OPERATING SYSTEMS (EE442)

PROCESSES & THREADS

Topics:

- Peterson's Solution (Critical Section problem)
- Test and Set Lock
- Semaphores
- Producer - Consumer Problem
- Dining - Philosophers Problem
- Readers and Writers Problem
- First Come First Served Algorithm (FCFS)
- Shortest Job First (SJT)
- Shortest Remaining Time First (SRTF)
- Round Robin Scheduling

② Peterson's Solution → Humble solution :)

int turn → whose turn it is to enter its CS (critical section)

boolean flag[2] → a process is ready to enter its CS

(ex: i^{st} P_i then P_j)

Process P_i

```
do {
    flag[i] = true;          ✓ ①
    turn = j;
    while ( flag[j] && turn==[j] );     , if j is ready
        critical section
    flag[i] = false;
        remainder section
} while (TRUE);
```

Process P_j

```
do {
    flag[j] = true; ✓        → final value of turn
    turn = i;
    while ( flag[j] && turn==[j] );     , if j is ready   (wait)
        critical section
    flag[j] = false;
        remainder section
} while (TRUE);
```

✓ Mutual exclusion (one process ✓, other ✗)

✓ Progress

✓ Bounded waiting (no indefinite waiting)

③ Test and Set Lock

Shared lock variable → 0, 1 ✓

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

do {
   while ( TestAndSet (&lock ));
      // do nothing
      // CS  (lock = TRUE currently)
      lock = FALSE;
      // remainder section
} while(TRUE);
```

lock = False initially

→ HWS no other process can enter

at that time other process can enter critical section

✓ Satisfies mutual-exclusion

✗ Bounded-waiting

④ Semaphores

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

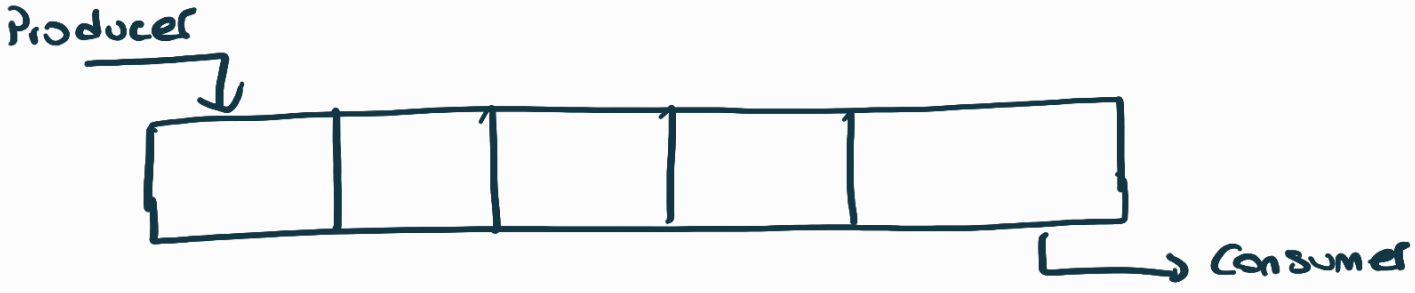
DOWN

```
signal()
V (Semaphore S) {
    S++
}
```

UP

Atomic Actions

⑤ Producer - Consumer Problem



Soution using semaphores:

1. m (mutex) : a binary semaphore acquire and release the lock
2. empty: a counting semaphore initial value = # of buffer slots
3. full : a counting semaphore, initially is 0.

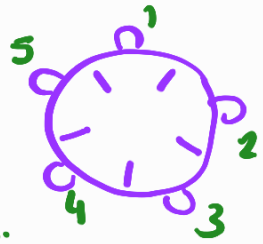
Producer

```
do{
    wait (empty);   // wait until empty>0   (then decrement empty)
    wait (mutex);   //acquire lock
    /*add data to buffer */
    signal (mutex); //release lock
    Signal (full);  //increment full
} while (TRUE)
```

Consumer

```
do {
    wait(full);  //wait until full>0   (decrement full)
    wait (mutex); //acquire lock
    /*remove data from buffer */
    Signal (mutex); //release lock
    signal (empty); //increment empty
} while (TRUE)
```

⑥ Dining - Philosophers Problem



Shared data: semaphore chopstick[5]; → all are initialized to 1.

