National University of Computer and Emerging Sciences

FAST School of Computing **Fall-2023** **Islamabad Campus**

CS-2006: Operating Systems

Sections - A,B,C,D

Serial No:

Sessional Exam-II

Saturday, 7th November, 2023

Total Time: 1 Hour

Course Instructors

Ms. Maryam Shahbaz,

Mr. Muhammad Aadil Ur Rehman

Total Marks: 32

Student Name	Roll No.	Course Section	Student Signature	Signature of Invigilator

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

Instructions:

1. Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
2. No additional sheet will be provided for rough work. Use the back of the last page for rough work.
3. If you need more space, write on the back side of the paper and clearly mark question and part number etc.
4. After asked to commence the exam, please verify that you have Eight (8) different printed pages including this title page. There are a total of 4 questions.
5. **Calculator sharing is strictly prohibited.**
6. Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

	Q-1	Q-2	Q-3	Q-4	Total
Marks Obtained					
Total Marks	6	6	6	14	32

Question 1 [6 Marks]

1. In a multi-threaded file management system, multiple threads are responsible for performing various operations on files and directories. To ensure data integrity and prevent race conditions, a lock solution is employed for synchronization.

Updation process	Reading process
while(lock ! =0);	while(lock ! =0);
lock =1;	lock =1;
//open file	//open file
//append logs to file	//read the latest updated data from file
//include updation date in file	//write reading date in file
//close the file	//close the file
lock =0;	lock =0;

Issue faced : Mutual Exclusion [1 Mark]
Solution (Name) : Strict Restriction / Peterson Solution / Bakery Algorithm [1 Mark]
Rewrite the code while handling this issue to resolve critical section problem [2 Marks]

Updation process (CAN write peterson / bakery algorithm)	Reading process
while(TRUE)	while(TRUE)
{	{
while (turn != 1); critical_section();	while(turn != 2);
turn = 2;	critical_section();
noncritical_section();	turn = 1;
}	noncritical_section();
	}

2. Peterson's solution solves the synchronization problem, Correct the following code for errors or add missing statements (if any).[2]

Process 0	Process 1
while(TRUE)	while(TRUE)
{ interested[0] = TRUE ;	{ interested[1] = TRUE ;
turn = ~~0~~ 1;	turn = ~~1~~ 0;
while(interested[1]==TRUE && turn == 1);	
critical_section();	while(interested[0]==TRUE && turn == 0);
interested[0] = FALSE;	critical_section();
noncritical_section();	interested[0] = FALSE;
}	noncritical_section();
	}

Question 2 [6 Marks]
Consider this code, and write the answer to the given questions?

Code

```
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 4

int x = 29;
int m = 0;

void* child_code(void* arg)
{
   while (x > 0) {
      x -= 2;
      m++;
   }
   pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
   int rc;
   pthread_t child_threads[NUM_THREADS];
   pthread_attr_t attr;
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr,
PTHREAD_CREATE_JOINABLE);
   for (int i = 0; i < NUM_THREADS; i++) {
      rc = pthread_create(&child_threads[i], &attr,
child_code, NULL);
      if (rc) {
         printf("Error creating thread %d; return code:
%d\n", i, rc);
         return 1;
      }
   }
   pthread_attr_destroy(&attr);

   x += 7;

   for (int i = 0; i < NUM_THREADS; i++) {
      pthread_join(child_threads[i], NULL);
   }
   printf("x has changed to %d\n", x);
   printf("m has changed to %d\n", m);
   pthread_exit(NULL);
}
```

Questions

Assuming that there is no error in code. Write the expected output of the code. Also, analyze whether the code ensures thread safety. Identify and describe any race conditions or concurrency problems that might arise in this code. [3]

solution1 : x has changed to 0
m has changed to 18

solution2 : x has changed to 6
m has changed to 15

No, it doesnot ensures thread safety.
Concurrency problem occur as x is updated in both thread and main function, It should be synchronized.

Write the name of any 2 functions to change the attributes of a pthread? [2]

1. pthread_attr_setstacksize
2. pthread_attr_setdetachstate

What would happen if we remove the following code line from this code? [1]
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

Output doesn't change.
Removing this line might cause confusion or potential errors if later modifications depend on the assumption of joinable threads.

