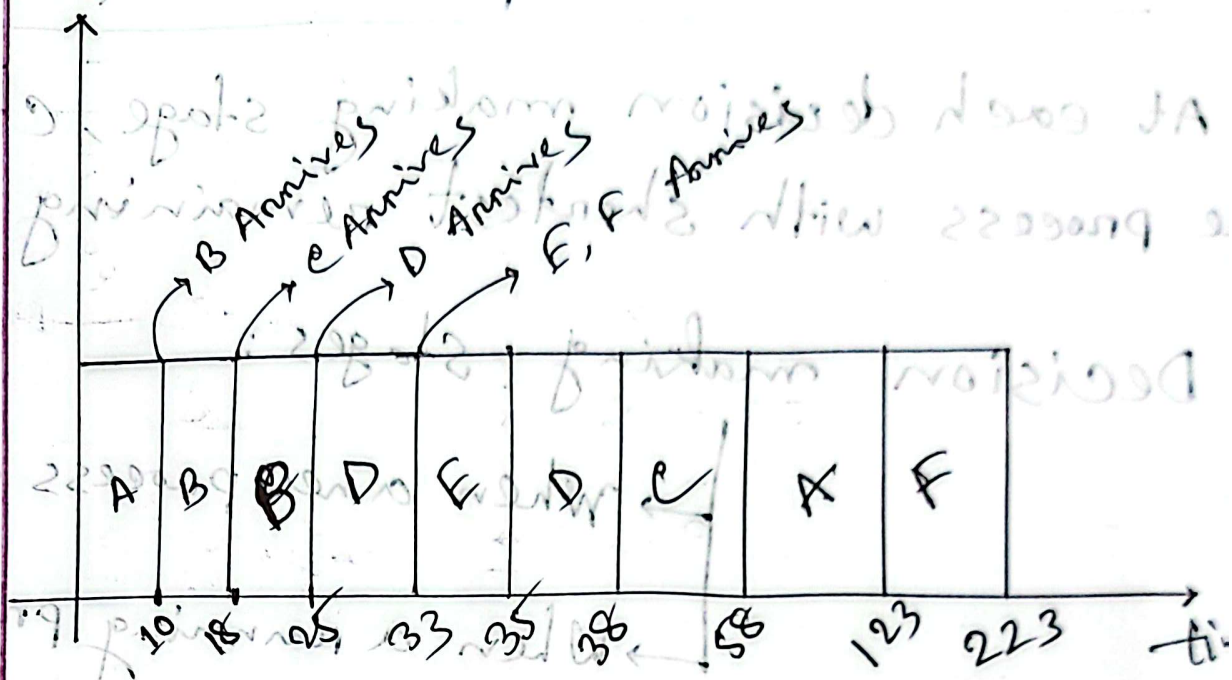
- when a new process arr
- When a running proce
  is completed.

Process	Arrival	Duration	Turnaround time
A	0	75	123
B	100	15	15
C	18	20	40
D	25	11	13
E	33	2	2
F	33	100	190

Avg turnaround

Non-Preemptive - sram
(FIFO, SJF)

Preemptive - sramar ₹r
STCF



Process	Remaining time
A	65
B	8
C	20
D	13
E	2
F	100

Round Robin

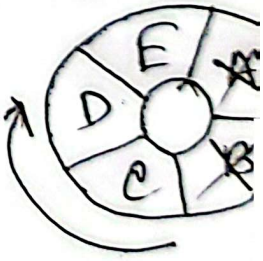
① Improves response time but poor perf in turnaround time.

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

③ After that time slice, RR schedule another process.

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

time-



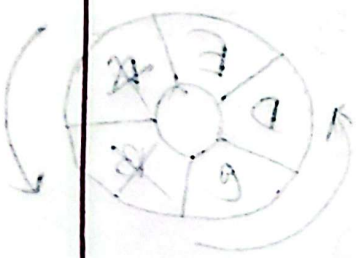
Avg response time

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

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

$$T_{response} = T_{firstrun} - T_{arrival}$$

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



13/08/2025

MLFQ

Rule 1: Priority (A) > Priority (B) → A runs

Rule 2: Priority (A) = Priority B → A and B r
in Round

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

Otherwise, priority lowers down.

Rule 3: Topmost priority at the beginig

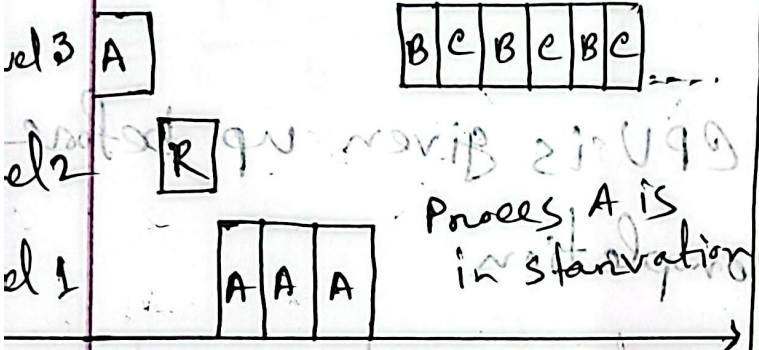
~~**Rule 4a:** CPU released before time~~

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

Rule 4b: CPU released before time-slice completion → Same priority.

