IPC Problems Using Semaphore

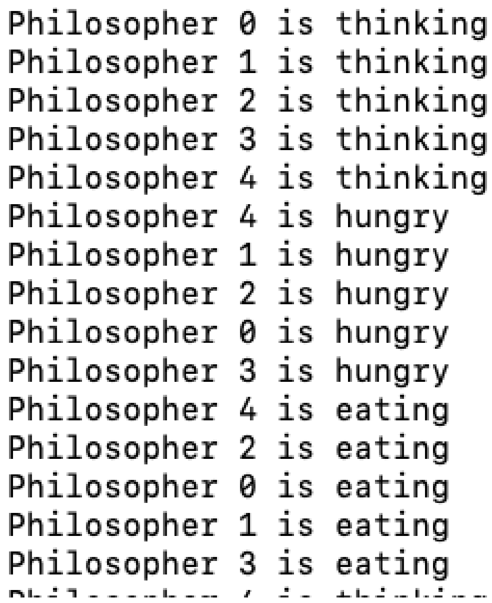
1. Dining Philosophers Problem

Code:

```
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<sys/wait.h>
#include<sys/sem.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#define EATING 0
#define HUNGRY 1
#define THINKING 2
#define KEY 123
union semun {
        int val;
        struct semid_ds *buf;
        unsigned short    *array;
};
struct sembuf p = { 0, -1, SEM_UNDO};    // WAIT
struct sembuf v = { 0, +1, SEM_UNDO};      // SIGNAL
struct smph
{
     int State[5];
};
void Initialize(struct smph * SHM )
{
     for(int i=0;i<5;i++)
     {
          SHM->State[i]=THINKING;
     }
}
void test(int i, struct smph * SHM, int semid)
```

```
{
    if(SHM->State[i]==HUNGRY&&SHM->State[(i+1)%5]!=EATIN
G&&SHM->State[(i+4)%5]!=EATING)
    {
        v.sem_num=i;
        semop(semid,&v,1);
        //sem_post(&S[i]);
    }
}
void Pickup(int i,struct smph * shm, int semid)
{
    p.sem_num=5;
    semop(semid,&p,1);
    //sem_wait(&mutex);
    shm->State[i]=HUNGRY;
    printf("Philosopher %d is hungry\n", i);
    sleep(1);
    test(i,shm,semid);
    v.sem_num=5;
    semop(semid,&v,1);
    //sem_post(&mutex);
    p.sem_num=i;
    semop(semid,&p,1);
    //sem_wait(&S[i]);
}
void PutDown(int i, struct smph * shm, int semid)
{
    //sem_wait(&mutex);
    p.sem_num=5;
    semop(semid,&p,1);
    shm->State[i]=THINKING;
    test((i+1)%5, shm, semid);
    test((i+4)%5, shm, semid);
    v.sem_num=5;
    semop(semid,&v,1);
    //sem_post(&mutex);
}
void    Philosopher(int i, struct smph * shm, int semid)
{
    while(1)
    {
        printf("Philosopher %d is thinking\n",i);
        sleep(2);
        Pickup(i,shm,semid);
        shm->State[i]=EATING;
        printf("Philosopher %d is eating \n",i);
        sleep(2);
        PutDown(i,shm,semid);
    }
}
int main()
{
    int shmid,semid,key;
    struct smph * shm;
    shmid=shmget(key,sizeof(struct smph),IPC_CREAT|0660);
    if(shmid == -1)
    perror("Shared Memory fault\n");
    shm= (struct smph *)shmat(shmid, NULL, 0);
    if(shm == (void *) - 1)
        perror("Attachment fault\n");
    Initialize(shm);
    union semun u;
    semid = semget(KEY, 6, 0666 | IPC_CREAT);
    u.val = 0;
    for(int i=0;i<5;i++)
    {
        semctl(semid, i, SETVAL, u);
    }
    u.val = 1;
    semctl(semid, 5, SETVAL, u);
    for(int i=0;i<5;i++)
    {
        if(fork() == 0)
        {
            shm = (struct smph *)shmat(shmid, NULL, 0);
            Philosopher(i, shm, semid);
            break;
        }
    }
    wait(NULL);
}
```

Output:



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

2. Producer Consumer Problem

Code:

