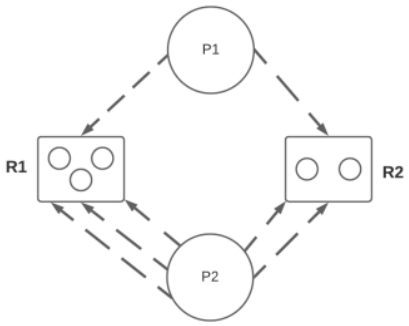
Midterm

Name : Punyaja Mishra Student Id : 0660001

Q1.



a. Show a sequence of operations leading from the given state to a deadlocked state. (1 point)

According to the given resource claim graph,

- ⇨ p1 is requesting for 1 unit of resource R1 and 1 unit of resource R2
- ⇨ p2 is requesting for 3 unites of resource R1 and 2 unites of resource R2
- ⇨ If, 1 unit of R2 is given to p1, and 3 units of R1 is given to p2, then the state will enter a deadlock state.

b. Show how the Banker's Algorithm would have prevented the deadlock.

Using Banker's algorithm : Have a temporary state s' where we grant the request by changing edges to allocation edges. Execute state s on the new state s', if it is reducible then accept s' as the new state else revert back to state s

For instance,

- ⇨ The given state is s.
- ⇨ Create a new state s' where p1 is granted R2 1 unit and p2 is granted R1 3 units.
- ⇨ Execute this s'
- ⇨ It will not be reducible
- ⇨ Revert back to state s
- ⇨ Keep going unless we find a reducible state

This way, we are not exactly converting into a new state. The temporary state acts like a loan which helps us determine if it will or will not be the right state to choose. Hence, preventing deadlock.

Q2.

Available Resources			
A	B	C	D
4	3	5	3

Current Allocation				
Process/Resource	A	B	C	D
P0	4	0	3	1
P1	0	2	1	5
P2	4	1	0	3
P3	1	0	0	4
P4	1	1	0	0
P5	1	0	1	1
Maximum Demand				
Process/Resource	A	B	C	D
P0	6	5	4	3
P1	2	2	3	8
P2	7	5	4	4
P3	3	4	2	8
P4	4	2	2	0
P5	4	4	2	3

a. **Verify whether or not the available array was calculated correctly.**

To determine if available array was calculated correctly. For each array, we do this calculation :

(I am doing only for process A because jus 0.5 marks 😊)

All the processes currently allocated to A : p0=4, p1 = 0, p2 = 4, p3 = 1, p4=1,p5=1

Available resources for A = 4

Total Available – processes allocated = 15 – 11 = 4, which is equal to the available resources on A

Therefore, array for A is calculated correctly

b. **Calculate the need matrix. (0.5 point)**

Need Matrix = Maximum Matrix – Allocation Matrix

A	B	C	D
4	0	3	1
0	2	1	5
4	1	0	3
1	0	0	4
1	1	0	0
1	0	1	1

-

A	B	C	D
6	5	4	3
2	2	3	8
7	5	4	4
3	4	2	8
4	2	2	0
4	4	2	3

=

A	B	C	D
2	5	1	2
2	0	2	3
3	4	4	1
2	4	2	4
3	1	2	0
3	4	1	2

c. **Show that the current state is safe (show a safe sequence of processes). (2 points)**

To know if current state is safe, we do it by the algorithm :

If(need<=available) then run

else don't run and move to next process

Let's do it for the processes for A

P0 need = 2 ; available = 4 : 2<4 : hence run

P1 need = 2 ; available = 4 : 2<4 : hence run

P2 need = 3 ; available = 4 : 3<4 : hence run

P3 need = 2 ; available = 4 : 2<4 : hence run

P4 need = 3 ; available = 4 : 3<4 : hence run

P5 need = 3 ; available = 4 : 3<4 : hence run

So running processes 0 to 5 for A would be a safe sequence of processes

d. Say a request (10,7,9,5) arrives from process 5, should this request be granted? Why? (1 point)