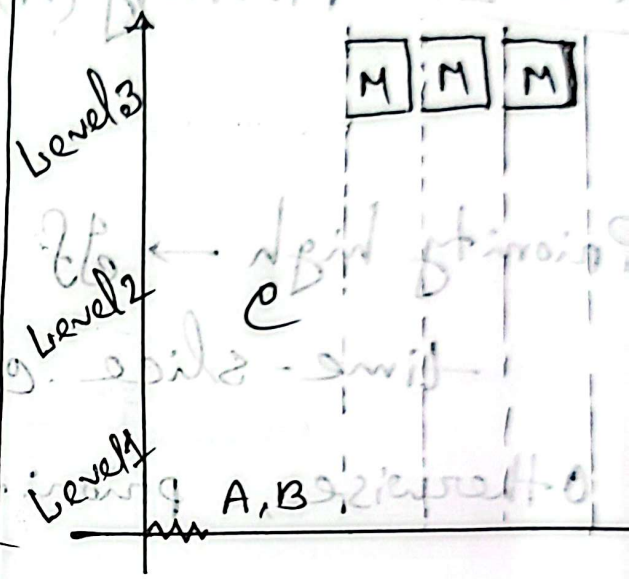
Problems with basic MLFQ

Starvation



B and C are interactive process, they are releasing CPU before the completion of time slice

Game the Sched



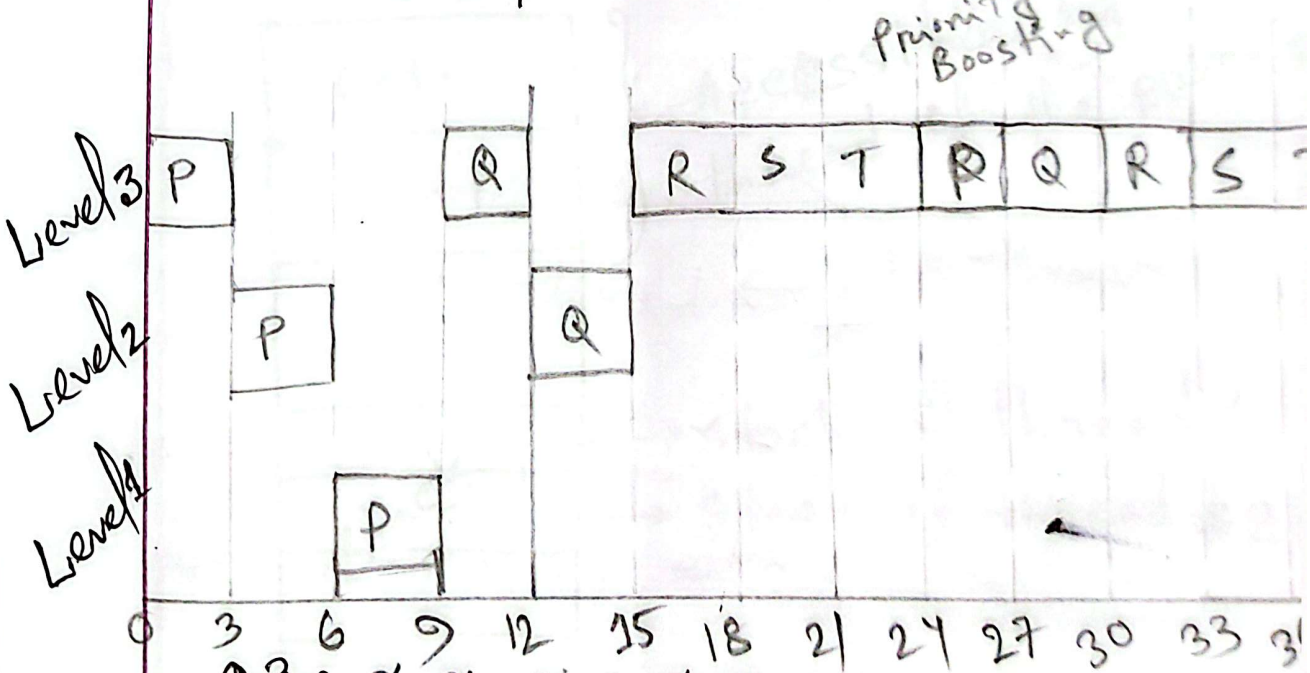
a.c

```
int main() {
    for (int i=0; i<10
        exec("%a");
    }
```

Soln: (Starvation)

