MUHAMMAD UMAR KHAN 10619
OS THEORY FINAL EXAM CID: 106342

Q1:

Given 4 memory partition of 600k, 619k, 250k, 500k Now we have to allocate the process of size 128k, 581k, 411k, 221k respectively

First Fit:-

Memory Size	Memory Partition
128	600 k
581	619 k
221	250 k
411	500 k

Best Fit:-

Memory Size	Memory Partition
581	600 k
221	619 k
128	250 k
411	500 k

Worst Fit:-

Memory Size	Memory Partition
581	600 k
128	619 k
221	250 k
411	500 k

MUHAMMAD UMAR KHAN 10619
OS THEORY FINAL EXAM CID: 106342

Observations:

best fit is better than first fit

first fit is better than worst fit

best fit > first fit > worst fit

Q4:

Data:

Size Of Physical Memory: 2^{39}

Size Of Pages: 2^{11}

Number Of Pages in logical address space: 2^{23}

Formula:

Number of bytes in logical address = Pages in logical address x Page Size

(a)

=> $2^{23} \times 2^{11}$

=> 2^{34} bytes **Ans.**

Hence, the number of bits in logical address is 34.

(b)

The size of the frame is same as the size of the pages: 2^{11} bytes **Ans.**

- The memory in the physical memory is divided into fixed sized blocks called pages frames
- Each process in the logical memory is divided into fixed size chunks called pages
- The pages are assigned to available fixed sized blocks called page frames.

(c)

The number of frames in physical memory is calculated as follows:

$$\text{Number of frames} = \frac{\text{Size Of Physical Memory}}{\text{Size Of Page Frame}}$$

$$\text{Number Of Frames} = \frac{2^{39}}{2^{11}}$$

Number Of Frames = 2^{28}

Therefore, the number of bits required to specify the frame location in physical memory is 28. Ans

The number of entries in the page table is the number of pages in the virtual memory therefore the number of entries in the page table is 2^{23} **Ans.**

Q3:

3(A)

A single threaded process can only run on one CPU, whereas the execution of a multi-threaded application may be split amongst available processors. single kernel thread can operate only on a single CPU, the many-to-one model does not allow individual processes to be split across multiple CPUs.

3(B)

Thread switching is a type of context switching from one thread to another thread in the same process. Thread switching is very efficient and much cheaper because it involves switching out only identities and resources such as the program counter, registers and stack pointers.

3(C)

observation:

1-Offloading a batch of work to a worker thread so your program can either continue responding to user input or so you can carry on running other code that doesn't rely on that work.

2- Handling INPUT/OUTPUT particularly server and network communication where the response time is variable or unknown.

3- Parallel processing of data where you can sub-divide the work down into discrete non-dependent units of work

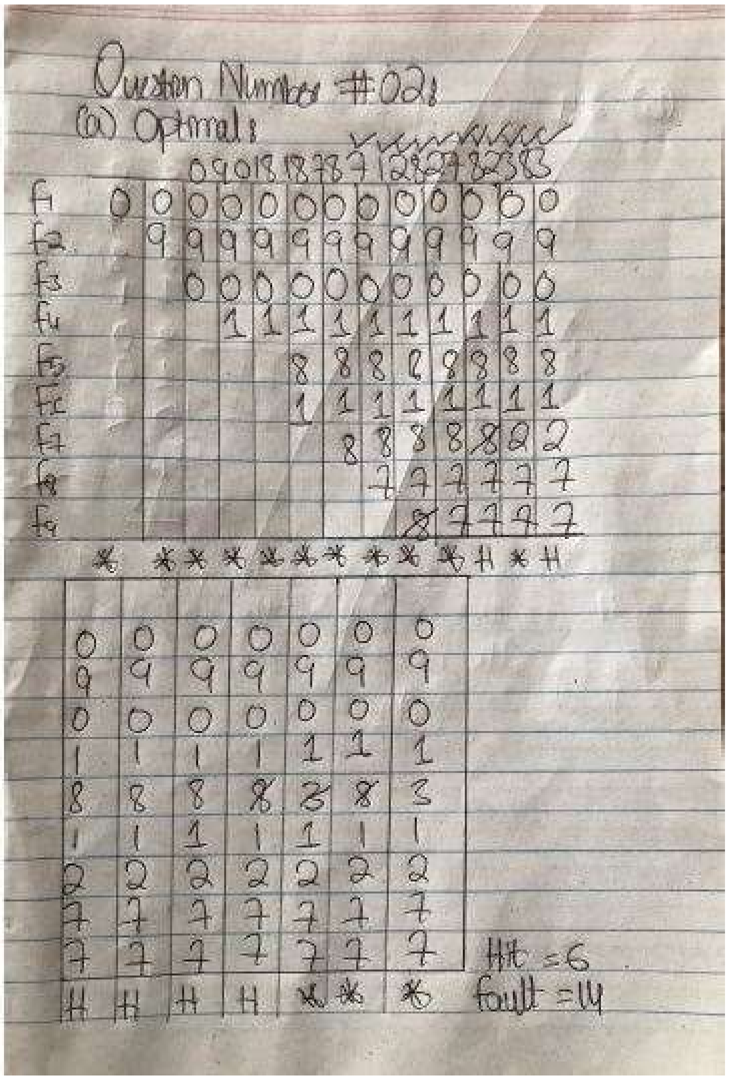
4. Timer related work that is very 500ms check if x has changed