```
do {
    wait ( chopstick [i] );
    wait ( chopstick [(i+1)%5] );

    //eat

    signal( chopstick [i] );
    signal ( chopstick [(i+1) %5] );

    //think
} while (TRUE)
```

} Left fork, right fork

* All philosophers take their left forks simultaneously
↓
DEADLOCK !

⑦ Readers and Writers Problem

1. mutex : a semaphore (init. to 1) used to ensure mutual exclusion when readcount is updated.
2. wrt (init to 1) a semaphore → common to reader and writer
3. readcount (init to 0) an integer

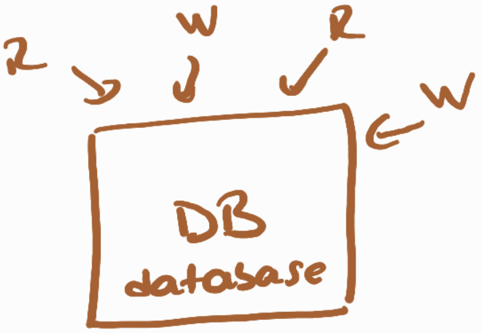
```
do {
    /* writer requests for CS
       wait (wrt); → wrt=0
    /* performs the write */
    /* leaves the CS
       signal (wrt); wrt=1
} while (TRUE)
```

→ Writer Process

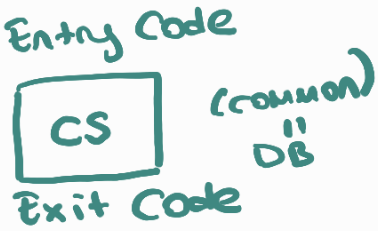
```
do {
    wait (mutex)                mutex = 0
    readcnt++;        1         , to ensure no writer can enter
    if (readcnt == 1) wait (wrt)
                              1
    signal (mutex)   → others readers can enter

    /* current reading */

    wait(mutex);
    readcnt --;   → reader wants to leave
    if (readcnt == 0) signal (wrt) → writers can enter
    signal (mutex);   // reader leaves
} while (TRUE)
```



- ! R – W → Case 1 ✓
- ! W – R → Case 2 ✓ Problems
- ! W – W → Case 3 ✓
- ✓ R – R → No problem → Case 4 ✓




```
int rc = 0;
semaphore mutex = 1;
semaphore db = 1;
void Reader (void) {        1 → 0 → R2
                            1 → 0
  while (true)              1 → 0
  {   down (mutex); → 0                 (R2)
      rc = rc + 1; → 1    1 1 2
                               0 → read is blocked
      if (rc == 1) then down (db); → 0
                                    1 → 0
      up (mutex) → 1 , 1
                        R2
      [DB] R1, R2 → 1
```

	RC	mutex	db
Case 1:	~~0~~ 1	~~1~~ ~~0~~, 1	~~1~~ 0

code of exit

```
down (mutex)
rc = rc - 1
if (rc == 0) then up (db)
up (mutex)
process_data
```