Rule 5: After some period S m
jobs to 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



Q3: ~~P~~, ~~Q~~, ~~R~~, ~~S~~, ~~T~~, ~~R~~, ~~Q~~, ~~R~~, ~~S~~, ~~T~~

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

Q1: ~~P~~ ~~Q~~

17/08/2025

Concurrency

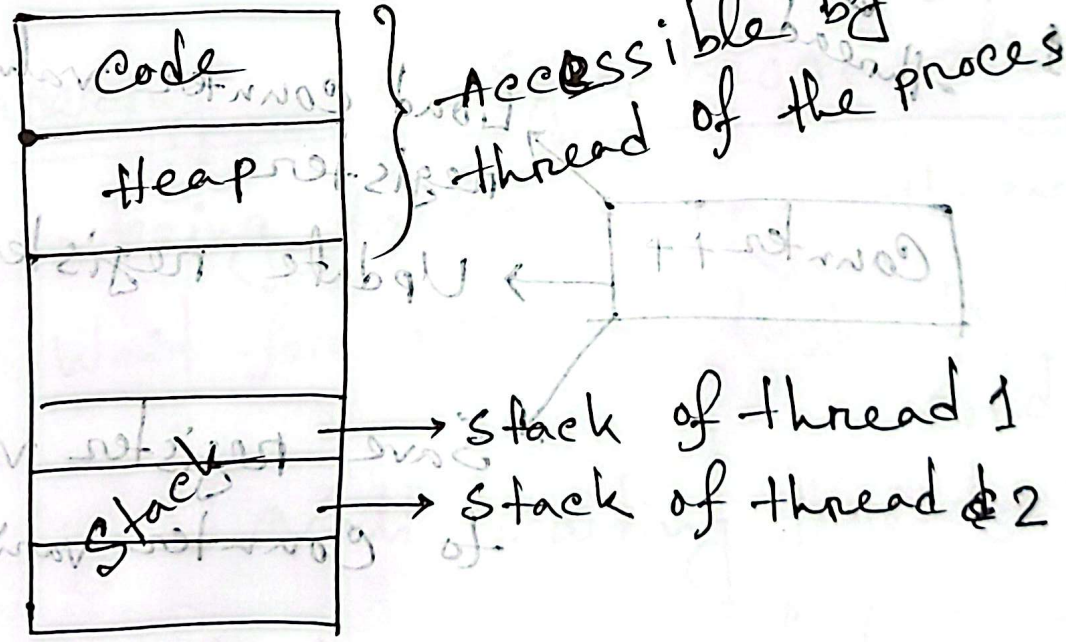
Performing a single task on multiple tasks in parallel and/or maintaining an order.

