ANS 1->

```
#include<stdio.h>

#include<pthread.h>

pthread_t tid[2];

unsigned int shared_data = 0;

pthread_mutex_t mutex;

unsigned int rc;

void* PrintEvenNos(void*);

void* PrintOddNos(void*);

void main(void)

{

pthread_create(&tid[0],0,&PrintEvenNos,0)
pthread_create(&tid[1],0,&PrintOddNos,0);

sleep(3);

pthread_join(tid[0],NULL);

]pthread_join(tid[1],NULL);

 }

 void* PrintEvenNos(void *ptr)

 {

 pthread_mutex_lock(&mutex);

 Do

 {

 if(shared_data%2 == 0)

 {

 printf("Even:%d\n",shared_data);

 Shared_data++;
```

```
 }
 Else
 {
rc=pthread_mutex_unlock(&mutex);//if number is odd, do not print,release mutex
 }
 }
 while(shared_data <= 100);
 }
 void* PrintOddNos(void* ptr1) { rc = pthread_mutex_lock(&mutex);
 Do
 {
 if(shared_data%2 != 0)
 {
 printf("odd:%d\n",shared_data);
 Shared_data++;
 }
 Else
 {
rc = pthread_mutex_unlock(&mutex);//if number is even, do not print,release
mutex
 }
 }
 while(shared_data <= 100); }
```

ANS 2->

```
#include<stdio.h>

#include<stdlib.h>


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


int main()


 int n;

 void producer();

 void consumer();

 int wait(int);

 int signal(int);

 printf("\n1.Producer\n2.Consumer\n3.Exit");

 while(1)

 {

       printf("\nEnter your choice:");

       scanf("%d",&n);

       switch(n)

       {

             case 1:     if((mutex==1)&&(empty!=0))

                               producer();
```

```
                        else
                              printf("Buffer is full!!");
                        break;
                case 2:     if((mutex==1)&&(full!=0))
                              consumer();
                        else
                              printf("Buffer is empty!!");
                        break;
                case 3:
                        exit(0);
                        break;
        }
 }


 return 0;



int wait(int s)


 return (--s);
```

```
int signal(int s)

 return(++s);



id producer()

 mutex=wait(mutex);
 full=signal(full);
 empty=wait(empty);
 x++;
 printf("\nProducer produces the item %d",x);
 mutex=signal(mutex);



id consumer()

 mutex=wait(mutex);
 full=wait(full);
 empty=signal(empty);
```

```
 printf("\nConsumer consumes item %d",x);
 x--;
 mutex=signal(mutex);
}
```

ANS 3->

```
clude <stdio.h>
clude <stdlib.h>
clude <fcntl.h>
clude <errno.h>
clude <sys/types.h>
clude <unistd.h>

fine BUF_SIZE 8192

 main(int argc, char* argv[]) {

 int input_fd, output_fd;    /* Input and output file descriptors */
 ssize_t ret_in, ret_out;    /* Number of bytes returned by read() and write() */
 char buffer[BUF_SIZE];      /* Character buffer */

 /* Are src and dest file name arguments missing */
```

```
 if(argc != 3){
      printf ("Usage: cp file1 file2");
      return 1;
 }

 /* Create input file descriptor */
 input_fd = open (argv [1], O_RDONLY);
 if (input_fd == -1) {
          perror ("open");
          return 2;
 }

 /* Create output file descriptor */
 output_fd = open(argv[2], O_WRONLY | O_CREAT, 0644);
 if(output_fd == -1){
      perror("open");
      return 3;
 }

 /* Copy process */
 while((ret_in = read (input_fd, &buffer, BUF_SIZE)) > 0){
          ret_out = write (output_fd, &buffer, (ssize_t) ret_in);
          if(ret_out != ret_in){
               /* Write error */
```

```
            perror("write");
            return 4;
        }
 }

 /* Close file descriptors */
 close (input_fd);
 close (output_fd);

 return (EXIT_SUCCESS);
```

ANS 4->

```
#include <stdio.h>


 current[5][5], maximum_claim[5][5], available[5];

 allocation[5] = {0, 0, 0, 0, 0};

 maxres[5], running[5], safe = 0;

 counter = 0, i, j, exec, resources, processes, k = 1;


 main()


 printf("\nEnter number of processes: ");

       scanf("%d", &processes);


       for (i = 0; i < processes; i++)

 {

       running[i] = 1;

       counter++;

       }


       printf("\nEnter number of resources: ");

       scanf("%d", &resources);
```

```
	printf("\nEnter Claim Vector:");
	for (i = 0; i < resources; i++)
{
	  scanf("%d", &maxres[i]);
	}

printf("\nEnter Allocated Resource Table:\n");
	for (i = 0; i < processes; i++)
{
	  for(j = 0; j < resources; j++)
	{
		scanf("%d", &current[i][j]);
	}
	}

	printf("\nEnter Maximum Claim Table:\n");
	for (i = 0; i < processes; i++)
{
	for(j = 0; j < resources; j++)
	{
			scanf("%d", &maximum_claim[i][j]);
```

```
	}
	}

printf("\nThe Claim Vector is: ");
	for (i = 0; i < resources; i++)
{
	  printf("\t%d", maxres[i]);
}

	printf("\nThe Allocated Resource Table:\n");
	for (i = 0; i < processes; i++)
{
	  for (j = 0; j < resources; j++)
	{
		printf("\t%d", current[i][j]);
	}
	printf("\n");
	}

	printf("\nThe Maximum Claim Table:\n");
	for (i = 0; i < processes; i++)
```

```
{
        for (j = 0; j < resources; j++)
        {
                printf("\t%d", maximum_claim[i][j]);
        }
        printf("\n");
        }

        for (i = 0; i < processes; i++)
{
        for (j = 0; j < resources; j++)
        {
                    allocation[j] += current[i][j];
        }
        }

        printf("\nAllocated resources:");
        for (i = 0; i < resources; i++)
{
        printf("\t%d", allocation[i]);
        }
```

```
	for (i = 0; i < resources; i++)
{
	  available[i] = maxres[i] - allocation[i];
}

	printf("\nAvailable resources:");
	for (i = 0; i < resources; i++)
{
	printf("\t%d", available[i]);
	}
	printf("\n");

	while (counter != 0)
{
	safe = 0;
	for (i = 0; i < processes; i++)
	{
		if (running[i])
		{
			exec = 1;
```

```
                        for (j = 0; j < resources; j++)
                {
                        if (maximum_claim[i][j] - current[i][j] >
ilable[j])
                        {
                                exec = 0;
                                break;
                        }
                }
                        if (exec)
                {
                        printf("\nProcess%d is executing\n", i +

                        running[i] = 0;
                        counter--;
                        safe = 1;

                        for (j = 0; j < resources; j++)
                        {
                                available[j] += current[i][j];
                        }
```

```
                    break;
                }
            }
}
if (!safe)
{
            printf("\nThe processes are in unsafe state.\n");
            break;
}
else
{
            printf("\nThe process is in safe state");
            printf("\nAvailable vector:");

            for (i = 0; i < resources; i++)
      {
                  printf("\t%d", available[i]);
            }

         printf("\n");
}
```

```
}

return 0;
```