This request should not be granted because the new allocation matrix, new need matrix and the available resources will be the following : (the allocation is very much more than the available and it will not run since it is not a safe sequence)

New allocation matrix :

A	B	C	D
4	0	3	1
0	2	1	5
4	1	0	3
1	0	0	4
1	1	0	0
11	7	10	6

New need matrix :

A	B	C	D
2	5	1	2
2	0	2	3
3	4	4	1
2	4	2	4
3	1	2	0
-7	-3	-8	-3

Available resources :

A	B	C	D
-6	-4	-4	-2

Q3. Imagine a system that has n processes and a resource with m identical units. The maximum claim of any process is k.
Why is the system deadlock free for all possible states of m > n (k-1)? (1 point)

m – identical units of resource

k-1 = maximum claim -1

m>n(k-1)

this is because number of resource units is greater than each process permuting with one less than the claim they can do. In simple words, let's say 4 processes and each can claim 3 units. This increased number of resources than the possible permutation of claims, avoids any kinds of deadlocks. This is because the number of current allocation edges and claim edges will never be clashing – meaning that there will be a possibility of having a free resource and hence avoid deadlock. At least one way of reducing the system will be possible.

Q4. Imagine a system that has 4 processes and a resource with 4 identical units, only two processes can claim resource units at the same time, and the maximum claim each process can make is 2 units. Is this system a deadlock free system? Why or why not? (1 point)

We do this by using the formula m>n(k-1) , where

m=number of identical units in the resource

n=number of resource

k=maximum claim possible

Number of processes = 4

Number of resources identical units = 4

Maximum claim possible = 2

Number of processes * (maximum claim -1) = R (just a variable)

= 4 *(2-1)

= 4*1

= 4

Number of resources identical units = 4 which is equal to the R

For the system to be deadlock free, it should be greater than R. This means there is still a possibility of a deadlock.

Q5- Given the six-state process activity model discussed in class (New, Ready, Running, Blocked, Suspended, Terminated), which of the following series of transitions are possible (assume you entered into a starting state from a valid series of transitions). For each series of transitions that is not possible explain why. (2.5 points)

a. **New→Ready→Running→Ready→Suspended→Ready→Terminated**

Not valid. A process cannot be terminated unless running. That means it can not go from ready state to terminated.

b. **Ready→Running→Suspended→Blocked→Ready→Running**

Not valid. A process can't go from suspended to blocked if it got suspended from running. It moves to ready.

c. **Running→Blocked→Suspended→Ready→Running→Terminated**

Not valid. A process if suspended in blocked, must go back in blocked state and it can't go to ready state.

d. **Blocked→Ready→Running→Blocked→Suspended→Running→Terminated**

Not valid. A process can't go from suspended to running. Only from ready to running.