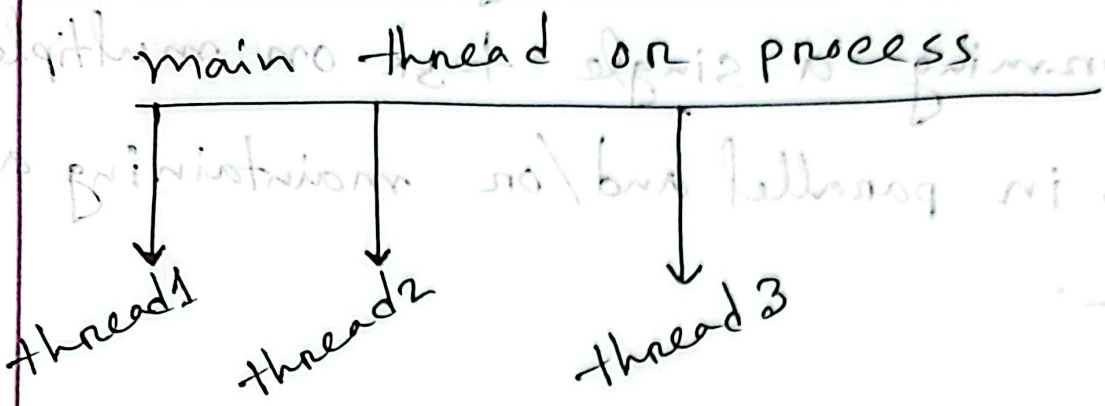
Thread

Similar to process

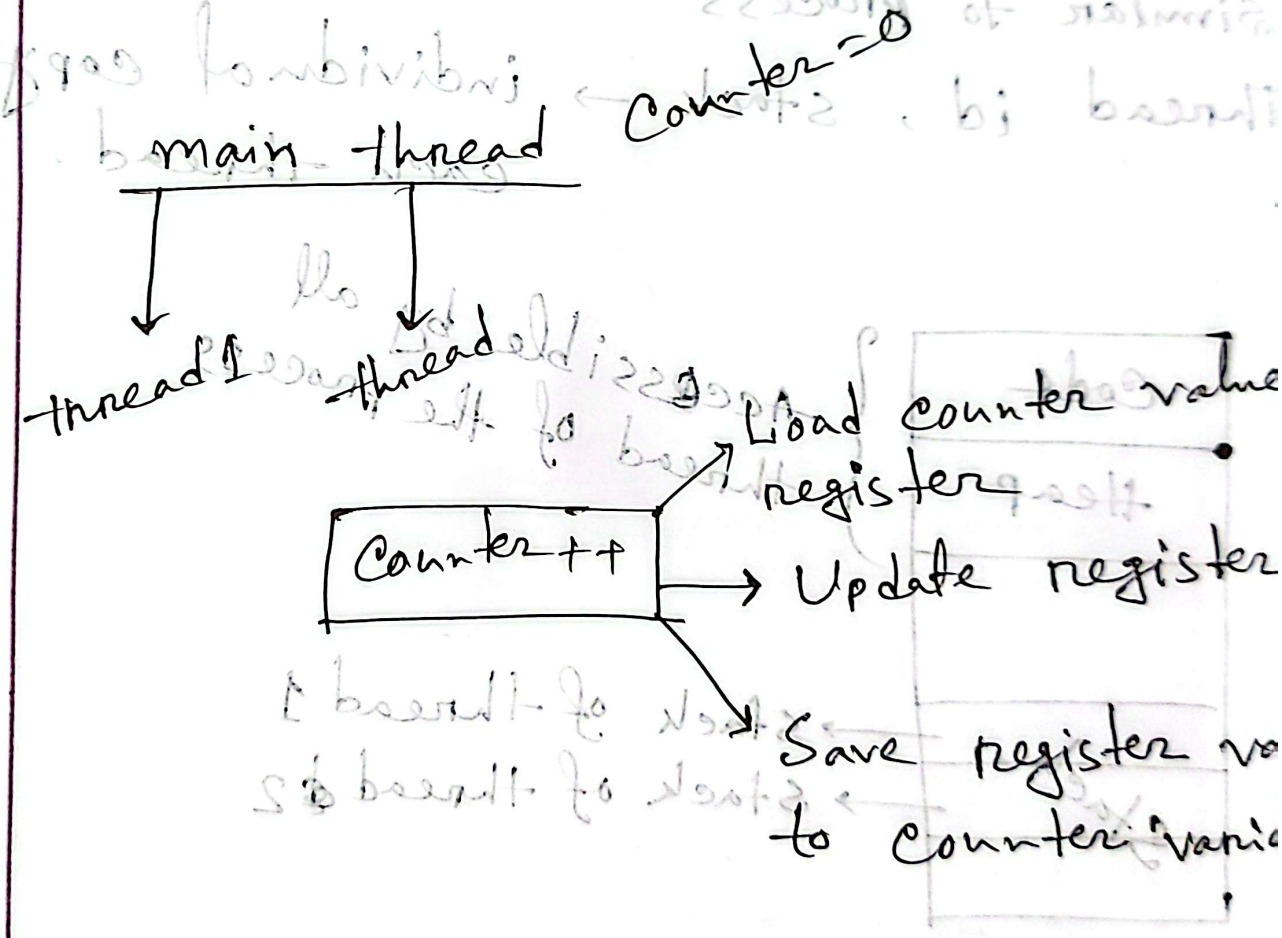
Thread id, stack → individual copy for each thread.

Process





Total thread = 4



Mutual exclusion: If one thread is
critical section (using shared data)
threads must wait for it to comple

→ Achieved with locks / mutex / se

```
lock(m);    //m is a lock variable
counter++;
unlock(m);
```

Ordering or Synchronization

* Ordering tasks among threads
* Wait for previous task to comp
  cond - w
* Signal a/all waiting threads not
  the task is completed → cond
  /cond-brodcast(c),
  ↓ notifies all
  waiting threads