However, switching to multi-threaded or concurrent programming is not without it's

MUHAMMAD UMAR KHAN 10619
OS THEORY FINAL EXAM CID: 106342

pitfalls particularly if those threads need to access shared data.

Q2:



Question Number #02:

(a) Optimal:

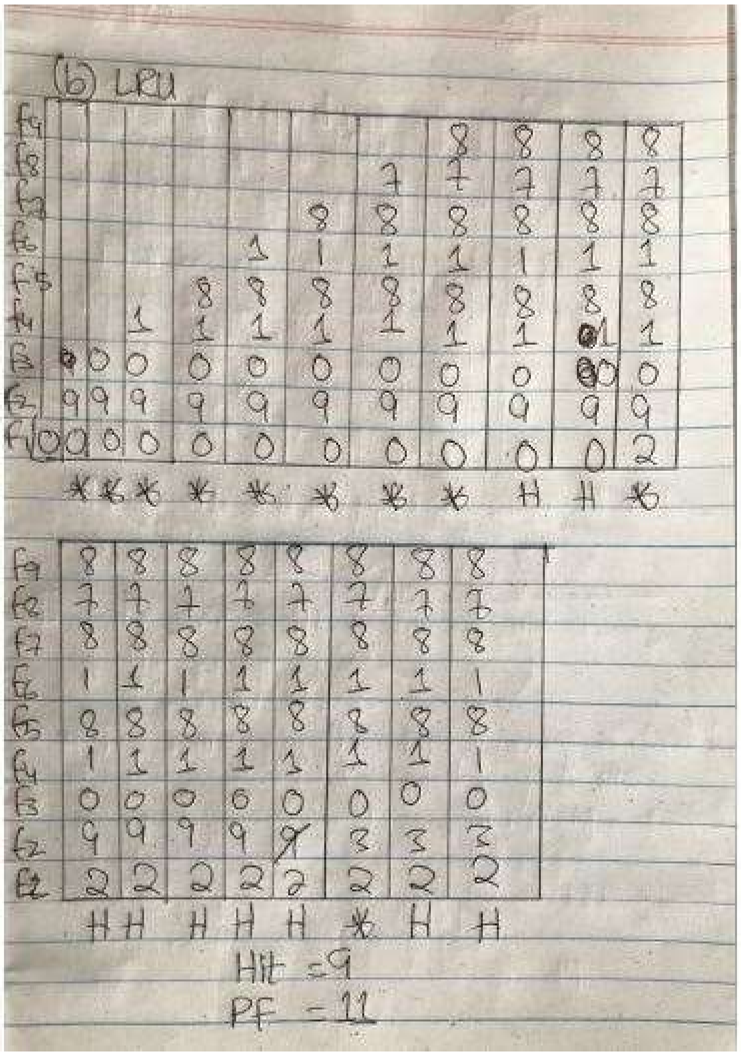
0901818787128278238 3

f1	0	0	0	0	0	0	0	0	0	0	0	0	0
f2		9	9	9	9	9	9	9	9	9	9	9	9
f3			0	0	0	0	0	0	0	0	0	0	0
f4				1	1	1	1	1	1	1	1	1	1
f5						8	8	8	8	8	8	8	8
f6						1	1	1	1	1	1	1	1
f7							8	8	8	8	8	2	2
f8								7	7	7	7	7	7
f9									~~8~~	7	7	7	7
	*		*	*	*	*	*	*	*	*	H	*	H

