IPC Problems Using Threads

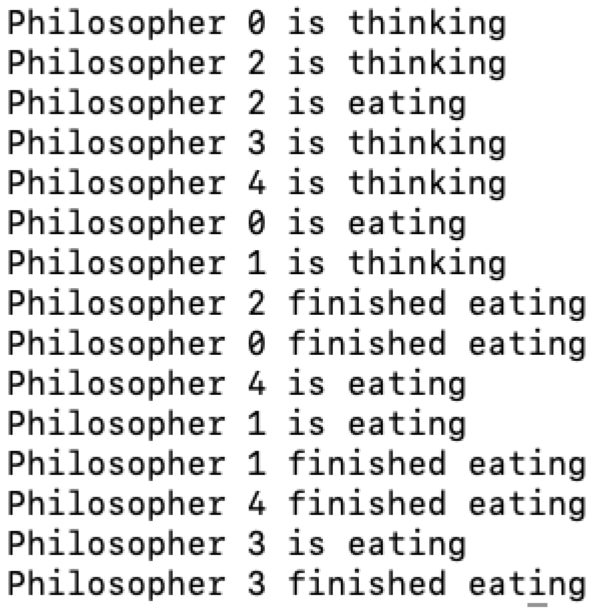
1. Dining philosophers Problem

Code:

```
#include<stdio.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#include<iostream>
#define EATING 2
#define THINKING 0
#define HUNGRY 1
using namespace std;
int State[5];
sem_t S[5];
sem_t mutex;
int TimesEaten[5];
void test(int i)
{
    if(State[i]==HUNGRY&&State[(i+1)%5]!=EATING&&State[(i+
4)%5]!=EATING)
    {
        sem_post(&S[i]);
    }
}
void Pickup(int i)
{
    sem_wait(&mutex);
    State[i]=HUNGRY;
    cout<<"Philosopher "<<i<<" is hungry\n";
    sleep(1);
    test(i);
    sem_post(&mutex);
    sem_wait(&S[i]);
```

```
}
void PutDown(int i)
{
    sem_wait(&mutex);
    State[i]=THINKING;
    test((i+1)%5);
    test((i+4)%5);
    sem_post(&mutex);
}
void * Philosopher(void * arg)
{
    int i= *((int *) arg);
    while(1)
    {
        cout<<"Philosopher "<<i<<" is thinking\n";
        sleep(2);
        Pickup(i);
        State[i]=EATING;
        TimesEaten[i]++;
        cout<<"Philosopher "<<i<<" is eating for
the"<<TimesEaten[i]<<"th time\n";
        sleep(2);
        PutDown(i);
    }
}
int main()
{
    int Phil[5]={0,1,2,3,4};
    pthread_t T[5];
    for(int i=0;i<5;i++)
    {
        State[i]=THINKING;
        TimesEaten[i]=0;
        sem_init(&S[i],0,0);
    }
    sem_init(&mutex,0,1);
    for(int i=0;i<5;i++)
    {
        pthread_create(&T[i],NULL,Philosopher,&Phil[i]);
    }
    for(int i=0;i<5;i++)
    {
        pthread_join(T[i],NULL);
    }
}
```

Output:



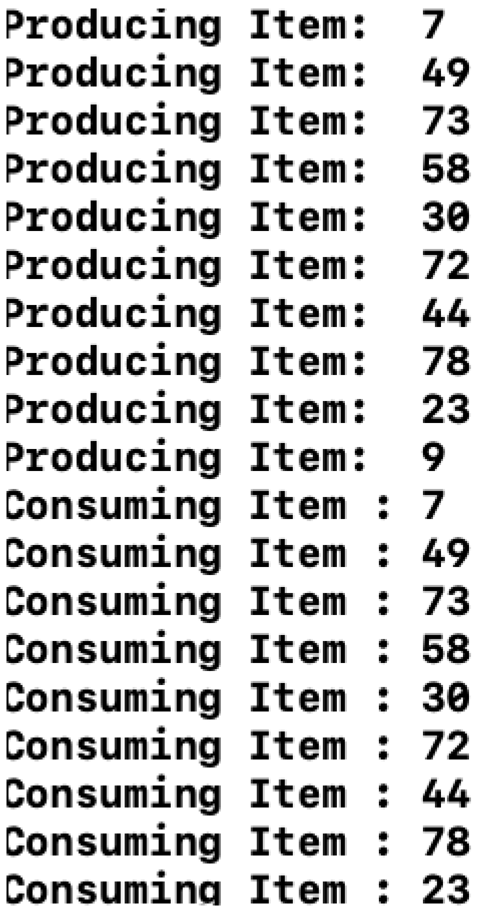
```
Philosopher 0 is thinking
Philosopher 2 is thinking
Philosopher 2 is eating
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 0 is eating
Philosopher 1 is thinking
Philosopher 2 finished eating
Philosopher 0 finished eating
Philosopher 4 is eating
Philosopher 1 is eating
Philosopher 1 finished eating
Philosopher 4 finished eating
Philosopher 3 is eating
Philosopher 3 finished eating
```

2. Producer Consumer Problem

Code:

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#define MAXITEMS 10
typedef int item;
item buffer[MAXITEMS];
int in=0;
int out=0;
pthread_mutex_t mv =PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t Bufferfull=PTHREAD_COND_INITIALIZER;
pthread_cond_t Bufferempty=PTHREAD_COND_INITIALIZER;
void *producer()
{
    item it;
    while(1)
    {
        pthread_mutex_lock(&mv);
        it=rand()%100;
        printf("Producing Item:    %d \n",it);
        if((in+1)%MAXITEMS==out)
        {
            pthread_cond_wait(&Bufferempty,&mv);
        }
        buffer[in]=it;
        in=(in+1)%MAXITEMS;
        pthread_mutex_unlock(&mv);
        pthread_cond_signal(&Bufferfull);
    }
}
void *Consumer()
{
    item it;
    while(1)
    {
        pthread_mutex_lock(&mv);
        if(in==out)
        {
            pthread_cond_wait(&Bufferfull,&mv);
        }
        it=buffer[out];
        printf("Consuming Item : %d \n",it);
        out=(out+1)%MAXITEMS;
        pthread_mutex_unlock(&mv);
        pthread_cond_signal(&Bufferempty);
    }
}
int main()
{
    pthread_t consumertid,producertid;
    pthread_create(&producertid,NULL,producer,NULL);
    pthread_create(&consumertid,NULL,Consumer,NULL);
    pthread_join(producertid,NULL);
    pthread_join(consumertid,NULL);
    return 0;
}
```

Output:



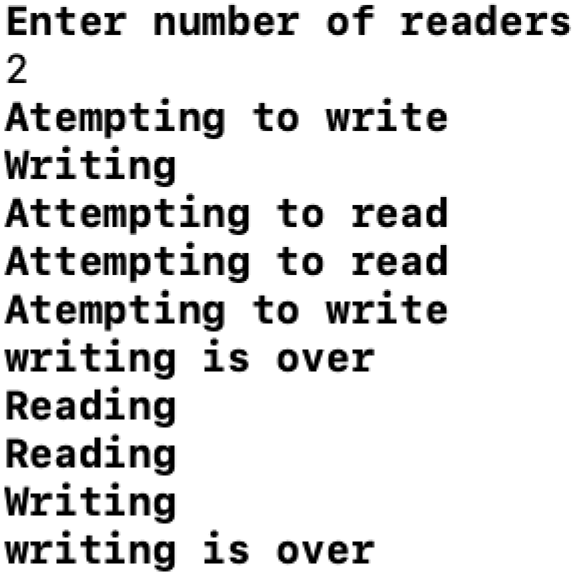
```
Producing Item:  7
Producing Item:  49
Producing Item:  73
Producing Item:  58
Producing Item:  30
Producing Item:  72
Producing Item:  44
Producing Item:  78
Producing Item:  23
Producing Item:  9
Consuming Item : 7
Consuming Item : 49
Consuming Item : 73
Consuming Item : 58
Consuming Item : 30
Consuming Item : 72
Consuming Item : 44
Consuming Item : 78
Consuming Item : 23
```

3. Readers Writers Problem

Code:

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_mutex_t lock,wrt;
int readercount=0;
void    *reader(void * v)
{
    printf("Attempting to read\n");
    pthread_mutex_lock(&lock);
    readercount++;
    if(readercount==1)
    {
        pthread_mutex_lock(&wrt);
    }
    int p=rand()%5;
    printf("Reading\n");
    pthread_mutex_unlock(&lock);
    sleep(p);
    pthread_mutex_lock(&lock);
    readercount--;
    if(readercount==0)
    pthread_mutex_unlock(&wrt);
    pthread_mutex_unlock(&lock);
}
void *writer(void * v)
{
    int p=rand()%3;
    printf("Atempting to write\n");
    pthread_mutex_lock(&wrt);
    printf("Writing \n");
    sleep(p);
    pthread_mutex_unlock(&wrt);
}
int main()
{
    pthread_t Thread[30];
    pthread_mutex_init(&lock,NULL);
    pthread_mutex_init(&wrt,NULL);
    for(int i=0;i<30;i++)
    {
        pthread_create(&Thread[i],NULL,&reader,NULL);
        pthread_create(&Thread[29-i],NULL,writer,NULL);
    }
    for(int i=0;i<10;i++)
    pthread_join(Thread[i],NULL);
    return 0;
}
```