Example of ordering

① If produced ① then can be consu ②

② If bus arrives ① passengers can boa ②

③ If door is opened ① then guests ca

enter ②

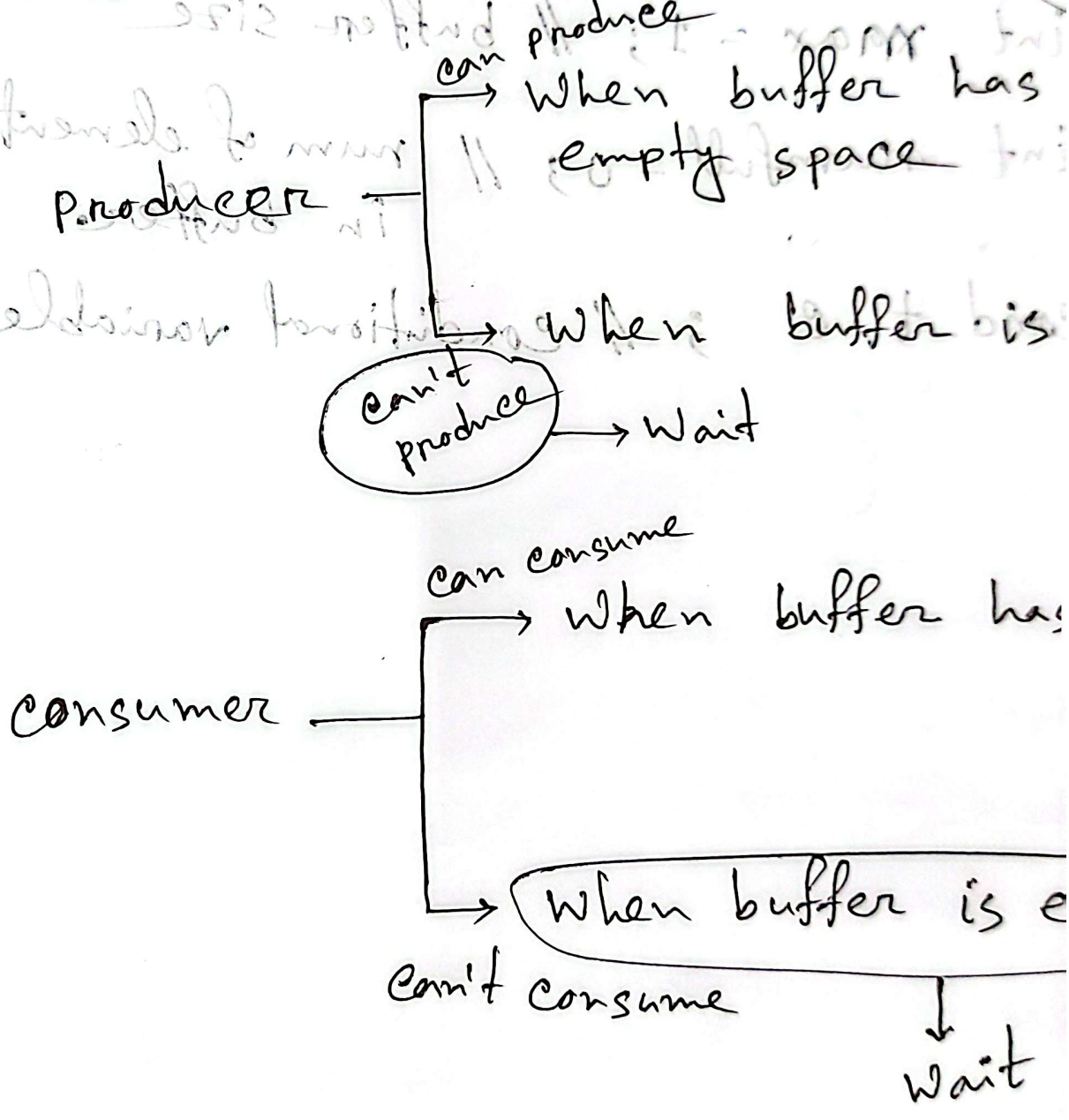
```
mutex m;
lock(m);
cond_wait(c, m);
        ...
unlock(m);
```

Producer / Consumer Problem

1 producer

1 consumer

buffer ~~size~~ with size 1.