Question 3 [6 Marks]

A. Consider the following scenarios and choose which interprocess communication approach is preferable and why? (Marks of approach will only be given , is reason is correct) [3]

1. Scientific simulations, such as climate modeling or fluid dynamics simulations, require fine-grained communication to exchange data about specific simulation parameters, variables, or time steps
   Approach : Message Passing

   Reason
   Message passing enables controlled data sharing among processes, ensuring precise exchange of simulation details and variables, vital for complex scientific simulations requiring fine-grained communication.
2. Network protocols and applications that require low-latency communication, such as online gaming and financial trading platforms, must minimize communication overhead to ensure rapid data transfer and response times.
   Approach : Message Passing / Shared memory
   Reason
   Message Passing (then reason will be low latency communication)
   Shared memory (then reason will be to minimize communication overhead)
3. Distributed database systems often consist of multiple database nodes distributed across a network. IPC is required for nodes to coordinate distributed transactions, share data, and replicate data across the network.
   Approach : Message Passing
   Reason
   Message passing is typically preferred in distributed database systems for interprocess communication between nodes due to its ability to handle communication between remote entities efficiently and reliably over a network, enabling coordination for transactions, data sharing, and replication across distributed environments.

B. Considering a scenario where Parent wants to write to pipe, while child wants to read from pipe, assuming that parent runs before child [3]

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int pipefd[2];
    pid_t child_pid;
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    child_pid = fork();
    if (child_pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }
```