```
void Write (void) {
  { while (true) 1
    { down (db) ;
      DB
      up (db);
    }
  }
}
```

$W_2 = 0$ → ✗ W_2 is blocked

1 → 0

0

db = 0

no writing occurs

writes | W1

(W-R)

✓ Case 2:

rc: 0 → 1

db: 1 → 0

mutex: 1 → 0

(W_1-W_2)

Case 3:

rc: 0

db: 1 → 0

mutex: 1

(R-R)

Case 4

rc: 0 → 1

db: 1 → 0

mutex: 1 → 0 → 1 → 0

⑧ Dining Philosophers Solution using Monitors

- enum { thinking, hungry, eating } state [5];

state [i] = eating only if

state [(i-1) % 5] (LEFT) != eating

state [(i+1) % 5] (RIGHT) != eating

condition self [5];

DEADLOCK FREE SOLN

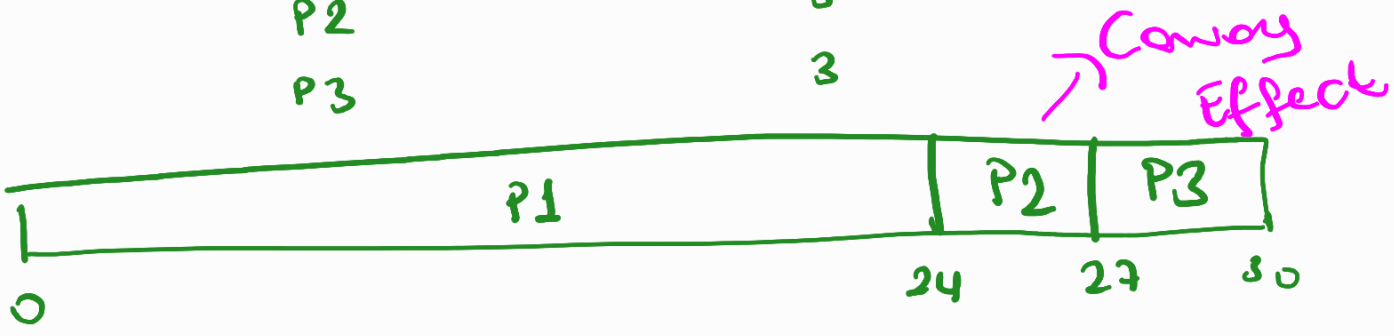
take_forks → test

(eat)

put_forks → test (LEFT), test (RIGHT)

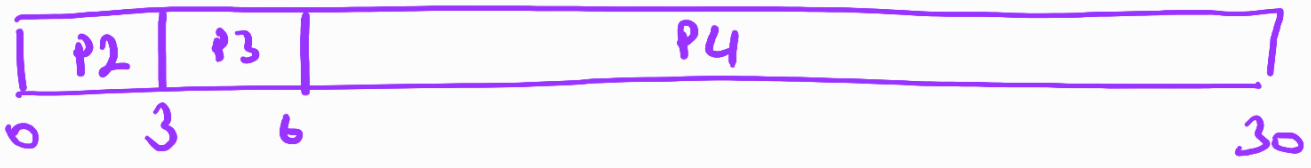
① FCFS

Process	Burst time (ms)
P1	24
P2	3
P3	3



Waiting Time for P1 = 0ms
" P2 = 24
P3 = 27
} Average = 17 ms

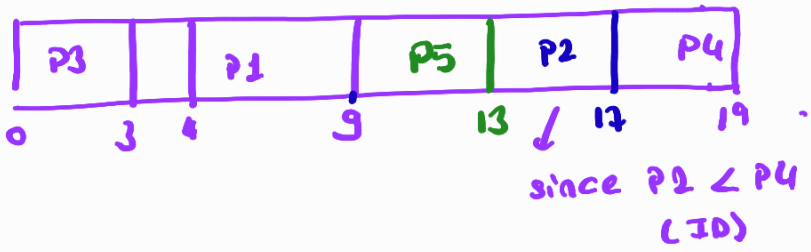
P2 arrive first this time



Waiting time P1 = 6