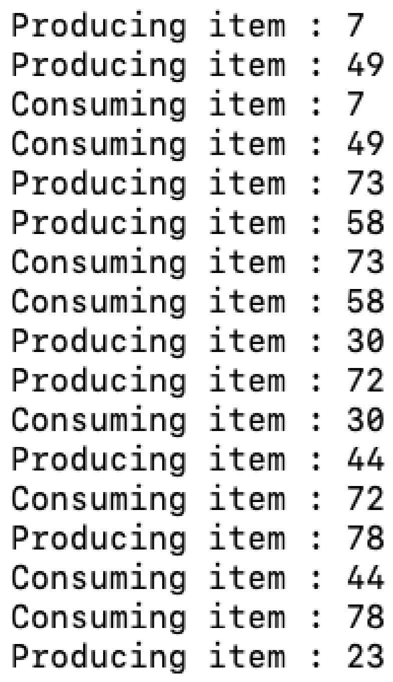
```
#include<unistd.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<stdlib.h>
#include<sys/wait.h>
```

```
#include<sys/sem.h>
#define KEY 123
#define N 10
union semun {
        int val;
        struct semid_ds *buf;
        unsigned short    *array;
};
struct sembuf p = { 0, -1, SEM_UNDO};    // WAIT
struct sembuf v = { 0, +1, SEM_UNDO};      // SIGNAL
struct smph
{
     int Array[10];
     int in;
     int out;
};
void Producers(int semid, struct smph * shm)
{
     int item=rand()%100;
     p.sem_num=0;
     semop(semid,&p,1);
     p.sem_num=2;
     semop(semid,&p,1);
     shm->Array[shm->in]=item;
     printf("Producing item : %d \n",item);
     shm->in=(shm->in+1)%N;
     v.sem_num=2;
     semop(semid,&v,1);
     v.sem_num=1;
     semop(semid,&v,1);
}
void Consumers(int semid, struct smph * shm)
{
     int item;
     p.sem_num=1;
     semop(semid,&p,1);              //    Wait for empty
     p.sem_num=2;
     semop(semid,&p,1);
     item=shm->Array[shm->out];
     shm->out=(shm->out+1)%N;
     printf("Consuming item : %d \n",item);
     v.sem_num=2;
     semop(semid,&v,1);
// Signal for full
     v.sem_num=0;
     semop(semid,&v,1);

}
int main()
{
     int shmid,semid,key;
     struct smph * shm;
     shmid=shmget(key,sizeof(struct smph),IPC_CREAT|0660);
     if(shmid==-1)
     perror("Shared Memory fault\n");
     shm=shmat(shmid,NULL,0);
     if(shm== (void *) -1)
          perror("Attachment fault\n");
     shm->in=0;
     shm->out=0;
     union semun u;
     u.val = N;
     semid = semget(KEY, 3, 0666 | IPC_CREAT);
     semctl(semid, 0, SETVAL, u);
     u.val=0;
     semctl(semid,1,SETVAL,u);
     u.val=1;
     semctl(semid,2,SETVAL,u);
     int pid=fork();
     if(pid==0)
```

```
    {
        shm=shmat(shmid,NULL,0);
        while(1)
        Producers(semid,shm);
    }
    else
    {
        //shm=shmat(shmid,NULL,0);
        while(1)
        Consumers(semid,shm);
    }
    return 0;
}
```

Output:



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

3. Readers Writers Problem

Code:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
```

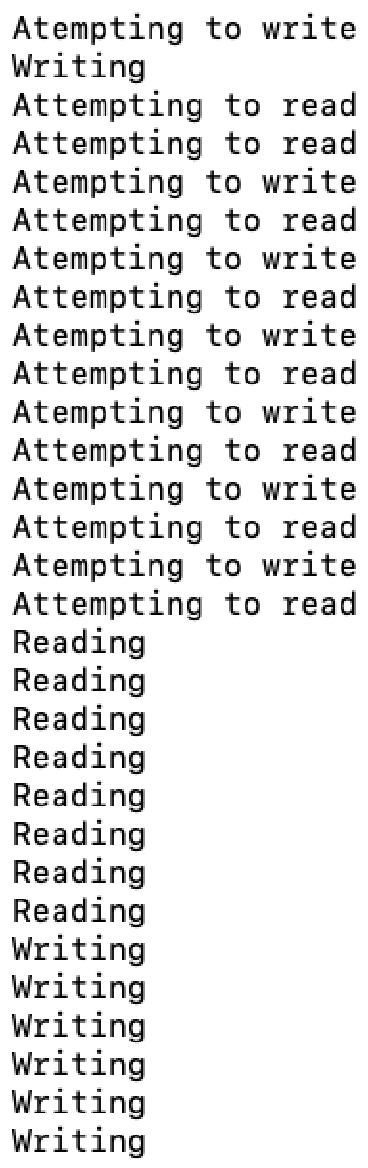
```
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/sem.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#define KEY 123
union semun {
        int val;
        struct semid_ds *buf;
        unsigned short    *array;
};
struct sembuf p = { 0, -1, SEM_UNDO};    // WAIT
struct sembuf v = { 0, +1, SEM_UNDO};      // SIGNAL
struct smph
{
     int readercount;
};
void    reader(int semid, struct smph * shm)
{
     printf("Attempting to read\n");
     p.sem_num=0;
     semop(semid,&p,1);
     shm->readercount++;
     if(shm->readercount==1)
     {
          p.sem_num=1;
          semop(semid,&p,1);
     }
     int q=rand()%5;
     printf("Reading\n");
     v.sem_num=0;
     semop(semid,&v,1);
     sleep(q);
     p.sem_num=0;
```

