New chapter

Threads

PC1 → name = input()

PC2 → x = 0
while x < 1000 :
x += 1.

to do the work, we need multiple counters so that independently it can work and finish the work.

Threads ⟶ multiple process counter.

Same Python program

PC3 → x = 0
while x > - 10000 :
x -= 1

PC1 → name = input()

PC2 → x = 0
while x < 10000 :
x += 1.

একটা thread এর local variable change হলে অন্যটায় change হয় না। So, we will have independently stack to store local variable.

Calculation করার জন্য, যদি thread change হয় So then we need, multiple registers, so that it doesn't lost the calculations.

Sometimes, it may need that, multiple threads may need to access some data. Then, we need to declare, the variable in Data section as global variable.

Code section has the whole code. Threads are just part of a whole code. So, no need to have different codes.

Benefits of threads

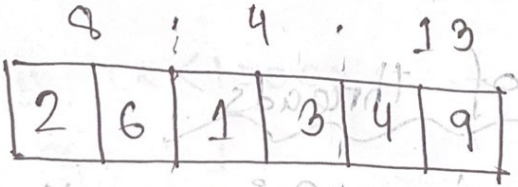
→ responsiveness : one thread may provide rapid response while other may be slow.

→ Sharing data : between threads we can share the variables.

→ Economy: Creating & managing threads is much faster than performing the same task for processes.

→ Scalability: A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be splitted.

Multi core Programming



Suppose have 3 CPU. {Saves time}. ***

Data parallelism, dividing data in small chunks and doing as single process.

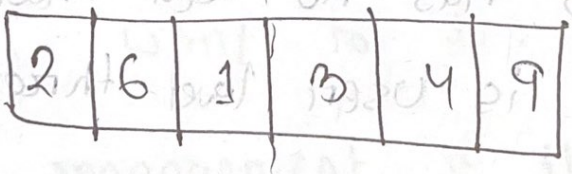
2	6	1	3	4	9

2 thread.

i) sum of elements
ii) product of elements.

Task parallelism → distributes threads across cores, each thread performing unique operation.

Data Parallelism → distributes subsets of the same data across multiple cores,



4 cores.

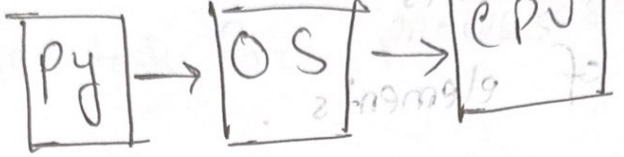
CPU 1 CPU 2

4 threads →

same task, data diff
- i) 2 6 1 add
- ii) 3 4 9 add

diff task, diff data
- iii) 2 6 1 product.
- iv) 3 4 9 product.

Multithreading Models



PC1 → name = input()

PC2 → x = 0

while x < 1000:

x += 1

The OS has no idea that what threading is. That is user level threads.

The entire threading is done inside the python program. Python multithreading library will decide which program will run for how much time, it doesn't bother the OS.