" P2 = 0
" P3 = 3
} Average 3 m/s

Ex

	ID	Arr. Time	Burst Time
②	✓ P1	4	5
	→ P2	6	4
①	✓ P3	0	3
	→ P4	6	2
③	✓ P5	5	4



since P2 < P4 (ID)

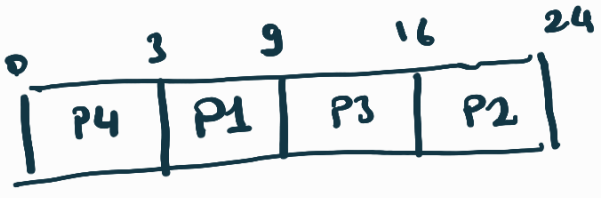
Turn Around Time = Completion time − Arrival time

Waiting Time = Turn Around Time − Burst time

② Shortest-Job-First (SJT)

Non-preemptive scheduling example:

ID	Burst time
P1	6
P2	8
P3	7
P4	3



Average WT (waiting time) : $\frac{(3+9+16+24)}{4} = 7$ ms

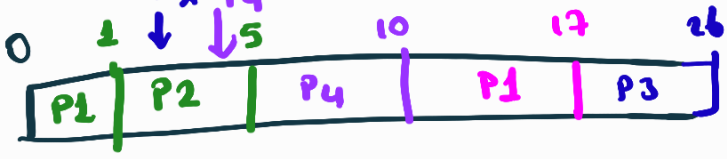
7 < 10.25 (FCFS)

Preemptive Ex

ID	AT	Burst time
✓ P1	0	~~8~~ 7 ~~0~~
✓ P2	1	~~4~~
P3	2	9
✓ P4	3	~~5~~

(1) 4<8 thus ~~P2~~ ← P2

P3 x P4

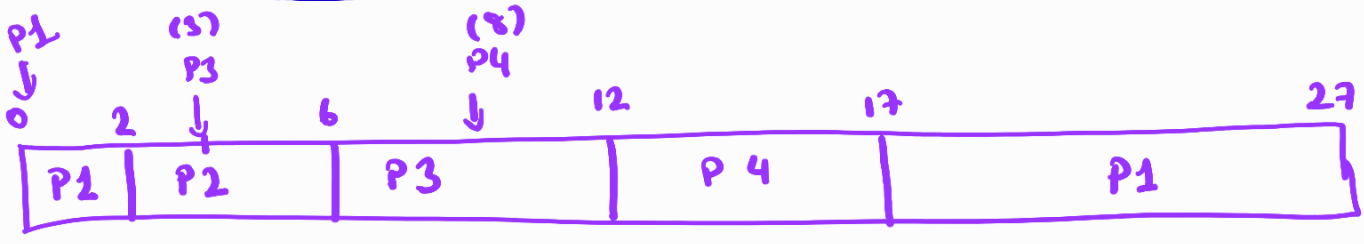


P3 > P2 → P2 still continues

9 > 7

Example 1

ID	Arrival Time	Burst Time	
P1	0	12	→ 10
P2	2	4	✓
P3	3	~~6~~ 4 att=8	✓
P4	8	5	



Average waiting time:

P1 → (17-2) = 15

P2 → (2-2) = 0

P3 → (6-3) = 3

P4 → (12-8) = 4

} $\frac{22}{4} = 5.5$ ms

