Fill your information below before starting to answer the exam questions.

Name	
ID	

Instructions

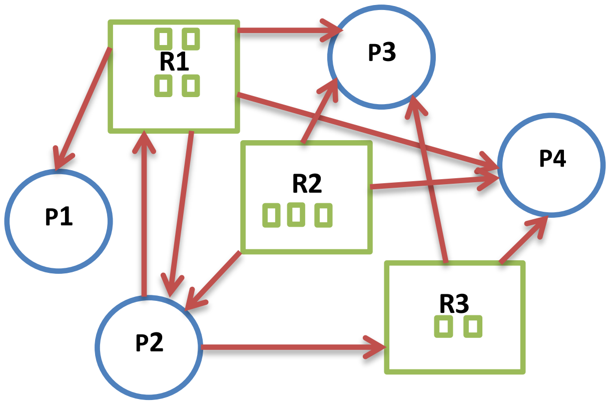
- This is a closed-book exam.
- The exam is 9 pages, consists of 6 questions. Answer all questions.
- **Phones, headsets, and smart glasses, watches, mobiles or other devices are not allowed in this exam.**
- Write your answer in the provided space. **DO NOT** write at the back side of the exam paper.

Grading

Question	Points	Result
Q.1	10	
Q.2	20	
Q.3	15	
Q.4	15	
Q.5	15	
Q.6	10	
Q.7	15	
Total	100	

Q.1 Answer all of the following short-answer questions:

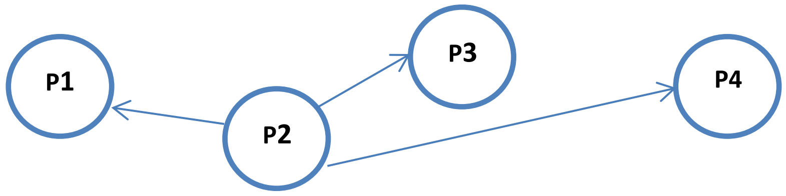
A. Given the following resource allocation graph. [5 POINTS]



I. Find the available vector. [2 POINTS]

<0,0,0>

II. Draw the wait-for graph. [3 POINTS]



B. A process with a memory reference string "3, 9, 3, 5, 9, 6, ,7, 5, 3, 6" is allocated 3 frames in memory. Using the optimal page replacement algorithm, trace the utilization of these frames by this process and find the total number of page faults. [5 POINTS]

Frame	3	9	3	5	9	6	7	5	3	6
1	**3**	**3**		**3**		**3**	**3**			**6**
2		**9**		**9**		**6**	**7**			**7**
3				**5**		**5**	**5**			**5**

Number of page faults = . . 6. . . .

Q.2 Consider a system of **16 GB** of physical memory and a frame size of **16 KB**. Process **P**, in this system, has a logical address space of size **1,200,128 bytes** and page **n** of process **P** is assigned to frame **n+100,000** in memory. Answer the following questions:

Powers of 2	Decimal Byte	Memory Unit	Powers of 2	Decimal Byte	Memory Unit	Powers of 2	Decimal Byte	Memory Unit
2^{0}	1	1B	2^{12}	4,096	4KB	2^{24}	16,777,216	
2^{1}	2		2^{13}	8,192		2^{25}	33,554,432	
2^{2}	4		**2^{14}**	**16,384**	**16KB**	2^{26}	67,108,864	
2^{3}	8		2^{15}	32,768		2^{27}	134,217,728	
2^{4}	16		2^{16}	65,536		2^{28}	268,435,456	
2^{5}	32		**2^{17}**	**131,072**		2^{29}	536,870,912	
2^{6}	64		2^{18}	262,144		**2^{30}**	**1,073,741,824**	**1GB=1024^{6}**
2^{7}	128		2^{19}	524,288		2^{31}	2,147,483,648	
2^{8}	256		**2^{20}**	**1,048,576**	**1MB=1024^{3}**	2^{32}	4,294,967,296	
2^{9}	512		2^{21}	2,097,152		2^{33}	8,589,934,592	
2^{10}	**1024**	**1KB**	2^{22}	4,194,304		**2^{34}**	**17,179,869,184**	**16GB**
2^{11}	2,048		2^{23}	8,388,608		2^{35}	34,359,738,368	