```
        if (child_pid == 0) {
          close(pipefd[1]);
           char buffer[100];
           int bytes_read = read(pipefd[0], buffer, sizeof(buffer));
           if (bytes_read > 0) {
               printf("Child received: %.*s\n", bytes_read, buffer);
           }
         printf("\nChild Complete the reading");
           close(pipefd[0]);
           exit(EXIT_SUCCESS);
        } else {
           close(pipefd[1]);
           char message[] = "Hello from parent!";
           write(pipefd[1], message, sizeof(message));
           close(pipefd[0]);
           wait(NULL);     }
        return 0;
    }
```

1. What is output of this code?

Child Complete the reading

2. Is their any logical error in this code? If yes, identify it.

yes, close(pipefd[1]) instead of close(pipefd[0]) , so unable to write as pipe is closed from writing.

3. Write the correct code, and then write its output?

close(pipefd[0]) before writing in pipe,

Output:

Child received: Hello from parent!

Child Complete the reading

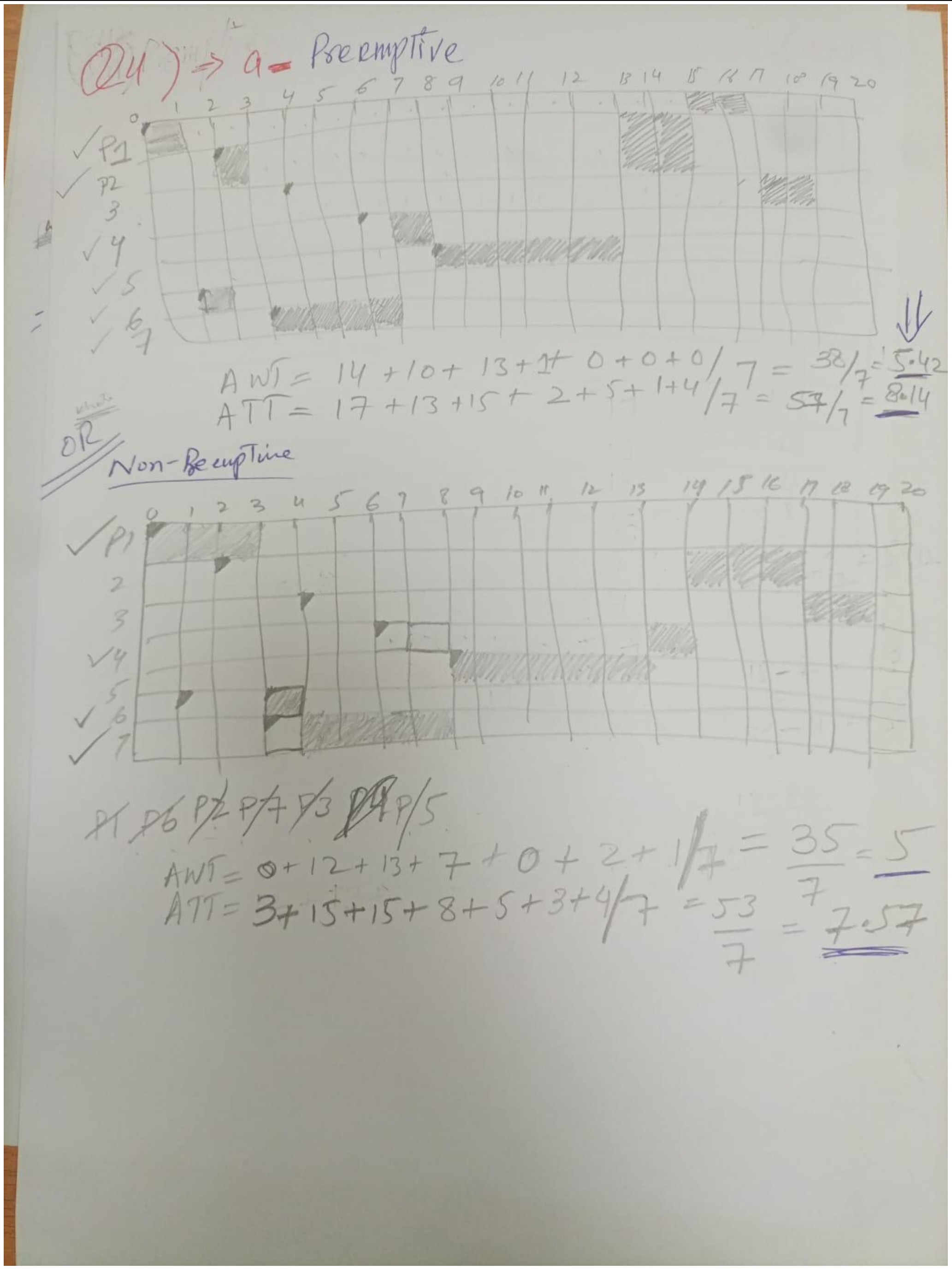
Question 4 [14 Marks]

Considering the table of process provided below, answer the following questions:

Process	Arrival Time	Initial Priority (0 = lowest)	Burst Time
P1	0	1	3