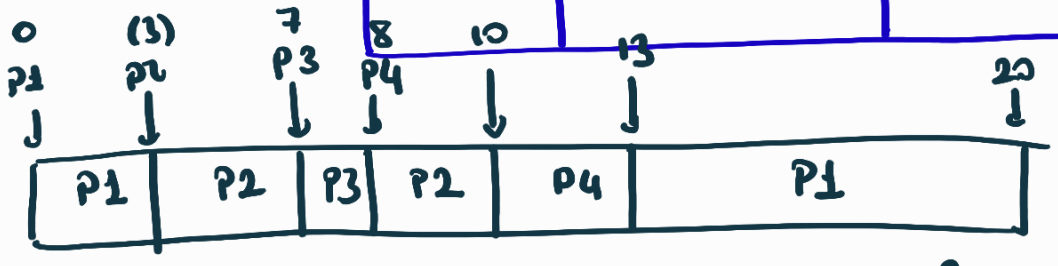
Waiting time = Total waiting TIME – No. of milliseconds process executed – Arrival time

Example 2

ID	Arr Time	Burst Time
P1	0	~~12~~ 7
P2	3	~~4~~ ~~20~~ ✓
P3	7	(1) ✓
P4	8	3

P2 < P3

P4 < P1



Turn around = Completion - Arrival

P1: (3-0) + (20-13) = 3+7 = 10

P2: (7-3) + (13-8) = 4+2 = 6

P3: 8-7 = 1

P4: 13-10 = 3

P1: 20-0 = 20

P2: 10-3 = 7

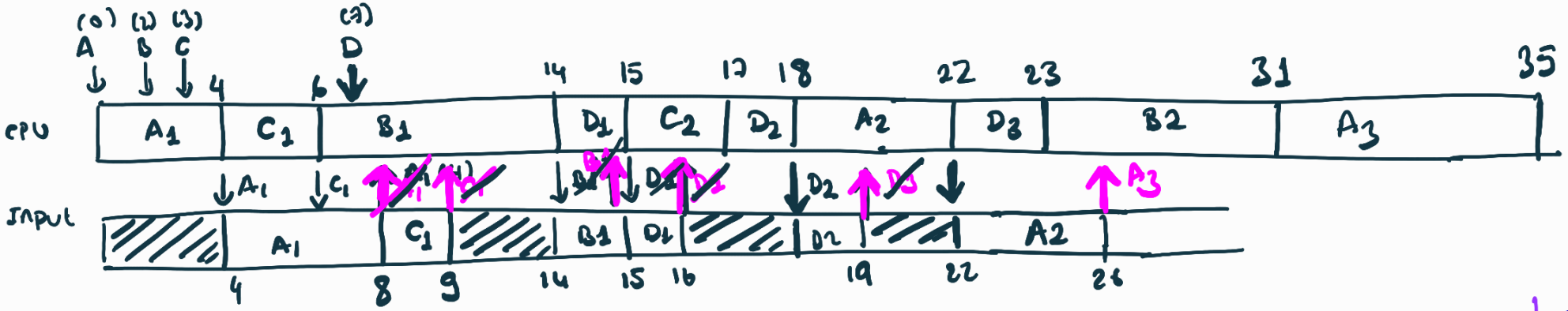
P3: 8-7 = 1

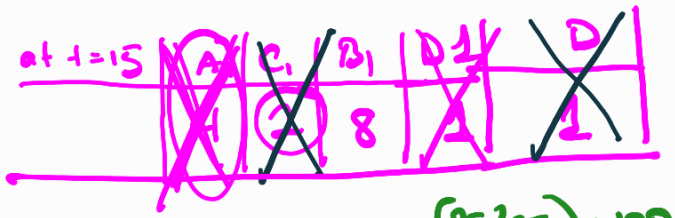
P4: 13-8 = 5

$\frac{33}{4} = 8.25$

Example (Non-preemptive) SJF

Process	AT	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0	4 ✓	4 ✓	4 ✓	4 ✓	4
B	2	8	–	8 ✓	–	–
C	3	2 ✓	–	2 ✓	–	–
D	7	1	–	1	1	1 ✓





Process Utilization: $(35/35) * 100 = 1$ ✓

Throughput: $4 / 35 = 0.11$ ✓ (# of process)

Turnaround Time = $t_{process\ comp} - t_{process\ submit}$

Response time = $t_{(first\ response)} - t_{(submission\ of\ request)}$