0	0	0	0	0	0	0
9	9	9	9	9	9	9
0	0	0	0	0	0	0
1	1	1	1	1	1	1
8	8	8	8	8	8	3
1	1	1	1	1	1	1
2	2	2	2	2	2	2
7	7	7	7	7	7	7
7	7	7	7	7	7	7
H	H	H	H	*	*	*

Hit = 6
fault = 14

MUHAMMAD UMAR KHAN 10619
OS THEORY FINAL EXAM CID: 106342



(b) LRU

f9								8	8	8	8
f8							7	7	7	7	7
f7						8	8	8	8	8	8
f6					1	1	1	1	1	1	1
f5				8	8	8	8	8	8	8	8
f4			1	1	1	1	1	1	1	1	1
f3	0	0	0	0	0	0	0	0	0	0	0
f2	9	9	9	9	9	9	9	9	9	9	9
f1	0	0	0	0	0	0	0	0	0	0	2
	*	*	*	*	*	*	*	*	H	H	*

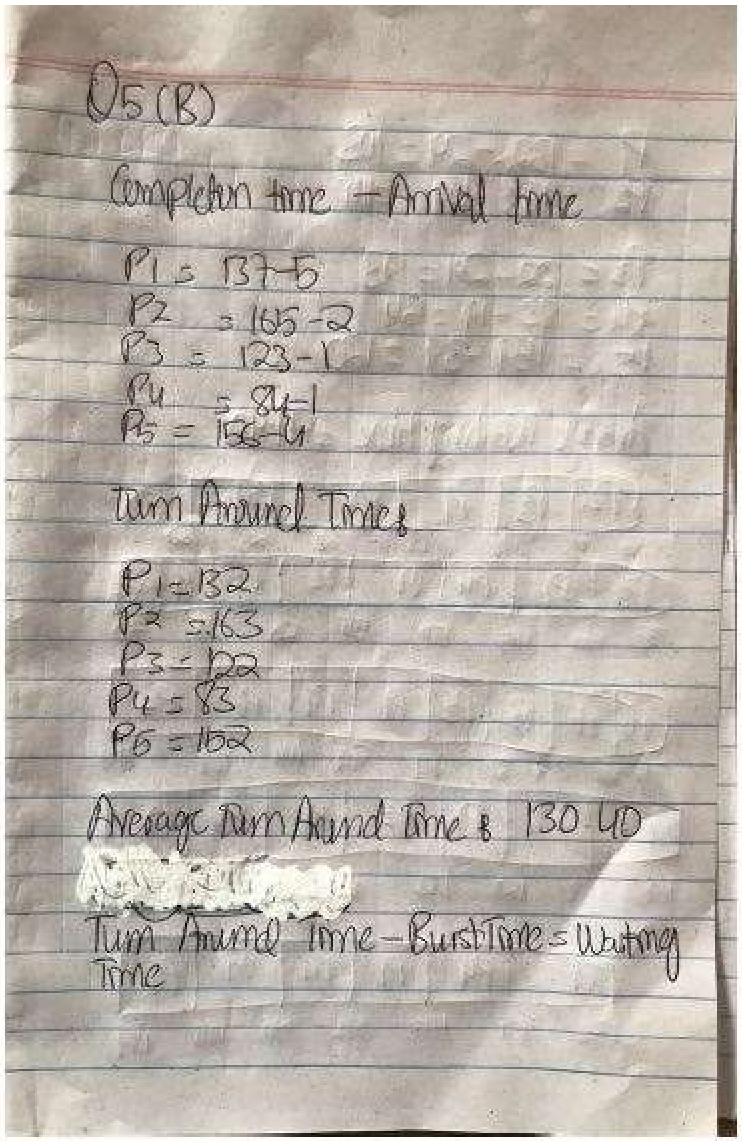
f9	8	8	8	8	8	8	8	8
f8	7	7	7	7	7	7	7	7
f7	8	8	8	8	8	8	8	8
f6	1	1	1	1	1	1	1	1
f5	8	8	8	8	8	8	8	8
f4	1	1	1	1	1	1	1	1
f3	0	0	0	0	0	0	0	0
f2	9	9	9	9	~~9~~	3	3	3
f1	2	2	2	2	2	2	2	2
	H	H	H	H	H	*	H	H