e. **Suspended→Blocked→Ready→Suspended→Terminated**

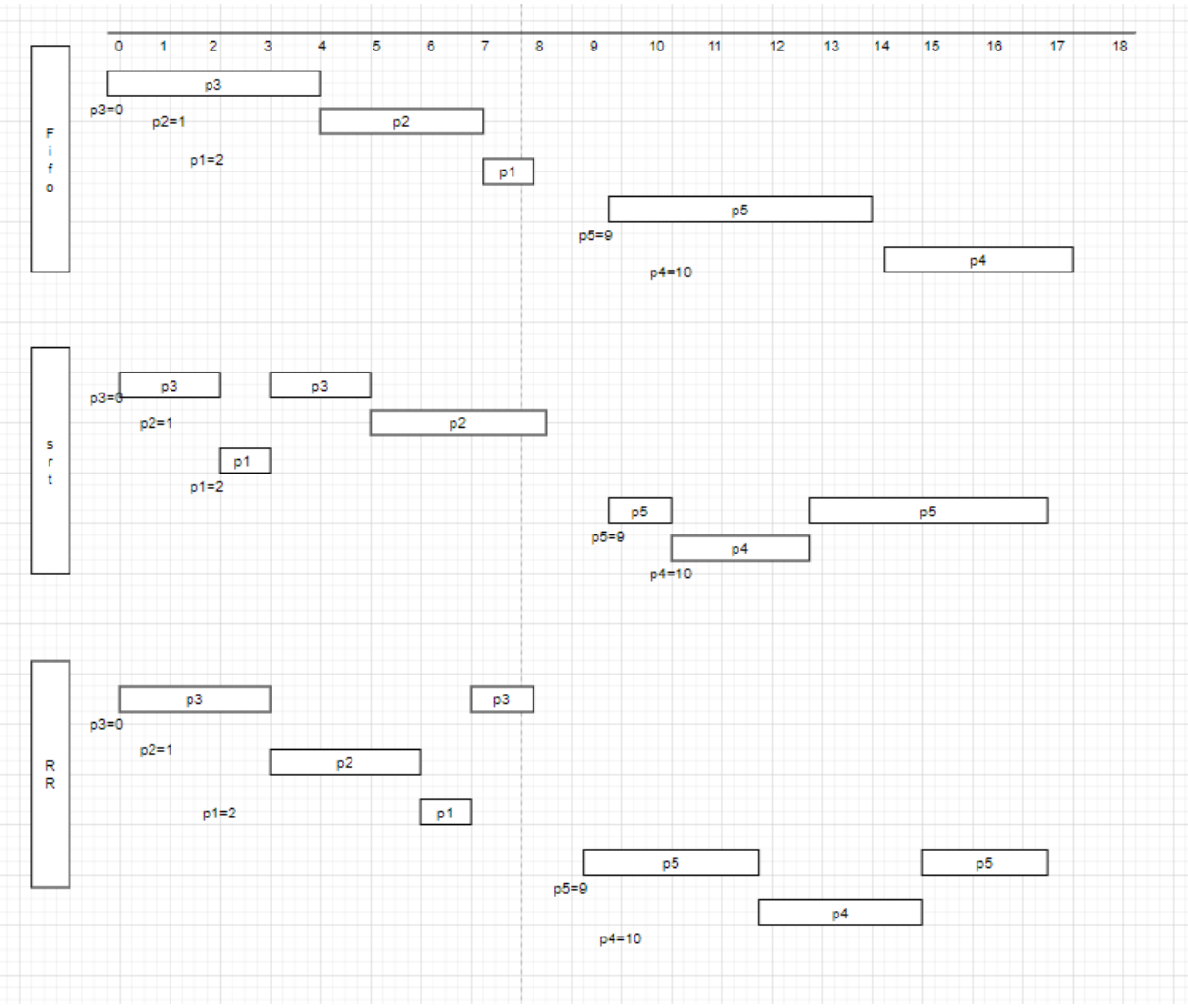
Not valid. A process can't go from suspended to terminated. Only running process can be terminated.

Q6.

Process	Total CPU Time	Arrival Time
P1	1	2
P2	3	1
P3	4	0
P4	3	10
P5	5	9

a. **Complete the following Gantt chart illustrating the execution of these processes using FCFS, SRT, and RR (q=3). All the processes for each scheduling algorithm should be on the same chart. (3 points)**

The chart includes the points when the processes arrive as well



b. What is the Average Turnaround Time (ATT) for each of the scheduling algorithms in (a)? (3 points)

For each process -> waiting time + running time

ATT -> add each process/number of process (5)

ATT for FCFS :

P3 = 0 + 4 = 4

P2 = 3 + 3 = 6

P1 = 5 + 1 = 6

P5 = 0 + 5 = 5

P4 = 4 + 3 = 7

ATT = (4+6+6+5+7)/5 = 5.6

ATT for SRT :