Turn-around Time

A: $35 - 0 = 35$

B: $31 - 2 = 29$

C: $17 - 3 = 14$

D: $23 - 7 = 16$

$ta_{avg} = \frac{35+29+30}{4} = \frac{94}{4} = 23.5$

Waiting Time

A: $(0-0) + (18-8) + (31-26) = 15$

B: $(6-2) + (23-15) = 12$

C: $(4-3) + (15-9) = 7$

D: $(14-7) + (17-16) + (22-19) = 11$

$wt_{avg} = \frac{34+11}{4} = \frac{45}{4} = 11.25$

Response time

0

$6 - 2 = 4$

$4 - 3 = 1$

$14 - 7 = 7$

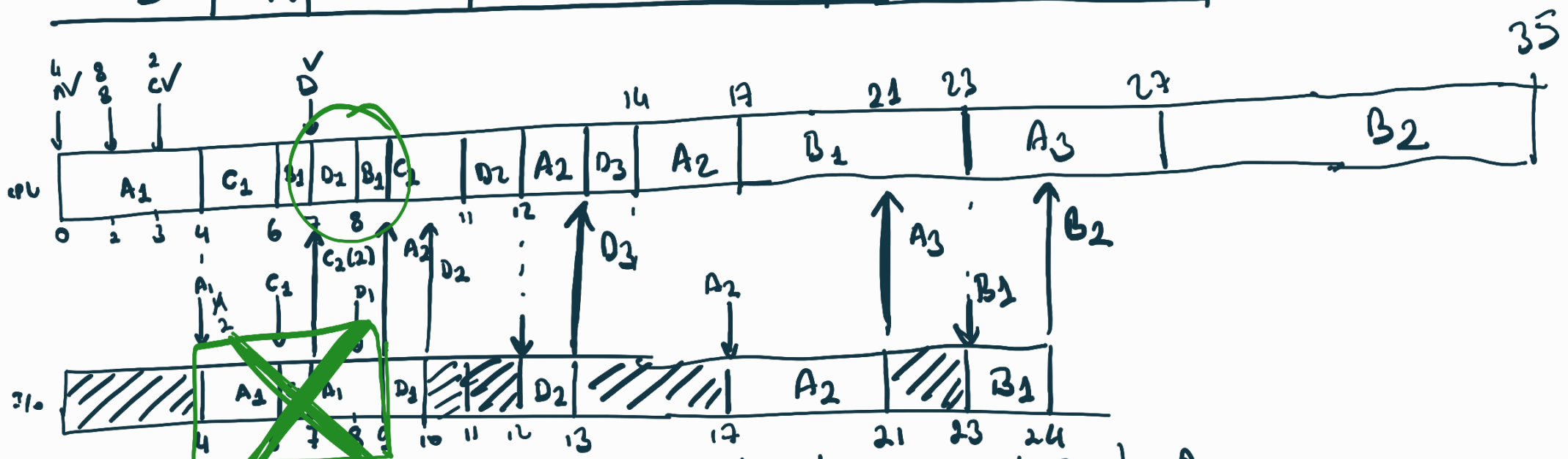
$rt_{avg} = \frac{12}{4} = 3$

Shortest-Remaining-Time-First

preemptive

Example

Process	AT	1st exec	1st IO	2nd exec	2nd IO	3rd exec
A	0 ✓	4 ✓	4 ✓	4 ✓	4 ✓	4 ✓
B	2 ✓	8 ✓	1 ✓	8 ✓	-	-
C	3 ✓	2 ✓	1 ✓	2 ✓	-	-
D	7 ✓	1 ✓	1 ✓	1 ✓	1 ✓	1 ✓



At t=7

	B1	C2	D1
remain	7	2	1 ✓

At t=9

	B1	C2	A2
	6	~~2~~ 1	4

t=10

	D1	C2	A2	D2
	6	~~1~~	~~4~~ 3	

t=13

	B1	A2	D3
	6	~~7~~	~~1~~

t=21

	B1	A3
	2	4

t=24

	B2	A3
	8	~~3~~

SRTF

Process	AT	1st exec	1st I/O	2nd exec	2nd I/O	3rd exec
A	0 ✓	4 ~~1~~ ✓	4 ✓	4 ✓	4 ✓	4 ✓