Hit = 9
PF = 11

MUHAMMAD UMAR KHAN 10619
OS THEORY FINAL EXAM CID: 106342

Q5:

(a)



Q5 (B)

Completion time - Arrival time

P1 = 137-5
P2 = 165-2
P3 = 123-1
P4 = 84-1
P5 = 156-4

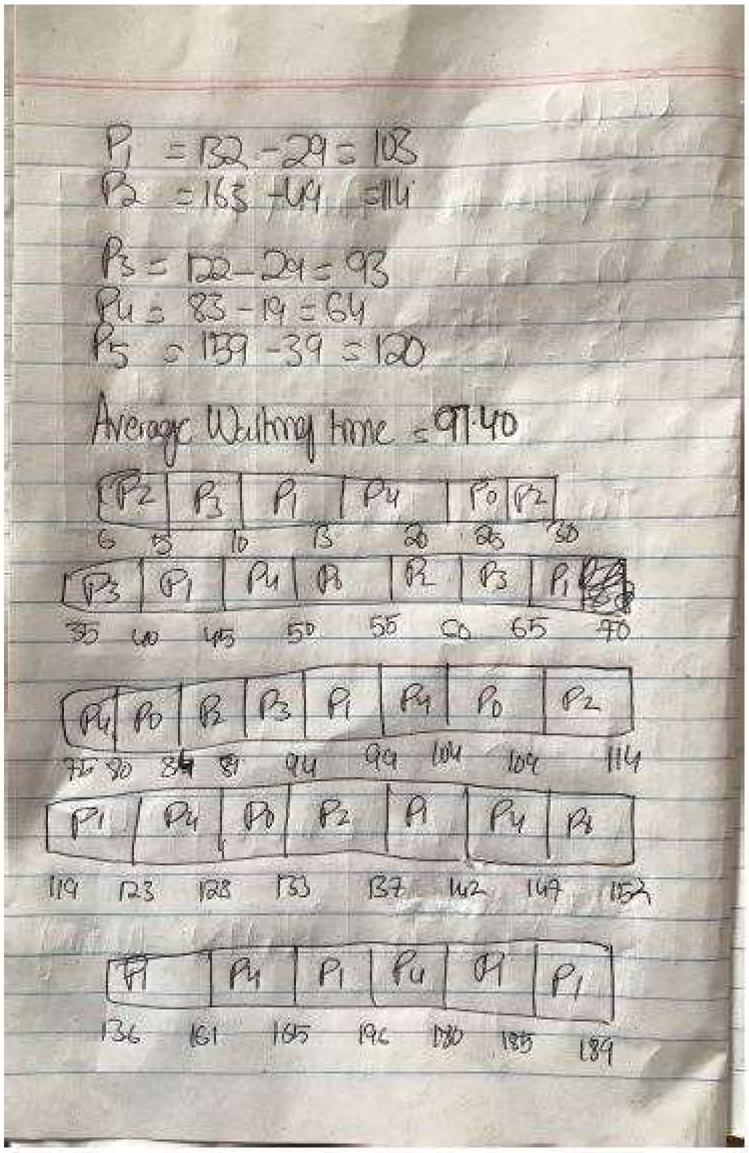
Turn Around Time:

P1 = 132
P2 = 163
P3 = 122
P4 = 83
P6 = 152

Average Turn Around Time: 130 40

Turn Around Time - Burst Time = Waiting Time

MUHAMMAD UMAR KHAN 10619
OS THEORY FINAL EXAM CID: 106342



P1 = 132 − 29 = 103
P2 = 163 + 49 = 114

P3 = 122 − 29 = 93
P4 = 83 − 19 = 64
P5 = 139 − 39 = 120

Average Waiting time = 97.40

P2	P3	P1	P4	P0	P2	
0	5	10	13	20	25	30

P3	P1	P4	P0	P2	P3	P1	
35	40	45	50	55	60	65	70

P4	P0	P2	P3	P1	P4	P0	P2	
72	80	84	89	94	99	104	109	114

P1	P4	P0	P2	P1	P4	P1	
119	123	128	133	137	142	147	152

P1	P4	P1	P4	P1	P1	
156	161	165	176	180	185	189

(b)

```
#include<stdio.h>
void main()
{
 int n,i,j,TEMP,TEMP1,TEMP2,TEMP3,TEMP4;
 float WTSUM=0,TATSUM=0;
 int bt[10],at[10],P[10],ct[10],tat[10],wt[10],pt[10];
 printf("enter the no of  process\n");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  printf("Enter the arrival time of p%d:\t",i);
  scanf("%d",&at[i]);
  printf("Enter burst time of p%d: \t",i);
  scanf("%d",&bt[i]);
  printf("Enter the priority :\t");
  scanf("%d",&pt[i]);
  P[i]=i;
  printf("\n");
 }
 ct[0]=0;
    if(bt[1]>=at[n])
    {
        ct[1]=bt[1]+at[1];
        tat[1]=ct[1]-at[1];
        wt[1]=tat[1]-bt[1];
  WTSUM=wt[1];
  TATSUM=tat[1];
        for(i=2;i<=n;i++)
        {
```

```
        for(j=i+1;j<=n;j++)

        {

           if(pt[j]<pt[i])

           {

              TEMP4=pt[i];

              pt[i]=pt[j];

              pt[j]=TEMP4;

              TEMP1=bt[i];

              bt[i]=bt[j];

              bt[j]=TEMP1;

              TEMP2=at[i];

              at[i]=at[j];

              at[j]=TEMP2;

              TEMP3=P[i];

              P[i]=P[j];

              P[j]=TEMP3;

           }

        }

           if(ct[i-1]<at[i])

           {

              TEMP=at[i]-ct[i-1];

              ct[i]=ct[i-1]+bt[i]+TEMP;

              TEMP1=bt[i];

           }

           else

           {

              ct[i]=ct[i-1]+bt[i];

           }

           tat[i]=ct[i]-at[i];

           wt[i]=tat[i]-bt[i];

           WTSUM=WTSUM+wt[i]+wt[1];

           TATSUM=TATSUM+tat[i]+tat[1];

    }
```

```
 }
      if(at[n]==0)
      {
      ct[0]=0;
      for(i=1;i<=n;i++)
        {
          for(j=i+1;j<=n;j++)
          {
            if(pt[j]<pt[i])
            {
              TEMP4=pt[i];
              pt[i]=pt[j];
              pt[j]=TEMP4;
             TEMP1=bt[i];
             bt[i]=bt[j];
             bt[j]=TEMP1;
            TEMP2=at[i];
            at[i]=at[j];
            at[j]=TEMP2;
            }
          }
        if(ct[i-1]<at[i])
        {


          TEMP=at[i]-ct[i-1];
          ct[i]=ct[i-1]+bt[i]+TEMP;
          TEMP1=bt[i];
        }
        else
        {
          ct[i]=ct[i-1]+bt[i];
        }
      tat[i]=ct[i]-at[i];
```

```
          wt[i]=tat[i]-bt[i];

          WTSUM=WTSUM+wt[i];

         TATSUM=TATSUM+tat[i];

      }

   }

 printf("\n\n\n");

 printf("\tPROCESS\tBT\tAT\tPT\tTATshyam\tWT\n");

 for(i=1;i<=n;i++)

 {

 printf("\tP%d\t%d\t%d\t%d  \t%d\t%d\n\n",P[i],bt[i],at[i],pt[i],tat[i],wt[i]);

 }

 printf("Average_waiting_time = %f\n",WTSUM/n);

 printf("Average_turn_around_time= %f\n",TATSUM/n);

 printf("\n\n");

 printf("Gantt chart\n");

 for(i=1;i<=n;i++)

 {

  printf("\tP%d\t",P[i]);


 }

 printf("\n");

 for(i=1;i<=n;i++)

 {

 printf("\t%d\t",ct[i]);

 }

}
```