**Note that page numbering starts from 0.*

A. How many bits are needed to represent a frame number in this system? [4 POINTS]

34 – 14 = 20 Bits

B. What is the physical address of the logical address 85,000? Show the steps of your work. [12 POINTS]

Frame # = 85,000 / 16,384 = 5.18798828125 = 5
Offset = 85,000 % 16,384 = 3080
VLA = (5, 3080)
VPA = (100,005, 3080)
PA = 100,005 * 16,384 + 3080 = 1,638,485,000

C. What is the amount of internal fragmentation of the process P? Show the steps of your work.[4 POINTS]

I.F. = 16,384 – (1,200,128 % 16,384) = 16,384 – 4,096 = 12,288

Q.3 Answer the following questions given the following segmentation table of a process.

Segment number	Base Address	Limit
0	1,638,400,000	376,832
1	1,640,038,400	163,840
2	1,641,676,800	1,146,880
3	1,643,315,200	540,672

A. What is the physical address of the logical address 820,000? Show your work. [7 POINTS]

VLA = (2, 820,000 – (376,832 + 163,840))

= (2, 820,000 – 540,672)

= (2, 279,328)

PA = 1,641,676,800 + 279,328 = 1,641,956,128

B. What is the size of the logical address space of this process. [4 POINTS]

Size of LAS = 376,832 + 163,840 + 1,146,880 + 540,672 = **2,228,224**

C. What is the virtual logical address of the physical address 1,640,200,000? Show your work. [4 POINTS]

VLA = (1, (1,640,200,000 - 1,640,038,400))
= (1, 161600)

Q. 4 Consider the following safe state snapshot of a system with 9 identical resources of each type A, B, C, and D. [15 POINTS]

Available

A	B	C	D
3	**3**	**3**	**3**

Needed				
	A	B	C	D
P0	8	4	0	1
P1	5 **2**	5 **2**	2 **1**	2 **1**
P2	5	3	4	5
P3	0	0	2	1
P4	4	5	3	6

Current Allocation				
	A	B	C	D
P0	1	1	0	2
P1	1 **4**	2 **5**	0 **1**	0 **1**
P2	1	0	2	1
P3	2	3	2	2
P4	1	0	2	1
	6	**6**	**6**	**6**

Will the request **(3, 3, 1, 1)** of process **P1** be granted? Show your work.

Test 1:

<3,3,1,1> <= <5,5,2,2> pass

Test2:

<3,3,1,1> <= <3,3,3,3> pass

Test3:

	Available			
	A	B	C	D
	0	**0**	**2**	**2**
P3	**2**	**3**	**4**	**4**
P1	**6**	**8**	**5**	**5**
P2	**7**	**8**	**7**	**6**
P4	**8**	**8**	**9**	**7**
P0	**9**	**9**	**9**	**9**

Execution sequence is:
<P3,P1,P2,P4,P0>

Therefore, **new safe state and request is granted.**

Q.5 Using tabular format or Gantt chart, trace the execution of the following processes using preemptive priority CPU scheduling algorithm then complete the table calculating the chosen performance measurement of interest. [15 POINTS]

Processes

PID	Arrival Time	Burst Time in milliseconds	Priority
1	2	7	15
2	7	4	9
3	9	5	3
4	12	6	7

Performance Measures

PID	Waiting Time
1	**15**
2	**11**
3	**0**
4	**2**
Sum	**28**
Avr.	**7**

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
PPr		**P1**					**P2**		**P3**					**P4**						**P2**		**P1**			

Time	Event	Ready Queue	CPU	Finished
2	**A**		**P1(7,15)**	
7	**A**	**P1(2,15)**	**P2(4,9)**	
9	**A**	**P1(2,15) P2(2,9)**	**P3(5,3)**	
12	**A**	**P1(2,15) P2(2,9)P4(6,7)**	**P3(2,3)**	
14	**D**	**P1(2,15) P2(2,9)**	**P4(6,7)**	**P3**
20	**D**	**P1(2,15)**	**P2(2,9)**	**P4P3**
22	**D**		**P1(2,15)**	**P2P4P3**
24	**D**			**P1P2P4P3**