Implementation: 1 producer, 1 consum

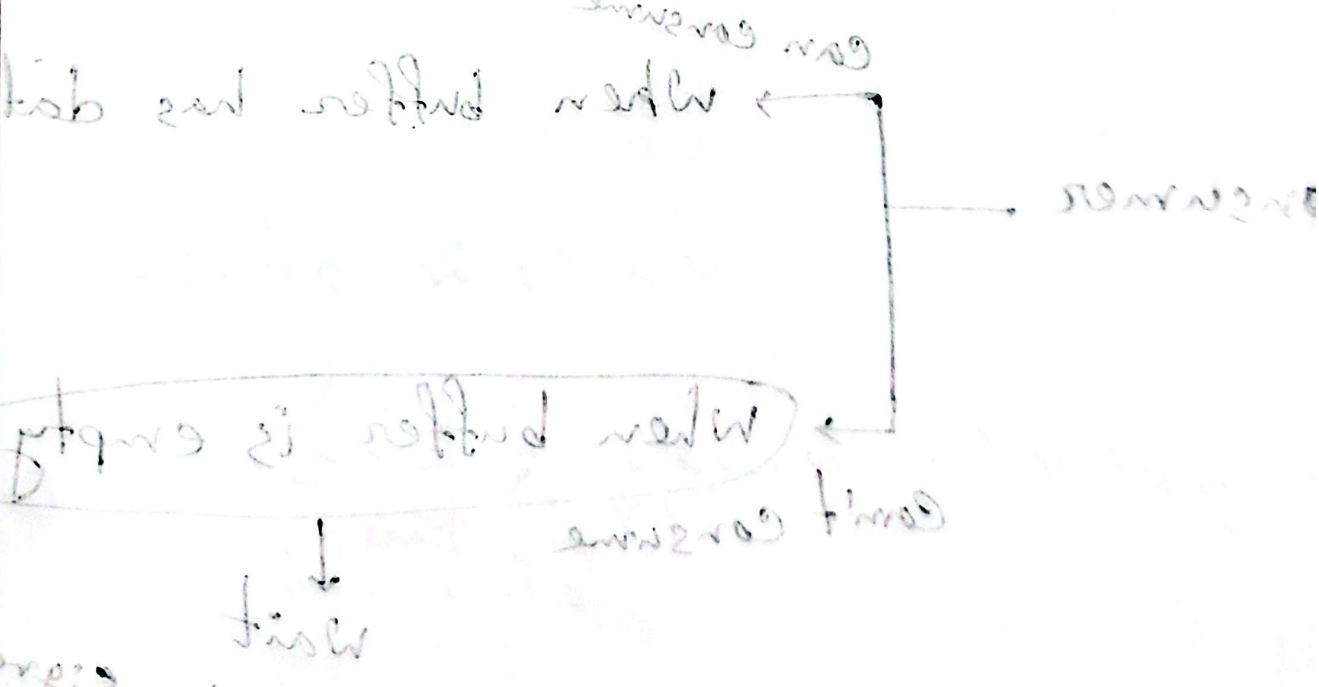
buffer size 1

mutex m; // lock variable

int max = 1; // buffer size

int numfull = 0; // num of elemen in buffer.

cond_t c; // conditional variabl

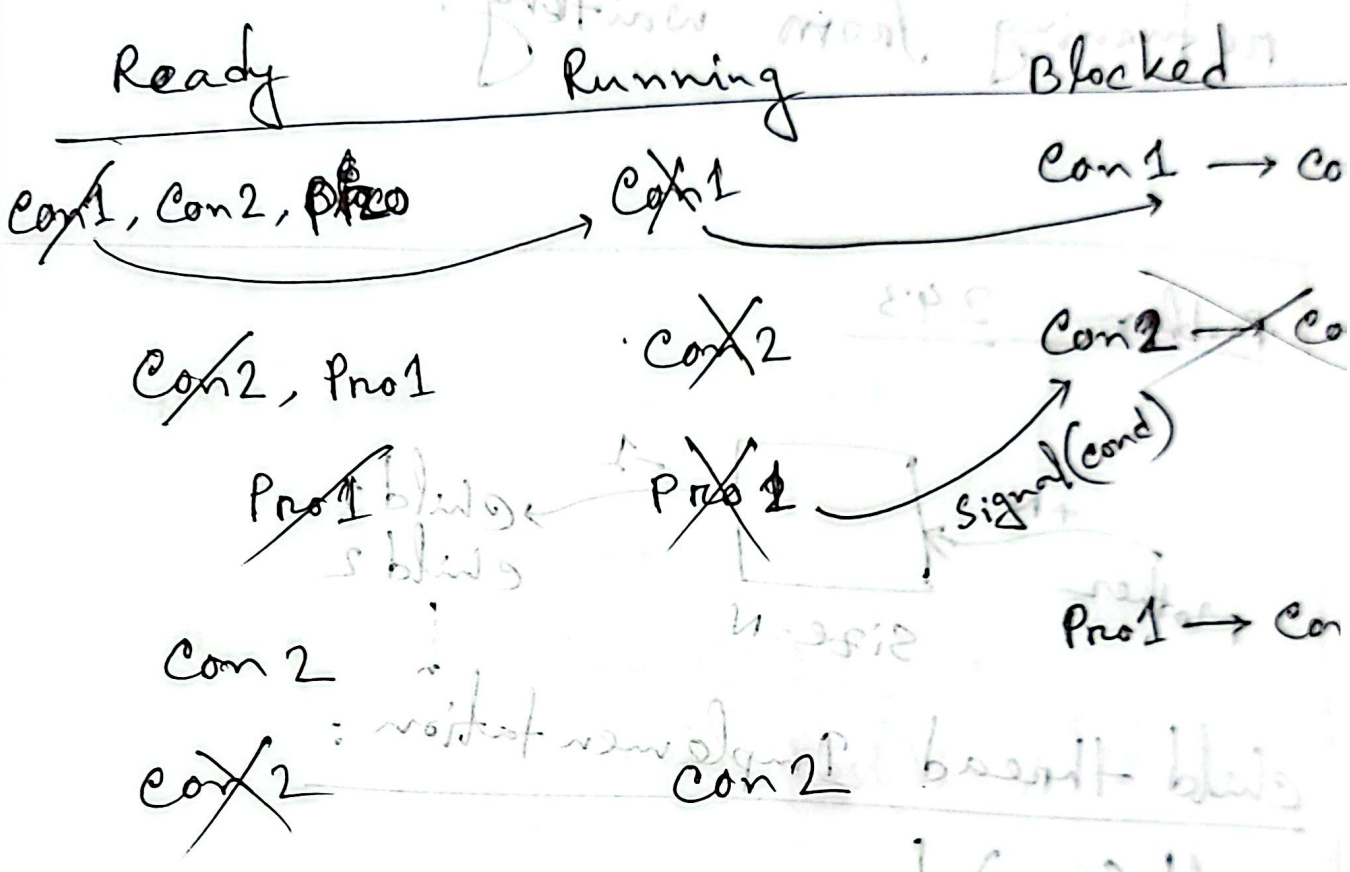


```
void * producer(){
for(int i=1; i<=10; i++){
    lock(m);
    if(numfull == max)
        cond_wait(c,m);
    producer(numfull);
    cond_signal(c);
    unlock(m);
}
```

```
Void *cosumer (){
    for(int i=1; i<=10
    {
        lock(m);
        if( numfull ==
            cond_wait(
        consume(nu
        cond_signal(
        unlock(m)
    }
}
```

