PROGRAM 1

```
Welcome to JS/Linux (riscv64)
 Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# pwd
/root
[root@localhost ~]# ls
bench.py      hello.c       hello.js      readme.txt    rv128test.bin
[root@localhost ~]# mkdir mit
[root@localhost ~]# ls
bench.py      hello.js      readme.txt
hello.c       mit           rv128test.bin
[root@localhost ~]# cd mit
[root@localhost mit]# pwd
/root/mit
[root@localhost mit]# mkdir mit exp1
[root@localhost mit]# ls
exp1  mit
[root@localhost mit]# rmdir exp1
[root@localhost mit]# ls
mit
[root@localhost mit]# touch s.txt
[root@localhost mit]# ls
mit    s.txt
[root@localhost mit]# cat > s.txt
Marathwada Institute of Technology
^Z[1]+  Stopped                 cat 1>s.txt
[root@localhost mit]# cat s.txt
Marathwada Institute of Technology
[root@localhost mit]# touch t.txt
[root@localhost mit]# ls
mit    s.txt  t.txt
[root@localhost mit]# cp s.txt t.txt
[root@localhost mit]# cat t.txt
Marathwada Institute of Technology
```

```
[root@localhost mit]# cat > p.txt
Hello Welcome to Aurangabad, Hello this is
first OS program, Hello to all
Hello World
^Z[2]+  Stopped                 cat 1>p.txt
[root@localhost mit]# cat p.txt
Hello Welcome to Aurangabad, Hello this is
first OS program, Hello to all
Hello World
[root@localhost mit]# grep 'Hello' p.txt
Hello Welcome to Aurangabad, Hello this is
first OS program, Hello to all
Hello World
[root@localhost mit]# grep -c 'Hello' p.txt
3
[root@localhost mit]# wc -l p.txt
3 p.txt
```

system call for copying content of one file to other

PROGRAM 2
2.A

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

#define buffersize 10000

int main() {
   char source[25], destination[25];
   char buffer[buffersize];
   ssize_t read_in, write_out;

   printf("Enter source file name: ");
   scanf("%s", source);
   printf("%s\n", source);

   int sourcefiledesc = open(source, O_RDONLY);
   if (sourcefiledesc == -1) {
      perror("Error opening source file");
      exit(1);
   }

   printf("Enter destination file name: ");
   scanf("%s", destination);

   int destfiledesc = open(destination, O_WRONLY | O_CREAT, 0644);
   if (destfiledesc == -1) {
      perror("Error opening destination file");
      exit(1);
   }

   while ((read_in = read(sourcefiledesc, buffer, buffersize)) > 0) {
      write_out = write(destfiledesc, buffer, read_in);
      if (write_out == -1) {
         perror("Error writing to destination file");
```

```
            exit(1);
        }
    }

    close(sourcefiledesc);
    close(destfiledesc);

    return 0;
}
```

Output:

```
Enter source file name: input.txt
input.txt
Enter destination file name: output.txt
```

program for system call of unix operating (fork ,getpid,exit)

2.B

```
#include<stdio.h>
#include<unistd.h>
main()
{
int pid,pid1,pid2;
pid=fork();
if(pid!=0)
{
pid1=getpid();
printf("\n the parent process ID is %d\n", pid1);
}
else
{
pid2=getpid();
printf("\n the child process ID is %d\n", pid2);
}
```

OUTPUT:

the parent process ID is 97

the child process ID is 98

fcfs cpu scheduling algorithms

PROGRAME 3

```
#include<stdio.h>
int main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
{
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
}

}
```

OUTPUT:

```
Enter the number of processes -- 3
Enter Burst Time for Process 0 -- 5
Enter Burst Time for Process 1 -- 8
Enter Burst Time for Process 2 -- 12
PROCESS      BURST TIME          WAITING TIME      TURNAROUND TIME

        P0              5               0               5
Average Waiting Time -- 6.000000
Average Turnaround Time -- 14.333333
        P1              8               5               13
Average Waiting Time -- 6.000000
Average Turnaround Time -- 14.333333
        P2              12              13              25
Average Waiting Time -- 6.000000
Average Turnaround Time -- 14.333333
```

SJF scheduling algorithms

PROGRAM 4

```
#include<stdio.h>

int main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
```

```
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);

}
```

OUTPUT:

```
Enter the number of processes -- 4
Enter Burst Time for Process 0 -- 8
Enter Burst Time for Process 1 -- 4
Enter Burst Time for Process 2 -- 9
Enter Burst Time for Process 3 -- 5
PROCESS      BURST TIME          WAITING TIME      TURNAROUND TIME