Q.6 Write the server side class **VotingServer** given the client side code **VotingClient** below. You do not have to write the import statements or the try/catch related statements. The server on receiving a vote for one of the 10 items from the client it increments the corresponding total votes of that item and writes to the client the results of each total of the 10 items' votes all in one formatted message.

```
package voting;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class VotingClient {
   private DatagramSocket client;
   private Scanner kb;

   public VotingClient() {
      try {
         client = new DatagramSocket();
         kb = new Scanner(System.in);
         System.out.print("Your vote [1 ... 10]: ");
         String message = new String(Integer.toString(kb.nextInt()));

         byte[] data = message.getBytes();
         DatagramPacket packet = new DatagramPacket(data,
                                            data.length,
                         InetAddress.getLocalHost(), 5000);
         client.send(packet);
         byte[] receivedData = new byte[1000];
         DatagramPacket receivedPacket = new DatagramPacket(
                                            receivedData, receivedData.length);
         client.receive(receivedPacket);
         System.out.printf("Up to this time the voting results are\n%s\n",
            new String(receivedPacket.getData(),0,
            receivedPacket.getLength()));
      } catch (IOException e) {
         System.exit(1);
      }
   }
   public static void main(String[] args){
      new VotingClient();
   }
}
```

```
package voting;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

public class VotingServer {
    DatagramSocket server;
    long[] votes = new long[10];

    public VotingServer() {
        try {
            server = new DatagramSocket(5000);
        } catch (SocketException socketException) {
            System.exit(1);
        }
        while (true) {
            try {
                byte[] data = new byte[100];
                DatagramPacket packet = new DatagramPacket(data, data.length);
                server.receive(packet);

                int index = Integer.parseInt((new String(data, 0,
data.length)).trim());
                votes[index]++;

                String response = new String("");
                for(int i=0; i<votes.length;i++)
                    response = Integer.toString(i)+":
"+Long.toString(votes[i])+"\n";

                byte[] r = response.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(r, r.length,
packet.getAddress(), packet.getPort());
                server.send(sendPacket);
            } catch (Exception e) {
            }
        }
    }

    public static void main(String[] args) {
        new VotingServer();
    }
}
```

Q.7 An old bridge has only one lane and can only hold at most 3 cars at a time without risking collapse. Code the class Monitor with methods arriveAtBridge(int direction) and exitBridge() that controls traffic so that at any given time, there are at most 3 cars on the bridge, and all of them are going to the same direction.

A car calls arriveAtBridge when it arrives at the bridge and wants to go in the specified direction (0 or 1); arriveAtBridge should not return until the car is allowed to get on the bridge. A car calls exitBridge when it gets off the bridge, potentially allowing other cars to get on.

Don't worry about starving cars trying to go in one direction; just make sure cars are always on the bridge when they can be. A car is modeled by a multithreaded process that repeatedly enters and leaves the bridge. [15 POINTS]

```java
package oldbridge;

import java.util.concurrent.Semaphore;

public class Monitor {
    Semaphore lock = new Semaphore(1);
    Semaphore turn = new Semaphore(1);
    int[] waiters = {0,0};
    Semaphore brigeCarsSpace = new Semaphore(3);
    int currentdirection = 0;
    public Monitor() {

    }

    public void arriveAtBridge(int direction) {
        try {
            lock.acquire();
            waiters[direction]++;
            // while can't get on the bridge, wait
            if(brigeCarsSpace.availablePermits() == 3){
                turn.acquire();
                currentdirection = direction;
                waiters[direction]--;
            }
            else{
                turn.acquire();

            }

            while(currentdirection != direction)
                waiters[direction]++;
            if (currentdirection == direction &&
                    brigeCarsSpace.availablePermits() < 1)
                waiters[direction]++;
            //try to get on the bridge
            if(currentdirection == direction &&
                    brigeCarsSpace.availablePermits() > 0){
                brigeCarsSpace.acquire();
                waiters[direction]--;
            }
            lock.release();

            lock.acquire();
            if(waiters[direction]==0 && brigeCarsSpace.availablePermits() ==
0){
                turn.release();
                currentdirection = 1-direction;
            }
            lock.release();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public void exitBridge() {
          lock.acquire();

          // get off the bridge
          carsCount--;

          // if anybody wants to go the same direction, wake them
          if (waiters[currentdirection] > 0)
             waitingToGo[currentdirection].signal();
          // else if empty, try to wake somebody going the other way
          else if (carsCount == 0)
             waitingToGo[1-currentdirection].broadcast();

          lock.release();
        }
}
```