Pr 1 → $y = 0$

while < 2

$y += 1$

- - - - - - - - - -

Pr 2 → $x = 0$

while $x < 2$

$x += 1$

$x = 0$

$y = 0$

$x += 1$

$y += 1$

$x += 1$

$y += 1$

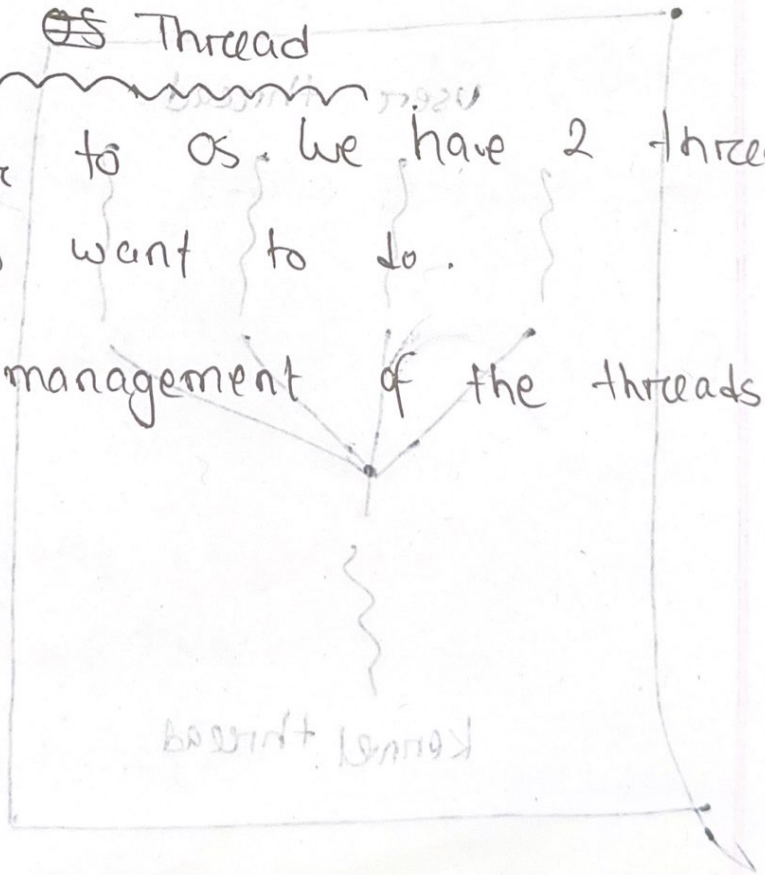
This is what OS will see.

Kernel level ~~OS~~ Thread

We surrender to OS. We have 2 thread.

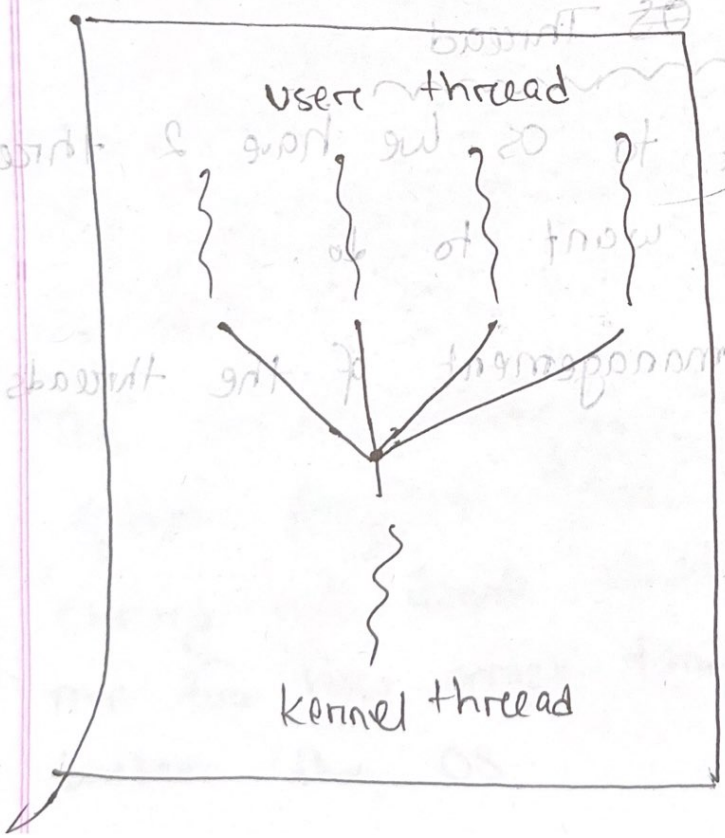
Do what you want to do.

OS do the management of the threads.



```
x = 0
while x < 10
    x += 1
    fork()
. . . . . .
y = 0
while y < 10:
    y += 1
    fork()
```

```
def fork()
kernel program counter  KPC →
```



many to one model.

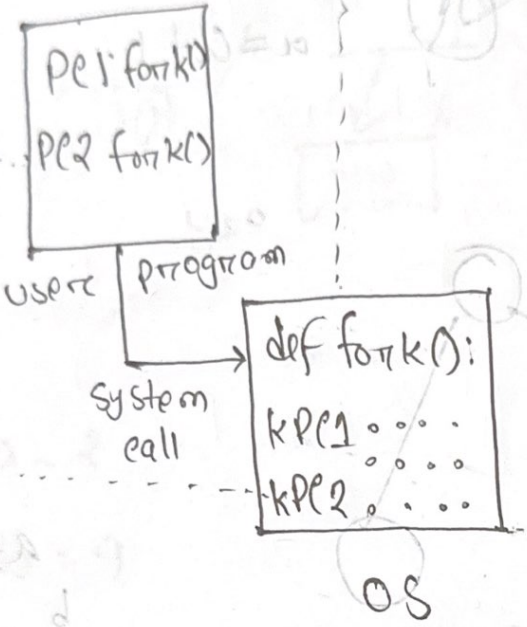
Many to one user thread means. After doing the work of 1st thread then it will be moved to another as kernel is single.