20/08/20

Producer 1, Consumer 2.

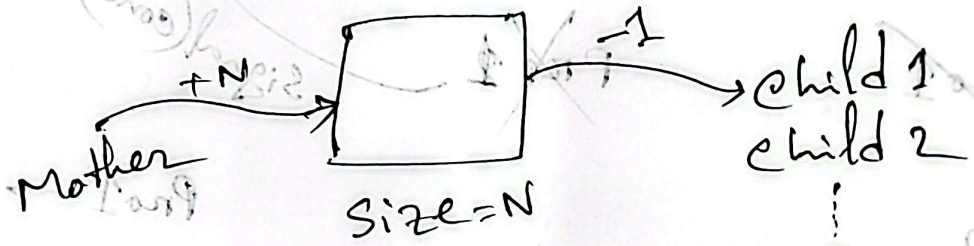


* Use different conditional variables for types of threads

Producer threads : 3
Consumer threads : 4
} Total threads = Different type

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

Problem 243



child thread Implementation:

```
While (true) {
    lock (m);
    while (count == 0){
        signal (emptyBox);
        wait (fullBox, m);
    }
    getChocolateFromBox ();
    count--;
```

```
    eat();
    unlock(m);
}
```

25.1

```
void *guest(void *args){
    lock(m);
    guest_count++;
    signal(cv_host);   // if(guest == signal(
    wait(cv_guest, m);
    signal(cv_guest); // broadcas
    enterHouse();
    unlock(m);
}
```

SPring 25, CT 2, Set D

24/08/

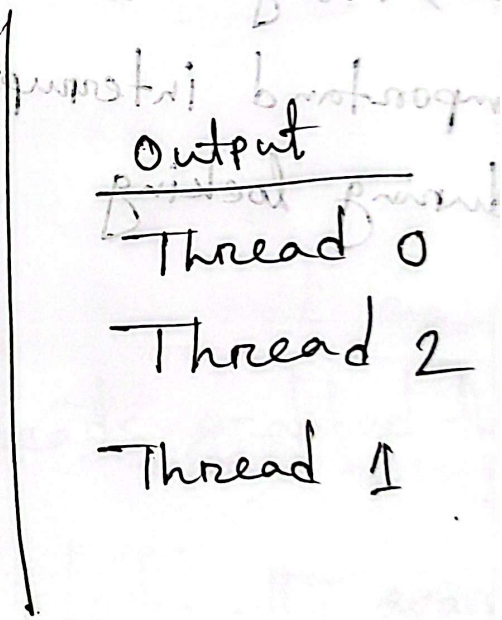
```
function bus_thread(){
    lock(mutex)
        bus_arrived = true;
    int boarded = min(K, waiting_count);
    while bus arrived:
    waiting_count -= boarded;
    for(int i=1; i<=boaded; i++){
        signal(bus_arrived_cond);
        wait(&passenger_boaded_cond,&
    }
    bas_arrived = false;
    depart();
unlock(mutex)
```

Mid 25I - 4(a) : Determining output from concurrency code.

Thread 0 → ~~K=0~~ i = 2, K = 2 % 3

Thread 1 → ~~i=2, K=2%3=2~~, signal (c[2]), done[2] = 0, i =

~~Thread 2~~ → signal (c[1])