        P1              4               0               4
        P3              5               4               9
        P0              8               9               17
        P2              9               17              26
Average Waiting Time -- 7.500000
Average Turnaround Time -- 14.000000
```

producer Consumer problem for process synchronization

PROGRAM 5:

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

#define SIZE 5

int in = 0;
int out = 0;
int buffer[SIZE];

void producer();
void consumer();

int main() {
   int ch;
   while (1) {
      printf("1. Producer\n2. Consumer\n3. Exit\nChoice: ");
      scanf("%d", &ch);

      switch (ch) {
         case 1:
            producer();
            break;
         case 2:
            consumer();
            break;
         case 3:
            exit(0);
         default:
            printf("Enter a proper choice.\n");
      }
   }
}

void producer() {
   int pitem;
   if ((in + 1) % SIZE == out) {
      printf("Buffer is full.\n");
   } else {
      printf("Enter item to produce: ");
```

```
        scanf("%d", &pitem);
        buffer[in] = pitem;
        in = (in + 1) % SIZE;
    }
}

void consumer() {
    int citem;
    if (in == out) {
        printf("Buffer is empty.\n");
    } else {
        citem = buffer[out];
        printf("Item consumed is %d.\n", citem);
        out = (out + 1) % SIZE;
    }
}
```

OUTPUT:

```
1. Producer
2. Consumer
3. Exit
Choice: 1
Enter item to produce: 10

1. Producer
2. Consumer
3. Exit
Choice: 1
Enter item to produce: 20

1. Producer
2. Consumer
3. Exit
Choice: 1
Enter item to produce: 30

1. Producer
2. Consumer
3. Exit
Choice: 2
```

Item consumed is 10.

1. Producer
2. Consumer
3. Exit
Choice: 2
Item consumed is 20.

1. Producer
2. Consumer
3. Exit
Choice: 2
Item consumed is 30.

1. Producer
2. Consumer
3. Exit
Choice: 2
Buffer is empty.

1. Producer
2. Consumer
3. Exit
Choice: 3

producer-consumer problem using semaphores

PROGRAM 6:

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

int mutex = 1, full = 0, empty = 3, x = 0;

void producer();
void consumer();
int wait(int);
int signal(int);

int main() {
   int n;

   printf("\n1. PRODUCER\n2. CONSUMER\n3. EXIT\n");
   while (1) {
      printf("\nENTER YOUR CHOICE: ");
      scanf("%d", &n);

      switch (n) {
         case 1:
            if ((mutex == 1) && (empty != 0))
               producer();
            else
               printf("BUFFER IS FULL\n");
            break;
         case 2:
            if ((mutex == 1) && (full != 0))
               consumer();
            else
               printf("BUFFER IS EMPTY\n");
            break;
         case 3:
            exit(0);
            break;
         default:
            printf("Enter a valid choice.\n");
      }
   }
}
```

```
int wait(int s) {
   return (--s);
}

int signal(int s) {
   return (++s);
}

void producer() {
   mutex = wait(mutex);
   full = signal(full);
   empty = wait(empty);
   x++;
   printf("\nProducer produces item %d\n", x);
   mutex = signal(mutex);
}

void consumer() {
   mutex = wait(mutex);
   full = wait(full);
   empty = signal(empty);
   printf("\nConsumer consumes item %d\n", x);
   x--;
   mutex = signal(mutex);
}
```

OUTPUT:

1. PRODUCER
2. CONSUMER
3. EXIT

ENTER YOUR CHOICE: 1

Producer produces item 1

ENTER YOUR CHOICE: 1

Producer produces item 2

ENTER YOUR CHOICE: 1

Producer produces item 3

ENTER YOUR CHOICE: 1
BUFFER IS FULL

ENTER YOUR CHOICE: 2

Consumer consumes item 3

ENTER YOUR CHOICE: 2

Consumer consumes item 2

ENTER YOUR CHOICE: 2

Consumer consumes item 1

ENTER YOUR CHOICE: 2
BUFFER IS EMPTY

ENTER YOUR CHOICE: 3

Algorithm of first fit

PROGRAM 7
7A