One to One model

CSE 321

Date: 08/07/25

Multithreading models



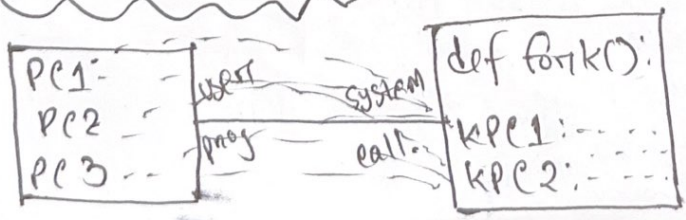
many to one

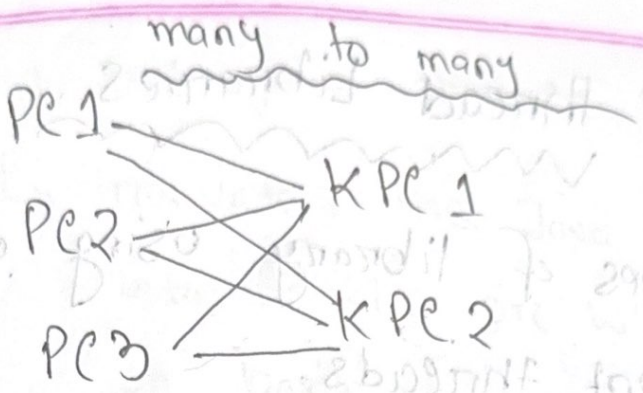
Only one thread যেকোন্ operating system block হলে বাকিগুলো।

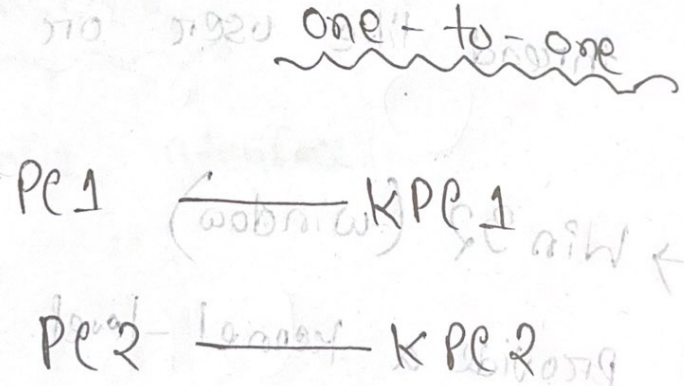
one to one

multiple threads একসাথে operating system এ কাজ করতে পারে, need as many kernel as thread.

many to many







Two level Model

Combination of many-to-many as well as one-to-one.

Thread Libraries

3 types of library using we can implement threads.

↳ posix (Linux based).

gives liberty to implement a thread like user or kernel level.

↳ Win 32 (window)

provide a kernel-level library on window system.

↳ Java threads.

Can run on any OS very smoothly. Linux or windows no worries. Cross platform.

JVM → Java virtual machine a virtual OS on my OS.

In JVM, the java threads is implemented.

↳ translates the Java threads into POSIX or Win32 lib on basis of underlying OS.

Java Threads

public interface

Program

include = import.

cannot do anything without a function.

def = int main().

There is nothing called : → { → opening

also } → closing.

```
void *funcThread (void* arg);

int main() {
```

pthread_t t1 → name of variable.

type of variable

pthread_create (&t1, NULL, funcThread, NULL);

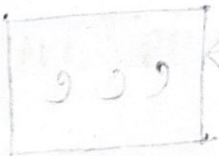
a function which creates a thread.

pthread_join (t1, NULL);

We can declare a variable in C but not assign a value to it.
This is a liberty in C.

BUT, we have to mention the type of the variable to declare.

```
main () {
thread → PC 1
}
function Thread () {
thread → PC 2
```



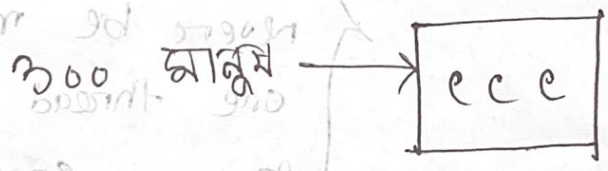
In the main fonction, there will never be more than one thread or program counter

This is how threading is called in C.

