Q1 [7 pts each] Consider five jobs A, B, C, D and E with run times of 12, 6, 8, 4 and 7, respectively. Calculate average turnaround time if

A. shortest job first scheduling algorithm is under consideration
B. first-in first-out scheduling algorithm is under consideration

Q2 [10 pts] Fill in the table according to the order the processes are called.

<table>
<thead>
<tr>
<th>Process</th>
<th>mutex</th>
<th>db</th>
<th>rc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader1 arrives into empty database</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writer1 arrives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reader2 arrives while reader1 is in database</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reader3 arrives while reader1 and reader2 are in database</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reader1 leaves the database</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reader2 leaves the database</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writer2 arrives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writer1 continues from where it was stopped</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

typedef int semaphore;                                 //use your imagination
semaphore mutex=1;                                      //controls access to ‘rc’
semaphore db=1;                                         //controls access to the database
int rc=0;                                               //#of processes reading or wanting to

void reader(void)
{
    while(TRUE)                                        //repeat forever
        down(&mutex);                                   //get exclusive access to ‘rc’
        rc=rc+1;                                       //one reader more now
        if(rc==1) down(&db);                            //if this is the first reader …
        up(&mutex);                                     //release exclusive access to ‘rc’
        read_data_base();                               //access the data
        down(&mutex);                                   //get exclusive access to ‘rc’
        rc=rc-1;                                        //one reader fewer now
        if(rc==0) up(&db);                               //if this is the last reader …
        up(&mutex);                                     //release exclusive access to ‘rc’
        use_data_read();                                //noncritical region
    }
}

void writer(void)
{
    while(TRUE)                                        //repeat forever
        think_up_data();                                //noncritical region
        down(&db);                                      //get exclusive access
typedef int semaphore;
semaphore mutex = 1; // controls access to critical region
semaphore empty = 3; // counts empty buffer slots
semaphore full = 0; // counts full buffer slots
void producer(void)
{
    int item;
    while(1) {
        produce_item(&item);
        down(&empty); // decrement empty count
down(&mutex); // enter critical region
enter_item(&item); // put new item in buffer
up(&mutex); // leave critical region
up(&full); // increment count of full slots
    }
}
void consumer(void)
{
    int item;
    while(1) {
        down(&full); // decrement full count
down(&mutex); // enter critical region
remove_item(&item); // take item from buffer
up(&mutex); // leave critical region
up(&empty); // increment count of empty slots
consume_item(item);
    }
}

Q4 [8 pnts] List four conditions that are necessary for multiple processes to cooperate correctly and efficiently using shared data.
Answers.

1. Average turnaround time for shortest job first scheduling algorithm is calculated as follows

\[
\frac{4+10+17+25+37}{5}=\frac{93}{5}=18.6.
\]

The same parameter for first-in first-out scheduling algorithm is found as follows

\[
\frac{12+18+26+30+37}{5}=\frac{123}{5}=24.6.
\]

In this case shortest job first scheduling algorithm is better compared to first-in first-out scheduling algorithm.

2. The values are changed as it is indicated in the table.

<table>
<thead>
<tr>
<th>Process</th>
<th>mutex</th>
<th>db</th>
<th>rc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader1 arrives into empty database</td>
<td>0,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Writer1 arrives</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Reader2 arrives while reader1 is in database</td>
<td>0,1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Reader3 arrives while reader1 and reader2 are in database</td>
<td>0,1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Reader1 leaves the database</td>
<td>0,1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Reader2 leaves the database</td>
<td>0,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Writer2 arrives</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Writer1 continues from where it was stopped</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

3. The completed table is shown below.

<table>
<thead>
<tr>
<th>Process</th>
<th>mutex</th>
<th>empty</th>
<th>full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Producer</td>
<td>0,1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Consumer</td>
<td>0,1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Producer</td>
<td>0,1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Producer</td>
<td>0,1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Producer</td>
<td>0,1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Producer</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Consumer</td>
<td>0,1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

4. The four conditions that are necessary for multiple processes to cooperate correctly and efficiently using shared data are listed below.

1. No two processes may be simultaneously inside their critical sections.
2. No assumptions may be made about speeds or the number of CPUs.
3. No process running outside its critical section may block other processes.
4. No process should have to wait forever to enter its critical section.