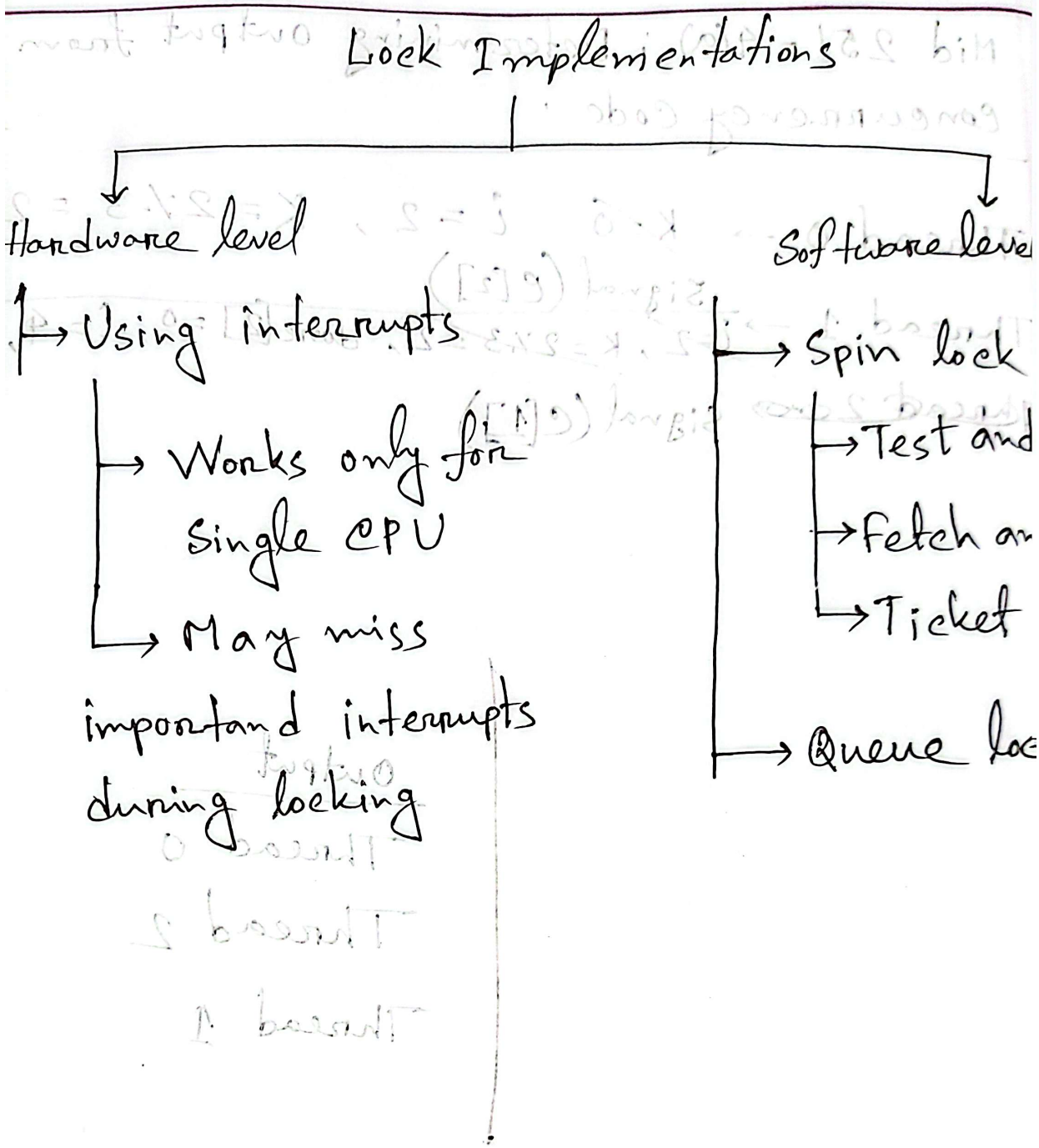


Output

Thread 0

Thread 2

Thread 1


Lock Implementations
Hardware level
Using interrupts
Works only for single CPU
May miss important interrupts during locking
Software leve
Spin lock
Test and
Fetch an
Ticket
Queue lo

Bug in spin lock

Thread 1

```
while (*lock);
```

Descheduled

```
*lock = true
```

Thread 2

```
While (*lock
*lock =
```

Desche

Bug: Both threads grabbed the lock

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

```
While (TestAndSet(&lock, 1) == 1);
    ↳ int old = *lock
      *lock = 1
      return old;
```

lock Availat
n unavai

```
Void acquire(int *lock){
    while(TestAndSet(lock, 1) == 1)
    {
    }

Void release(int *lock)
{
    *lock = 0
}
```

Thread 1

```
acquire(&lock)
Counter++
release(&lock)
```

[Compare and swap Spinlock

4. 2nd Problem with Spin Lock : No guarant
every thread will get lock eventually

Thread 1

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

Thread

```
acquire(&
counter+
release(&
```

If thread 1 is descheduled each ti
holding the lock, then thread 2 will
the lock and will be in starvation

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

turn
~~ticket~~ → The running serial number (

~~ticket~~ → What's my serial number

ticket

Process Creation (fork, wait, exec)	scheduling
Focus on questions similar to 243 mid, 251 mid, 251 CT1	simulation Conceptual questions similar to 252 CT2 243 CT-2

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

writer-thread