B	2 ✓	~~8~~ 6 ✓	1 ✓	8	-	-
C	3 ✓	2 ✓	1 ✓	2 ✓	-	-
D	~~2~~ 7 ✓	1 ✓	1 ✓	1 ✓	1 ✓	1 ✓

Turn Around timer

A: 27 − 0 = 27

B: 25 − 2 = 23

C: 11 − 3 = 8

D: 14 − 7 = 7

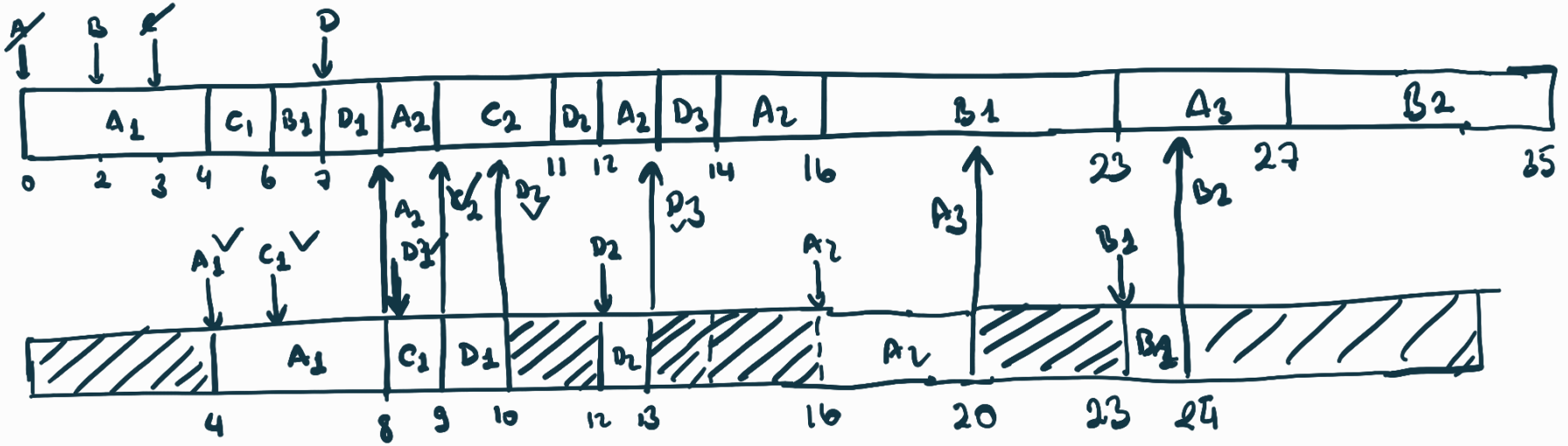
Response time

A: 0 − 0 = 0

B: 6 − 2 = 4

C: 4 − 3 = 1

D: 7 − 7 = 0



t=6

B	A2
7	4

t=9

B	A2	C2
7	4	2 ✓

t=11

B	A2	D2
7	~~4~~ 3	✓

t=13

B	A2	D3
7	3	1

t=17

B	A3
~~7~~	~~4~~ 3

t=25

A3	B2
3	8

Waiting time

A: (0−0) + (7−8) + (12−9) + (14−13) + (23−20) = 7

B: (6−2) + (16−7) + (27−24) = 9

C: (4−3) + (9−9) = 1

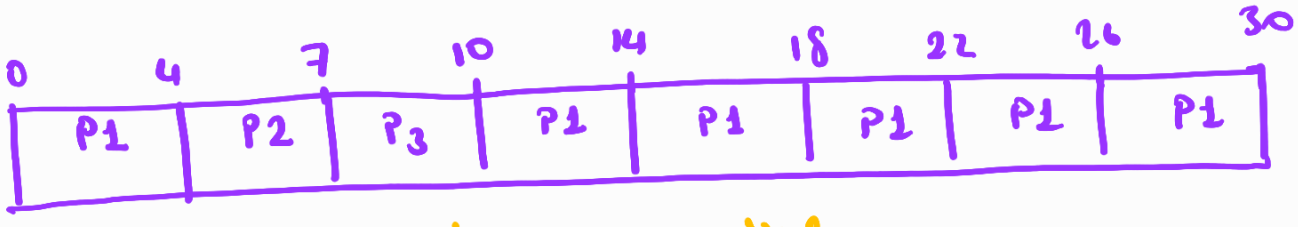
D: (7−7) + (11−10) + (13−13) = 1

③ Round-Robin Scheduling (Interactive Sys. Scheduling)

Ex

Process ID	Burst Time
P1	24
P2	3 ✓
P3	3 ✓

Time quantum: 4 miliseconds



Turn Around Time:

P1: 30-0 = 30

P2: 7-0 = 7

P3: 10-0 = 10

waiting time

P1: 30-24 = 6

P2: 7-3 = 4

P3: 10-3 = 7

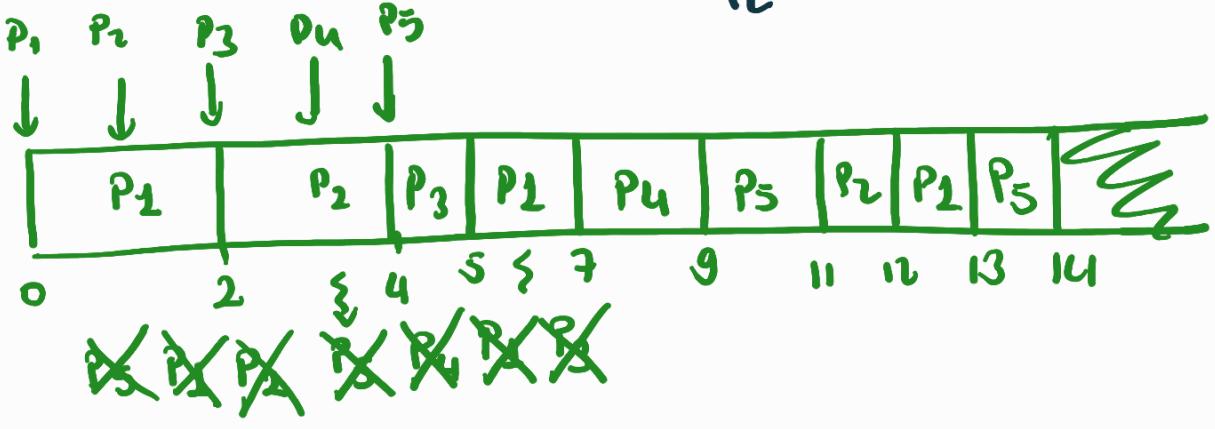
TA - BT → burst time