P2	2	2	3
P3	4	1	2
P4	6	4	1
P5	8	5	5
P6	1	3	1
P7	3	4	4

a. What would be the Average Waiting Time and Average Turnaround Time if Priority Scheduling is applied. [1+1+2 Marks]

A.W.T = (14) + (10) + (13) + (1) + (0) + (0) + (0) /7 = 5.42
A.T.T = (17) + (13) + (15) + (2) + (5) + (1) + (4) / 7 = 8.14
Gantt Chart



Q4) → a- Preemptive

$AWT = 14 + 10 + 13 + 1 + 0 + 0 + 0 / 7 = 38/7 = 5.42$

$ATT = 17 + 13 + 15 + 2 + 5 + 1 + 4 / 7 = 57/7 = 8.14$

OR

Non-Preemptive

~~P1~~ ~~P6~~ ~~P2~~ ~~P7~~ ~~P3~~ ~~P4~~ ~~P5~~

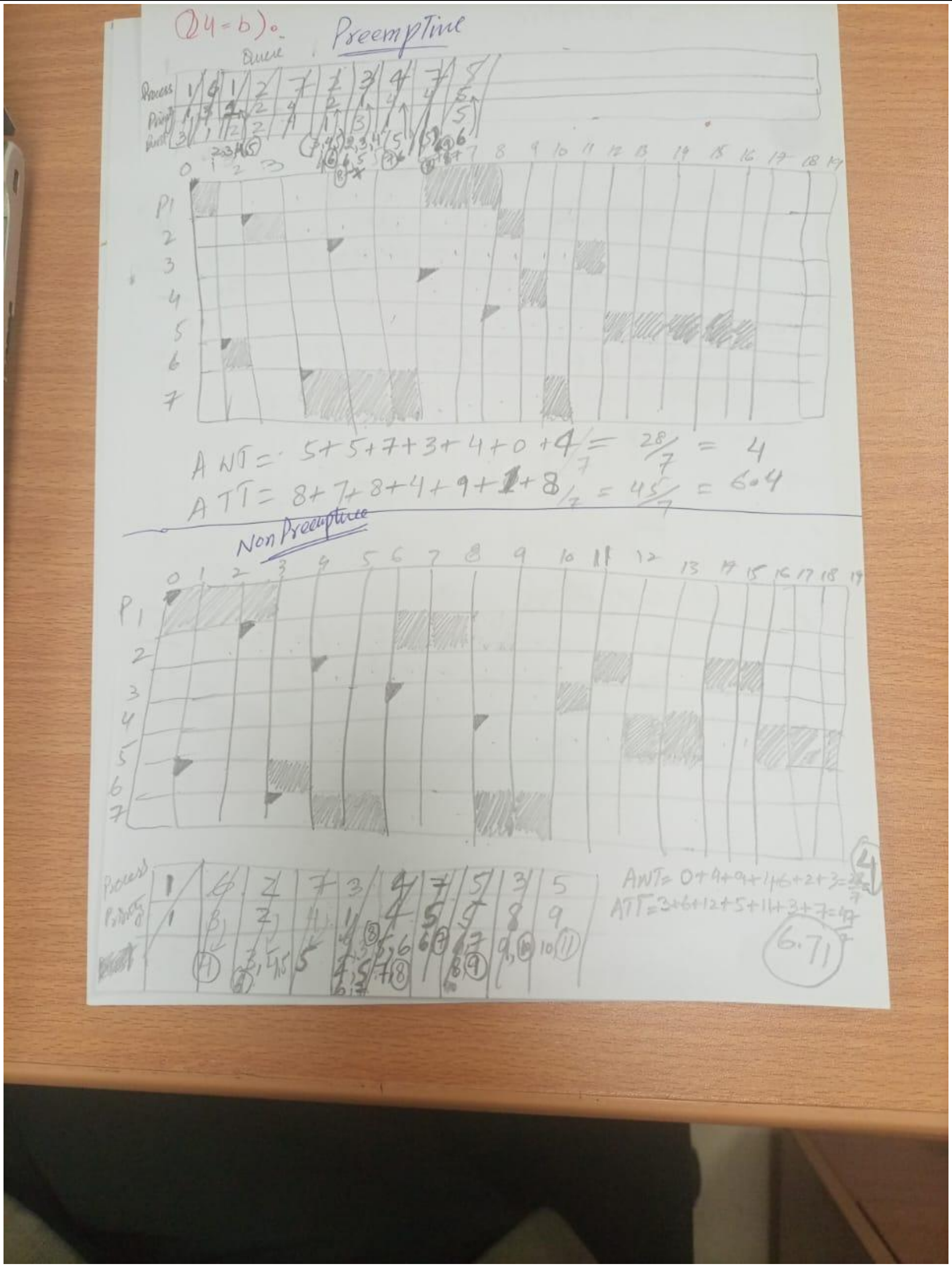
$AWT = 0 + 12 + 13 + 7 + 0 + 2 + 1 / 7 = \frac{35}{7} = 5$

$ATT = 3 + 15 + 15 + 8 + 5 + 3 + 4 / 7 = \frac{53}{7} = 7.57$

b. What would be the Average Waiting Time and Average Throughput time if Priority Scheduling with aging is applied. For aging you can say that a priority of a waiting process is incremented after 1 unit of execution. [1+1+2 Marks]

A.W.T = ( ) + ( ) + ( ) + ( ) + ( ) + ( ) + ( ) = ______________________
A.T.T = ( ) + ( ) + ( ) + ( ) + ( ) + ( ) + ( ) = ______________________

Gantt Chart:



AWT = 5+5+7+3+4+0+4 / 7 = 28/7 = 4

ATT = 8+7+8+4+9+1+8 / 7 = 45/7 = 6.4

AWT = 0+9+9+1+6+2+3 = 28/7 = 4

ATT = 3+6+12+5+11+3+7 = 47/7 = 6.71