Implicit Threading

ask sir after class.

Thread Pool



কোন counter ফাকা হলে কোন আনুমি কোথায় যাবে এটি manualy declare করতে হবে। যদি না করি, we call thread pool which will do the work. Automatic thread management. A library.

fork ছাড়া

exec() call করলে new process create হয়ে new program run হয়,
Thread just 1 টি হয়,

exec() call না করলে ager টা যেটা thread আসে সেটাই হবে,

Signal Handling

Sending a notfication outside the program, to stop/wait the prog is signal.

Before sending the signal if the program wants to do any specific work then we can use signal Handling.

→ can send signals to all thread.
→ can send signals to one thread.
→ can send signal to few threads.

Thread Cancellation

When to cancel the thread.

immediately off → asynchronous cancellation.

Deferred cancellation → can off when your work is done.

CSE 321 Date: 13/07/25

CPU Scheduling

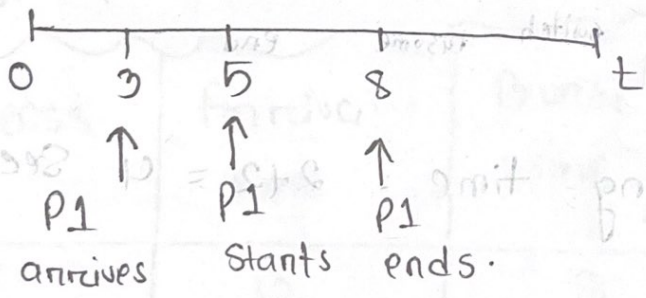
Which process should be running & which processes should be in the ready state.

Basic Concepts or Aim.

→ full utilization of CPU.

→ number of throughput should be increased.

→ Turnaround time.



Turnaround time = $T_{finish} - T_{arrival}$

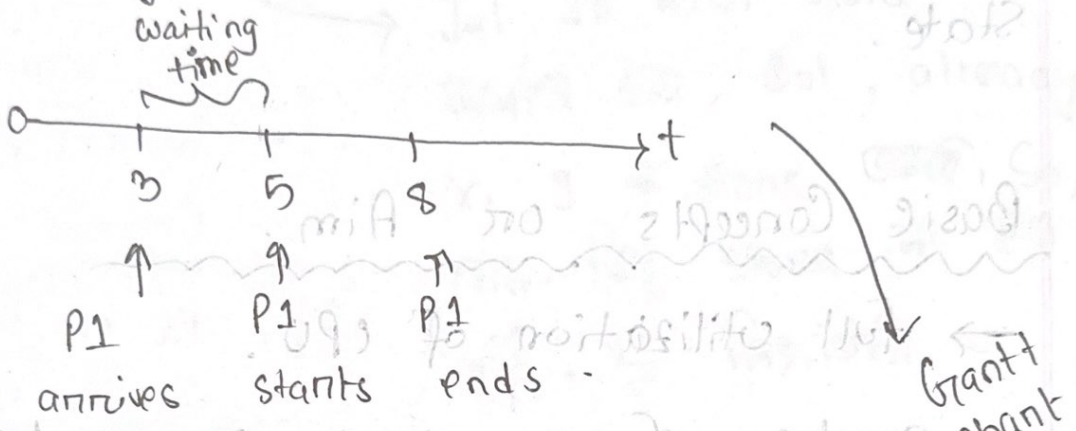
$= 8 - 3$

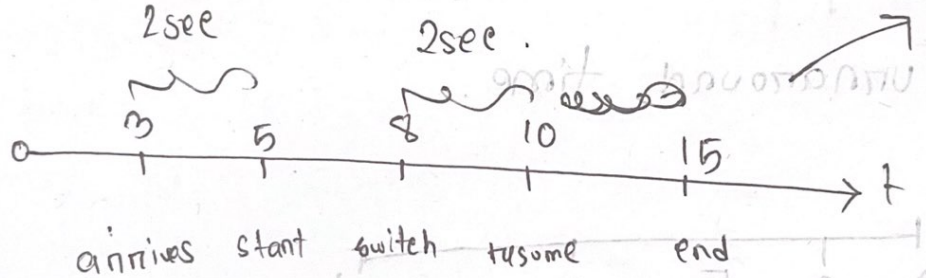
$= 5$ seconds

If we decrease it then, it is good.