Ex

ID	Arv. Time	Burst Time
P1	0	~~5~~ ~~3~~ ~~1~~
P2	1	~~3~~ ~~1~~
P3	2	~~1~~
P4	3	~~2~~
P5	4	~~3~~ ~~1~~

12

Time quantum : 2 Units



RR Example

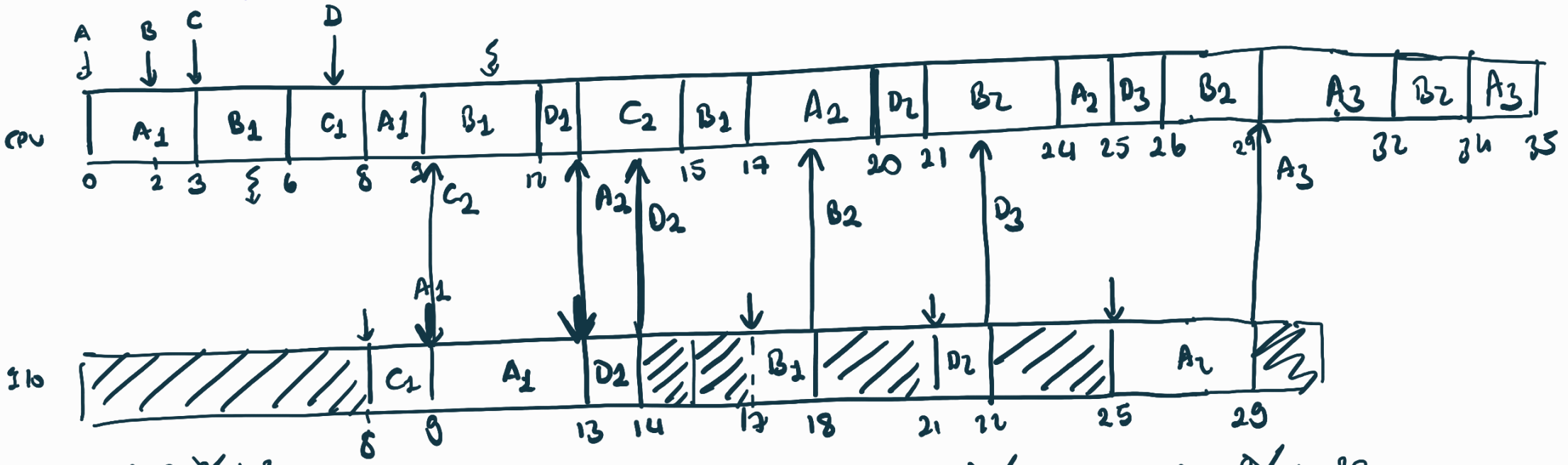
$rt_{max} \leq (n-1) * Q$ → time quantum
↙ response time ↘ multiprogramming degree

Time Quantum = 3

Process	AT	1st exc	1st I/O	2nd exc	2nd I/O	3rd exc
A	0 ✓	4 ✓	✓	4 ✗	✓	1 ✓ (1)
B	2 ✓	8 ✗ (2)	✓	8 ✗ (2)	–	–
C	3 ✓	2 ✓	1 ✓	2 ✓	–	–
D	7 ✓	1 ✓	✓	1 ✓	1 ✓	✓

Turn Around
A: 35 − 0 = 35
B: 30 − 2 = 31
C: 15 − 3 = 9
D: 26 − 7 = 14
Complete − arrived

Waiting time
A: (0−0) + (8−3) + (17−13) + (24−20) + (29−29) + (34−32)
B: (3−2) + (9−6) + (15−12) + (21−18) + (26−24) + (32−29)
C: (6−3) + (13−9)
D: (12−7) + (20−14) + (25−22)
↘ response time



A C ✗ t=3
B A ✗ t=6
D B ✗ t=8
C_2 D ✗ t=9
B_1 C_2 ✗ t=12

A_2 B_1 ✗ t=13
D_2 A_2 ✗ t=15
D_2 ✗ t=17

A_2 B_2 ✗ t=20
A_2 ✗ t=21
B_2 D_3 ✗ t=24

B_2 ✗ t=25
✗ t=26
B_2 ✗ t=29

A_3 ✗ t=32
A_3 t=34