Suppose that all processes have equal priority in the table provided above, which scheduling strategy would be suitable in this case.[2]

Round Robin or Priority Based Round Robin

Justification:

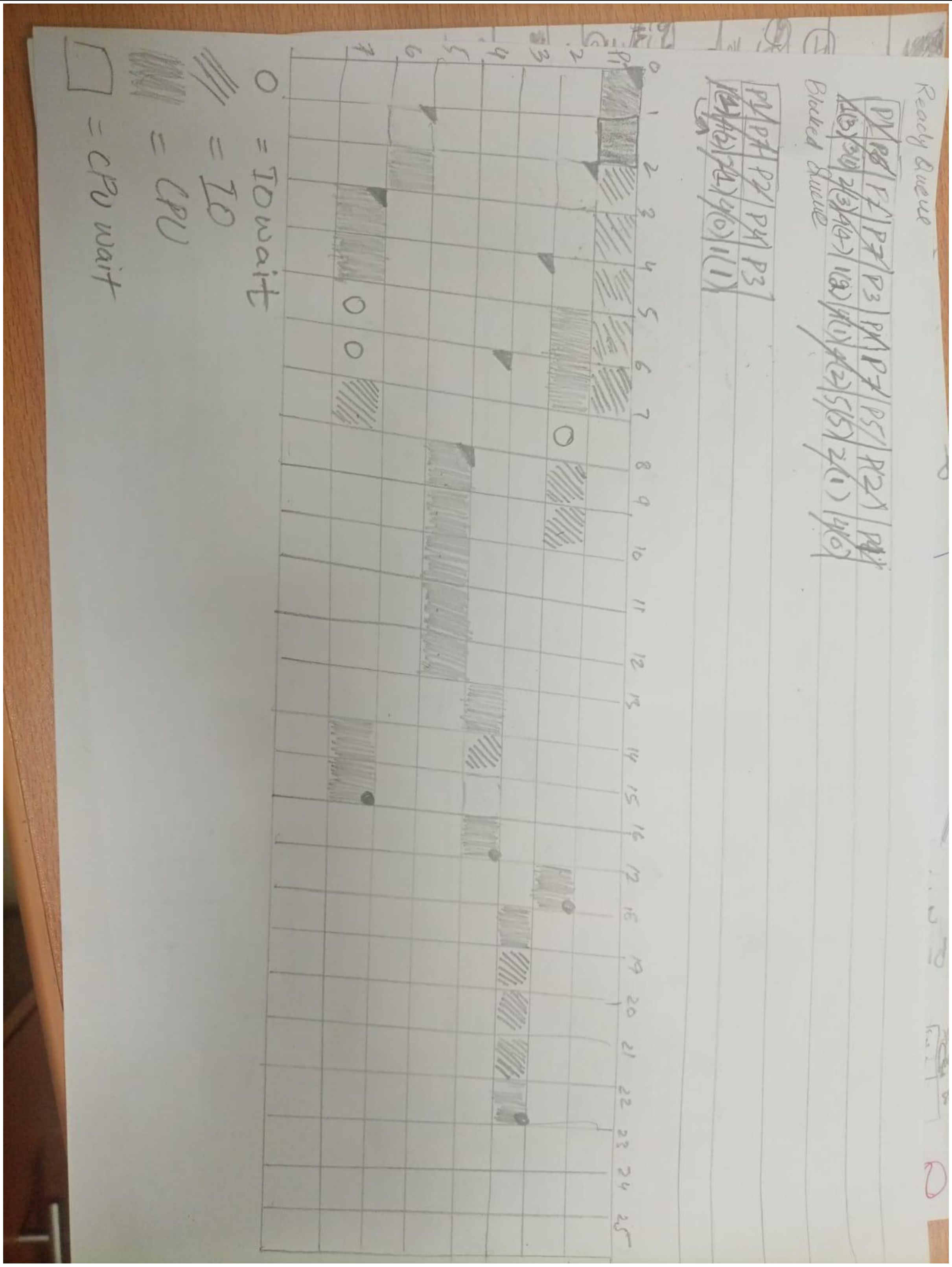
Round Robin (RR) or Priority-based Round Robin (P-RR) might be suitable. RR ensures each process gets an equal share of the CPU, while P-RR adds a priority consideration.

c. Consider the following example, IO Start point (relevant to process) and IO duration is also provided for each process.
For Example: P1 is starting IO 1 unit after getting the CPU and IO takes 5 units of time.
P3 starts IO immediately after it gets the CPU and takes 3 units for IO.

Process	Arrival Time	Initial Priority (0 = lowest)	IO Start - Duration	CPU Burst Time
P1	0	1	1 - 5	3
P2	2	2	1 - 2	3
P3	4	1	0 - 3	2
P4	6	4	1 - 1	1
P5	8	5	0 - 0	5
P6	1	3	0 - 0	1
P7	3	4	1 - 1	4

We know that when a process is doing its IO it is placed in a blocked queue. Suppose there is no more assumption on IO and IO completes in the exact mentioned duration.

Draw the gantt chart for Priority Based Scheduling again, considering the IO. Also, show the ready and blocked queue states. [4]


Ready Queue
Blocked Queue
○ = IO wait
/// = IO
▨ = CPU
□ = CPU wait