Assigment 4

A. Thread synchronization using counting semaphores. Application to demonstrate: producer-consumer problem with counting semaphores and mutex.

4A_consumer.c

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>
#define maxsize 20
typedef struct
{
	int in,out;
	int list[maxsize];
	sem_t full;
	sem_t emp;
	pthread_mutex_t lock;
}Itemlist;
Itemlist A;
int item,size;
void *prod(void *arg);
void *cust(void *arg);
void init();
void main()
{
	int np,nc,i;
	init();
	pthread_t pd[5],cus[5];
	printf("\nEnter no of producers ");
	scanf("%d",&np);
	printf("\nEnter no of customers ");
	scanf("%d",&nc);
	printf("\nhow many items to be produced ");
	scanf("%d",&size);
	for(i=0;i<np;i++)
	{
		int *arg=malloc(sizeof(int *));
		*arg=i;
		pthread_create(&pd[i],NULL,prod,arg);
		printf("\nproducer thread %d is created",i+1);
	}
	for(i=0;i<nc;i++)
	{
		int *arg=malloc(sizeof(int *));
		*arg=i;
		pthread_create(&cus[i],NULL,cust,arg);
		printf("\ncustomer thread %d is created",i+1);
	}
	for(i=0;i<np;i++)
	{
		pthread_join(pd[i],NULL);
		printf("\nproducer thread %d is finished",i+1);
	}
	for(i=0;i<nc;i++)
	{
		pthread_join(cus[i],NULL);
		printf("\ncustomer thread %d is finished",i+1);
	}
}
void *prod(void *arg)
{
	int i=*(int *)arg;
	while(item<size+1)
	{
		sem_wait(&A.emp);
		pthread_mutex_lock(&A.lock);
		printf("\nproducer %d has produced item %d ",i+1,item);
		A.list[(A.in++)%maxsize]=item++;
		pthread_mutex_unlock(&A.lock);
		sem_post(&A.full);
		sleep(2);
	}

}
void *cust(void *arg)
{
	int i=*(int *)arg;
	while(1)
	{
		sem_wait(&A.full);
		pthread_mutex_lock(&A.lock);
		printf("\ncustomer %d purchased item %d ",i+1,A.list[(A.out++)%maxsize]);
		pthread_mutex_unlock(&A.lock);
		sem_post(&A.emp);
	}

}
void init()
{
	A.in=0;A.out=0;
	sem_init(&A.full,0,0);
	sem_init(&A.emp,0,maxsize);
	item=1;
	pthread_mutex_init(&A.lock,NULL);
}
```

Output:

```
pl-17@pl17-OptiPlex-3020:~/IT/07$ gcc 4A_consumer.c
```

```
pl-17@pl17-OptiPlex-3020:~/IT/07$ ./a.out

Enter no of producers 4

Enter no of customers 8

how many items to be produced 2

producer thread 1 is created
producer 1 has produced item 1
producer thread 2 is created
producer 2 has produced item 2
producer thread 3 is created
producer thread 4 is created
customer thread 1 is created
customer 1 purchased item 1
customer thread 2 is created
customer 1 purchased item 2
customer thread 3 is created
customer thread 4 is created
customer thread 5 is created
customer thread 6 is created
customer thread 7 is created
customer thread 8 is created
producer thread 1 is finished
producer thread 2 is finished
producer thread 3 is finished
```

B. Thread synchronization and mutual exclusion using mutex. Application to demonstrate: Reader-Writer problem with reader priority

4A_rdwt.c

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#include<stdlib.h>
pthread_mutex_t rwmutex;
pthread_mutex_t lock;
int rdcnt,nr,nw;
pthread_t thread;
void *reader(void *arg);
void *writer(void *arg);
void init();
void main()
{
        int i;
        init();
        printf("\nEnter no of readers ");
        scanf("%d",&nr);
```

```
	printf("\nEnter no of writers ");
	scanf("%d",&nw);
	for(i=0;i<nw;i++)
	{
		int *arg=malloc(sizeof(int *));
		*arg=i;
		pthread_create(&thread,NULL,writer,arg);
	}
	for(i=0;i<nr;i++)
	{
		int *arg=malloc(sizeof(int *));
		*arg=i;
		pthread_create(&thread,NULL,reader,arg);
	}

	for(i=0;i<nw;i++)
	{
		pthread_join(thread,NULL);
	}
	for(i=0;i<nr;i++)
	{
		pthread_join(thread,NULL);
	}

}
void init()
{
	pthread_mutex_init(&lock,NULL);
	pthread_mutex_init(&rwmutex,NULL);
	rdcnt=0;
}
void *reader(void *arg)
{
	int i=*(int *)arg;
	int cnt=0;
	printf("\nreader %d is trying to read",i+1);
	pthread_mutex_lock(&lock);
	rdcnt++;
	if(rdcnt==1)
		pthread_mutex_lock(&rwmutex);
	printf("\nreader %d is reading ",i+1);
	pthread_mutex_unlock(&lock);
	sleep(3);
	pthread_mutex_lock(&lock);
	rdcnt--;
	if(rdcnt==0)
		pthread_mutex_unlock(&rwmutex);
	pthread_mutex_unlock(&lock);
	printf("\nreader %d is leaving",i+1);
}
void *writer(void *arg)
{
        int i=*(int *)arg;
        printf("\nwriter %d is trying to write",i+1);
        pthread_mutex_lock(&rwmutex);
        printf("\nwriter %d is writing ",i+1);
        sleep(3);
        pthread_mutex_unlock(&rwmutex);
        printf("\nwriter %d is leaving",i+1);

}
```

Output:

```
pl-17@pl17-OptiPlex-3020:~/IT/07$ gcc 4A_rdwt.c
pl-17@pl17-OptiPlex-3020:~/IT/07$ ./a.out

Enter no of readers 4

Enter no of writers 3

writer 1 is trying to write
writer 1 is writing
writer 2 is trying to write
writer 3 is trying to write
reader 1 is trying to read
reader 2 is trying to read
reader 3 is trying to read
reader 4 is trying to read
writer 1 is leaving
writer 2 is writing
writer 2 is leaving
writer 3 is writing
writer 3 is leaving
reader 1 is reading
reader 2 is reading
reader 3 is reading
reader 4 is reading
reader 1 is leaving
reader 2 is leaving
reader 4 is leaving
reader 3 is leaving
```