```
    semop(semid,&p,1);
    shm->readercount--;
    if(shm->readercount==0)
    {
        v.sem_num=1;
        semop(semid,&v,1);
    }
    v.sem_num=0;
    semop(semid,&v,1);
}
void writer(int semid, struct smph * shm)
{
    int q=rand()%3;
    printf("Atempting to write\n");
    p.sem_num=1;
    semop(semid,&p,1);
    printf("Writing \n");
    sleep(q);
    v.sem_num=1;
    semop(semid,&v,1);
}
int main()
{
    struct smph *shm;
    int shmid,semid,key,i;
    shmid=shmget(key,sizeof(struct smph),IPC_CREAT|0660);
    if(shmid==-1)
    perror("Shared Memory fault\n");
    shm=shmat(shmid,NULL,0);
    if(shm== (void *) -1)
        perror("Attachment fault\n");
        shm->readercount=0;
    union semun u;
    semid = semget(KEY, 2, 0666 | IPC_CREAT);
    u.val=1;
    semctl(semid, 0, SETVAL, u); // Lock
    semctl(semid,1,SETVAL,u);         // Write LOck
    for(i=0;i<15;i++)
    {
        int pid=fork();
        if(pid==0&&(i%2)==0)
        {
            shm= (struct    smph *) shmat(shmid,NULL,0);
            reader(semid,shm);
            break;
        }
        else if(pid==0&&(i%2)==1)
        {
            shm= (struct    smph *) shmat(shmid,NULL,0);
            writer(semid,shm);
            break;
        }
    }

    while(2)
    {
        int r=wait(NULL);
        if(r<0)
        break;
    }


    return 0;
}
```

Output:



```
Atempting to write
Writing
Attempting to read
Attempting to read
Atempting to write
Attempting to read
Atempting to write
Attempting to read
Atempting to write
Attempting to read
Atempting to write
Attempting to read
Atempting to write
Attempting to read
Atempting to write
Attempting to read
Reading
Reading
Reading
Reading
Reading
Reading
Reading
Reading
Writing
Writing
Writing
Writing
Writing
Writing
```

4. Sleeping Barber Problem

Code:

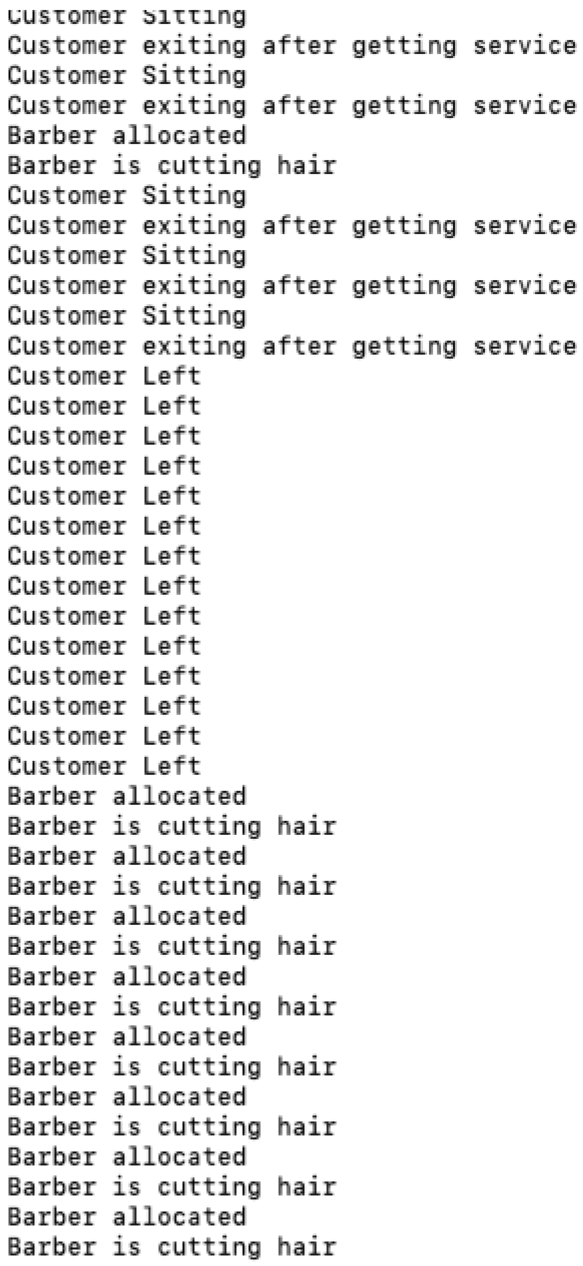
```
#include<stdio.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<semaphore.h>
#include<sys/wait.h>
#include<pthread.h>
#define MAXCHAIRS 5
```