→ waiting time.

amount of time a process waits in the ready queue.





∴ Total waiting time = 2+2 = 4 seconds.

→ response time.

= Tstart - Tarrival

The required amount of time, CPU needs to respond to the request of a process.

Optimization

→ max CPU

→ max throughput

→ min turnaround time
→ min waiting time.
→ min response time.

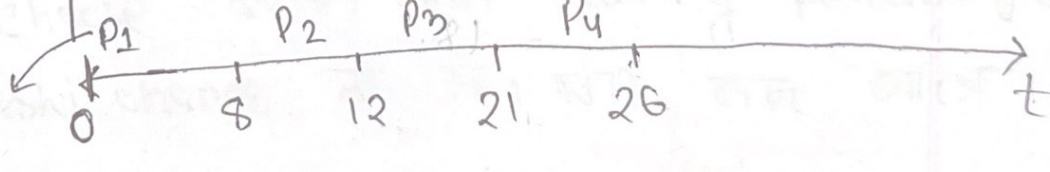
} we will work with these three.

CPU Scheduling Algo (FIFO)

First come First Serve

Process	Arrival time	Burst time / Execution time → spent in the CPU.
1	0	8
2	1	4
3	2	9
4	3	5

we select on basis of arrival time.



because arrival time 0. first come

For,

P1

turnaround = 8 − 0
= 8

waiting = 0

response = 0 − 0 = 0

P2

turnaround = 12 − 1
= 11

waiting = 8 − 1
= 7.

response = 8 − 1 = 7.

P3

turnaround = 21 − 2
= 19

waiting = 12 − 2
= 10

response = 12 − 0

P4

turnaround = 26 − 3
= 23

waiting = 21 − 3
= 18

response = 21 − 3
= 18.

	Turnaround	Waiting	Response
P1	8	0	0
P2	11	7	7
P3	19	10	10
P4	23	18	18
avg:	15.25		

In this (FCFS) algo, wait & response time is same.

	Arrival	Burst
P1	1	200
P2	101	2

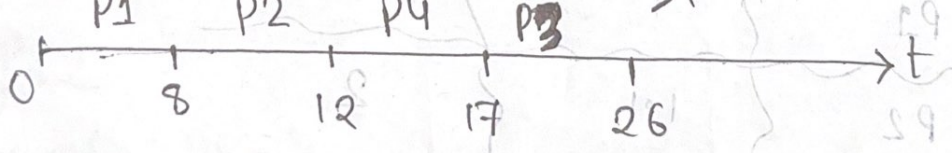
Cashier, যার কাছে আগে কিছু নাই, Cashier তাকে choose করবে। মানে waiting percentage যেন beshi change না হয়। যার কাছে আগে

Shortest Job first (SJF) / Shortest remaining time.

	Arrival	Burst
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

we decide, when a process ends

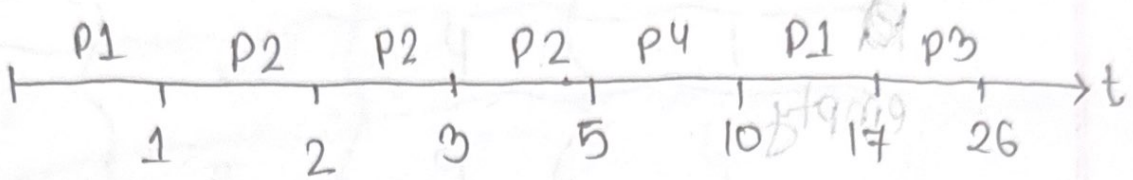
non-Preemptive scheduling
↳ Stop



	Turn
P1	8
P2	11
P3	24
P4	14

Shortest job first algo will give much better turnaround time than first come first serve.

Preemptive
we can decide, when process starts and ends.



	Arrival	Burst time / Remaining time
P1	0	~~8~~ 7
P2	1	~~4~~ ~~3~~ 2
P3	2	9
P4	3	5

	Turn	Wait	Resp
P1	17	9	0
P2	4		
P3	24		
P4	7		
Avg :	13		

Preemptive ⇒ wait and response time different.