Output:



```
Enter number of readers
2
Atempting to write
Writing
Attempting to read
Attempting to read
Atempting to write
writing is over
Reading
Reading
Writing
writing is over
```

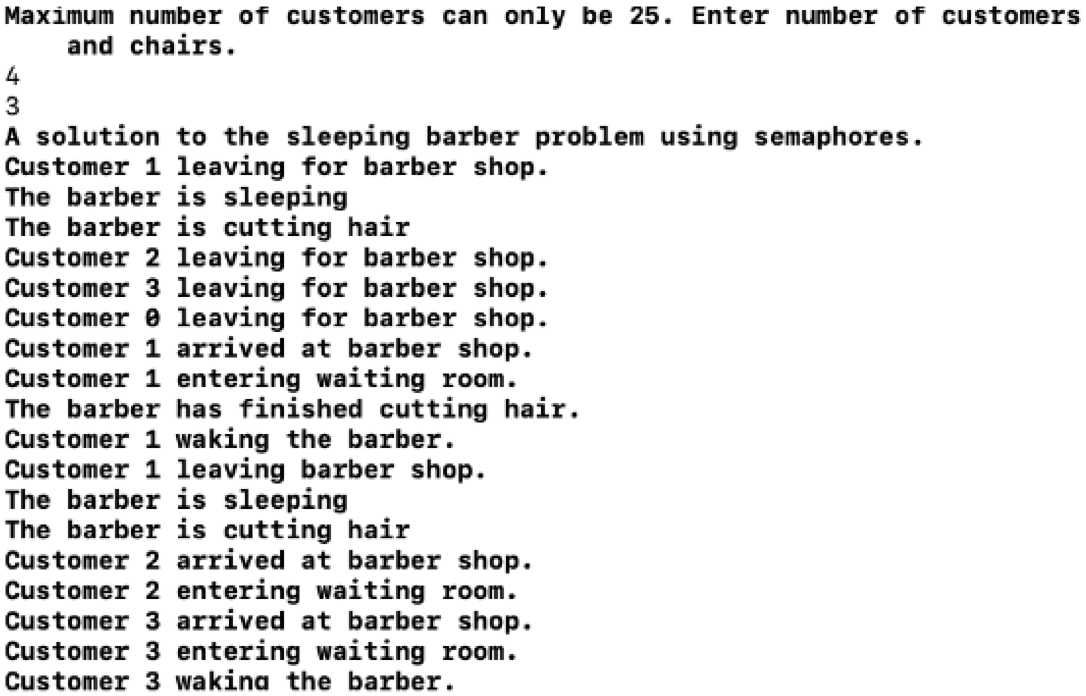
4. Sleeping Barber Problem

Code:

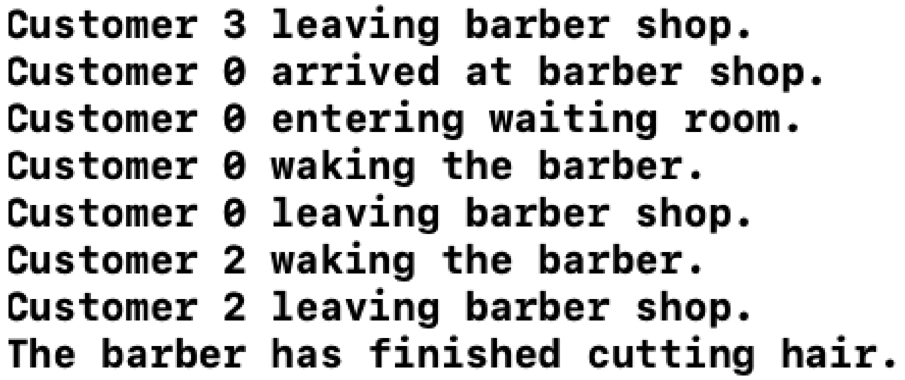
```
#include<iostream>
#include<semaphore.h>
```

```
#include<pthread.h>
#define MAXCHAIRS 5
#include<unistd.h>
using namespace std;
sem_t customer=0,barber=0,donecutting=0;
pthread_mutex_t lock;
int nowaiting=0;
void * Barber(void *Arg)
{
    while(true)
    {
        sem_wait(&customer);
        cout<<"Barber allocated\n";
        sem_post(&barber);
        cout<<"Barber is cutting hair\n";
        sleep(2);
        sem_post(&donecutting);
        pthread_mutex_lock(&lock);
        nowaiting--;
        pthread_mutex_unlock(&lock);
    }
}
void * Customer(void *)
{
    pthread_mutex_lock(&lock);
    if(nowaiting<MAXCHAIRS)
    {
        nowaiting++;
        cout<<"Customer Sitting\n";
        pthread_mutex_unlock(&lock);
        sem_post(&customer);
        sem_wait(&barber);
        sem_wait(&donecutting);
        cout<<"Customer exiting after getting service \n";
    }
    else
    {
        pthread_mutex_unlock(&lock);
    }
}
int main()
{
    pthread_t Thread[40];
    sem_init(&customer,0,0);
    sem_init(&barber,0,0);
    sem_init(&donecutting,0,0);
    pthread_create(&Thread[0],NULL,Barber,NULL);
    for(int i=1;i<40;i++)
    {
        pthread_create(&Thread[i],NULL,Customer,NULL);
    }
    for(int i=1;i<40;i++)
    {
        pthread_join(Thread[i],NULL);
    }
}
```

Output:



```
Maximum number of customers can only be 25. Enter number of customers
    and chairs.
4
3
A solution to the sleeping barber problem using semaphores.
Customer 1 leaving for barber shop.
The barber is sleeping
The barber is cutting hair
Customer 2 leaving for barber shop.
Customer 3 leaving for barber shop.
Customer 0 leaving for barber shop.
Customer 1 arrived at barber shop.
Customer 1 entering waiting room.
The barber has finished cutting hair.
Customer 1 waking the barber.
Customer 1 leaving barber shop.
The barber is sleeping
The barber is cutting hair
Customer 2 arrived at barber shop.
Customer 2 entering waiting room.
Customer 3 arrived at barber shop.
Customer 3 entering waiting room.
Customer 3 waking the barber.
```



```
Customer 3 leaving barber shop.
Customer 0 arrived at barber shop.
Customer 0 entering waiting room.
Customer 0 waking the barber.
Customer 0 leaving barber shop.
Customer 2 waking the barber.
Customer 2 leaving barber shop.
The barber has finished cutting hair.
```