```
#include<unistd.h>
#include<sys/shm.h>
#include<stdlib.h>
#define CUSTOMER 1
#define BARBER 0
#define DONECUTTING 2
#define LOCK 3
#define KEY 123
struct smph
{
    int nowaiting;
};
// Customer 0
// Barber 1
// Done cutting 2
union semun {
      int val;
      struct semid_ds *buf;
      unsigned short    *array;
};
void Post(int semid, int sid)
{
    struct sembuf v = { 0, +1,0};      // SIGNAL
    v.sem_num=sid;
    semop(semid,&v,1);
}
void Wait(int semid, int sid)
{
    struct sembuf p = { 0, -1,0};    // WAIT
    p.sem_num=sid;
    semop(semid,&p,1);
}
void    Barber(int semid, struct smph * shm)
{
    while(1)
    {
        Wait(semid,CUSTOMER);
        printf("Barber allocated\n");
        Post(semid,BARBER);
        printf("Barber is cutting hair\n");
        sleep(2);
        Post(semid, DONECUTTING);
        Wait(semid, LOCK);
        shm->nowaiting--;
        Post(semid,LOCK);
    }
}
void Customer(int semid, struct smph * shm)
{
    Wait(semid,LOCK);
    if(shm->nowaiting<MAXCHAIRS)
    {
        shm->nowaiting++;
        printf("Customer Sitting\n");
        Post(semid,LOCK);
        Post(semid,CUSTOMER);
        Wait(semid,BARBER);
        Wait(semid,DONECUTTING);
        printf("Customer exiting after getting service \n");
    }
    else
    {
                printf("Customer Left\n");
        Post(semid,LOCK);
    }
}
int main()
{
    int shmid,semid,key,i;
    struct smph * shm;
```

```
shmid=shmget(key,sizeof(struct smph),IPC_CREAT|0660);
if(shmid==-1)
perror("Shared Memory fault\n");
shm= (struct smph *)shmat(shmid,NULL,0);
if(shm== (void *) -1)
    perror("Attachment fault\n");
shm->nowaiting=0;
union semun u;
semid = semget(KEY, 4, 0660 | IPC_CREAT);
u.val=0;
for(int i=0;i<3;i++)
{
    semctl(semid, i, SETVAL, u);
}
u.val=1;
semctl(semid,3,SETVAL,u);
for(i=0;i<20;i++)
{
    int pid=fork();
    if(pid==0&&i==0)
    {
        //semid = semget(KEY, 4, 0666 | IPC_CREAT);
        shm= (struct smph *)shmat(shmid,NULL,0);
        Barber(semid, shm);
        exit(0);
    }
    else if(pid==0&&i!=0)
    {
        //semid = semget(KEY, 4, 0666 | IPC_CREAT);
        shm= (struct smph *)shmat(shmid,NULL,0);
        Customer(semid, shm);
        exit(0);
    }
}
while(1)
```

```
    {
        int y=wait(NULL);
        if(y<0)
        break;
    }
}
```

Output:



```
Customer Sitting
Customer exiting after getting service
Customer Sitting
Customer exiting after getting service
Barber allocated
Barber is cutting hair
Customer Sitting
Customer exiting after getting service
Customer Sitting
Customer exiting after getting service
Customer Sitting
Customer exiting after getting service
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Customer Left
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
Barber allocated
Barber is cutting hair
```