P3 = 1 + 4 = 5

P2 = 4 + 3 = 7

P1 = 0 + 1 = 1

P5 = 3 + 5 = 8

P4 = 0 + 3 = 3

ATT = (5+7+1+8+3)/5 = 4.8

ATT for RR :

P3 = 4 + 4 = 8

P2 = 2 + 3 = 5

P1 = 4 + 1 = 5

P5 = 3 + 5 = 8

P4 = 2 + 3 = 5

ATT = (8+5+5+8+5)/5 = 6.2

Q7. C code question

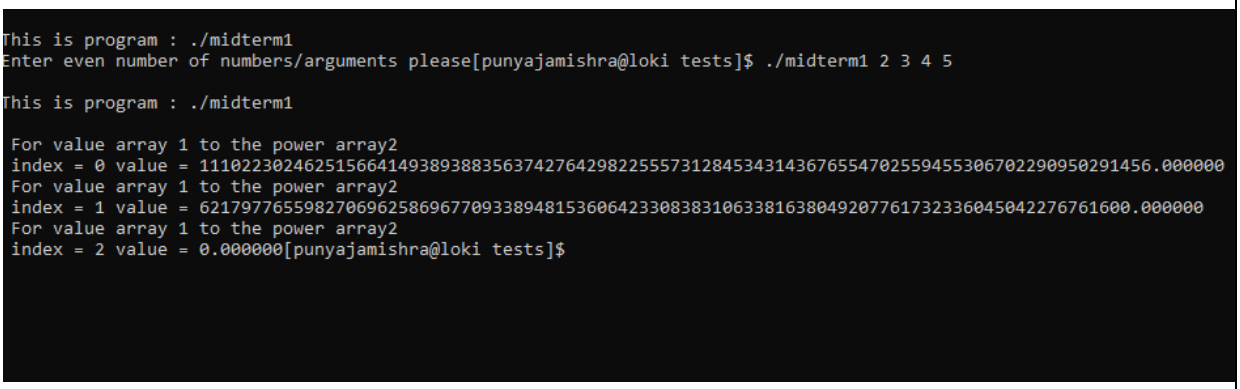
Test 1 : No argument Passed

```
unyajamishra@loki tests]$ ./midterm1

is is program : ./midterm1
 argument passed[punyajamishra@loki tests]$
```

Test 2 : Odd argument Passed

```
This is program : ./midterm1
No argument passed[punyajamishra@loki tests]$ ./midterm1 2 3 4

This is program : ./midterm1
Enter even number of numbers/arguments please[punyajamishra@loki
```

Test 3 : Even argument Passed



Code :

```
/*******************************************************************************

Welcome to GDB Online.
GDB online is an online compiler and debugger tool for C, C++, Python, PHP, Ruby,
C#, VB, Perl, Swift, Prolog, Javascript, Pascal, HTML, CSS, JS
Code, Compile, Run and Debug online from anywhere in world.

*******************************************************************************/
#include <stdio.h>
#include<stdlib.h>
#include<math.h>

int count =0;
int i=1;
int j=0;
int main(int argc, char *arguments[])
{
```

```
printf("\nThis is program : %s\n", arguments[0]);

//no argument passed
if(argc < 2){
   printf("No argument passed");
}
//odd number of arguments
else if((argc+1)%2!=0){
   printf("Enter even number of numbers/arguments please");
}
//all of that is fine
else{
   //initialize arrays
   int arraylength = (argc+1)/2;
   int a1[arraylength]; //array 1
   int a2[arraylength];  //array 2

   //fill in the arrays
   for(i; i< arraylength ; i++){
      a1[i-1] = (int)(*arguments[i]);
      a2[i-1] =(int)(*arguments[argc-i]);
   }

   //now printing the calculations of exponents
   for(j; j<arraylength; j++){
      double value = pow(a1[j], a2[j]);
      printf("\n For value array 1 to the power array2");
      printf("\n index = %d",j);
      printf(" value = %f", value);
```

```
        }


    }



    return 0;
}
```