```
#include<stdio.h>
void main()
{
	int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;
	for(i = 0; i< 10; i++)
	{
		flags[i] = 0;
		allocation[i] = -1;
	}
	printf("Enter no. of blocks: ");
	scanf("%d", &bno);
	printf("\nEnter size of each block: ");
	for(i = 0; i<bno; i++)
		scanf("%d", &bsize[i]);
	printf("\nEnter no. of processes: ");
	scanf("%d", &pno);
	printf("\nEnter size of each process: ");
	for(i = 0; i<pno; i++)
		scanf("%d", &psize[i]);
	for(i = 0; i<pno; i++)		//allocation as per first fit
		for(j = 0; j <bno; j++)
			if(flags[j] == 0 &&bsize[j] >= psize[i])
			{
				allocation[j] = i;
				flags[j] = 1;
				break;
			}
	//display allocation details
	printf("\nBlock no.\tsize\t\tprocess no.\t\tsize");
	for(i = 0; i<bno; i++)
	{
		printf("\n%d\t\t%d\t\t", i+1, bsize[i]);
		if(flags[i] == 1)
			printf("%d\t\t\t%d",allocation[i]+1,psize[allocation[i]]);
		else
			printf("Not allocated");
	}
```

}

OUTPUT:

Enter no. of blocks: 4

Enter size of each block: 100 200 300 400

Enter no. of processes: 3

Enter size of each process: 150 250 350

Block no.	size	process no.	size
1	100	1	150
2	200	2	250
3	300	3	350
4	400	Not allocated	

best fit algorithm for memory managment

PROGRAM
7B

```
#include<stdio.h>

void main()
{
int a[20],p[20],i,j,n,m;
printf("Enter no of Blocks.\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
                printf("Enter the %dst Block size:",i);
                scanf("%d",&a[i]);
}
printf("Enter no of Process.\n");
scanf("%d",&m);
for(i=0;i<m;i++)
{
                printf("Enter the size of %dst Process:",i);
                scanf("%d",&p[i]);
}
        for(i=0;i<n;i++)
{
for(j=0;j<m;j++)
                {
                        if(p[j]<=a[i])
                        {
                                printf("The Process %d allocated to %d\n",j,a[i]);
                                p[j]=10000;
                                break;
                        }
                }
}
for(j=0;j<m;j++)
{
if(p[j]!=10000)
                {
printf("The Process %d is not allocated\n",j);
                } }}
```

OUTPUT:

```
Enter no of Blocks.
3
Enter the 1st Block size: 200
Enter the 2nd Block size: 300
Enter the 3rd Block size: 100
Enter no of Process.
4
Enter the size of 1st Process: 150
Enter the size of 2nd Process: 250
Enter the size of 3rd Process: 100
Enter the size of 4th Process: 200
The Process 0 allocated to 200
The Process 1 allocated to 300
The Process 2 is not allocated
The Process 3 allocated to 100
```

FOFO Page Replacement algorithms

EXPERIMENT 8:

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

int main() {
   int i = 0, j = 0, k = 0, i1 = 0, m, n, rs[30], flag = 1, p[30];
   system("clear");
   printf("FIFO page replacement algorithm.\n");
   printf("Enter the number of frames: ");
   scanf("%d", &n);
   printf("Enter the reference string:\n");
   while (1) {
      scanf("%d", &rs[i]);
      if (rs[i] == 0)
         break;
      i++;
   }
   m = i;
   for (j = 0; j < n; j++)
      p[j] = 0;

   for (i = 0; i < m; i++) {
      flag = 1;
      for (j = 0; j < n; j++) {
         if (p[j] == rs[i]) {
            printf("Data already in page.\n");
            flag = 0;
            break;
         }
      }
      if (flag == 1) {
         p[i1] = rs[i];
         i1++;
         k++;
         if (i1 == n)
            i1 = 0;
         for (j = 0; j < n; j++) {
            printf("\nPage %d: %d", j + 1, p[j]);
            if (p[j] == rs[i])
               printf("*");
```

```
            }
            printf("\n\n");
        }
    }
    printf("Total number of page faults = %d\n", k);
    return 0;
}
```

OUTPUT:

```
FIFO page replacement algorithm.
Enter the number of frames: 3
Enter the reference string:
2
3
4
2
1
5
4
6
3
0
Page 1: 2
Page 2: 0
Page 3: 0

Page 1: 2
Page 2: 3
Page 3: 0

Page 1: 2
Page 2: 3
Page 3: 4

Data already in page.

Page 1: 2
Page 2: 3
Page 3: 4*
```

Page 1: 2
Page 2: 1
Page 3: 4

Page 1: 2
Page 2: 1
Page 3: 5

Page 1: 4
Page 2: 1
Page 3: 5

Page 1: 4
Page 2: 6
Page 3: 5

Page 1: 3
Page 2: 6
Page 3: 5

Total number of page faults = 6