**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++)

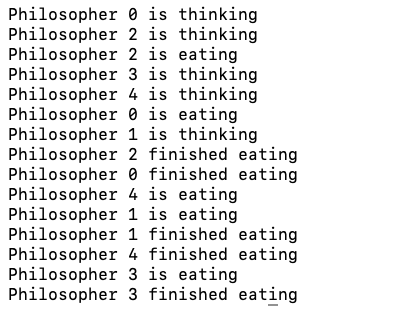
{

pthread\_join(T[i],NULL);

}

}

**Output:**



**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);

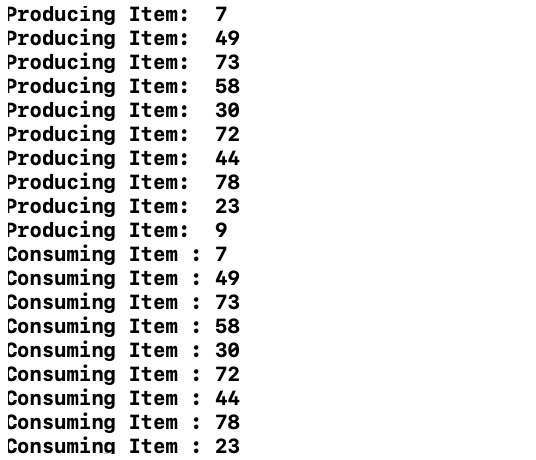
pthread\_join(producertid,NULL);

pthread\_join(consumertid,NULL);

return 0;

}

**Output:**



**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);

}

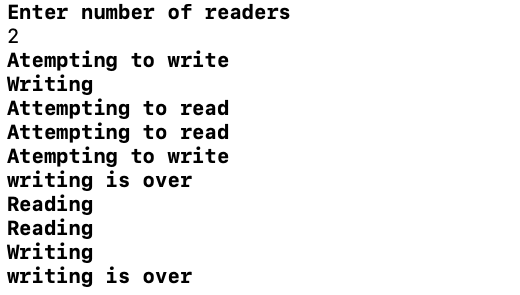
for(int i=0;i<10;i++)

pthread\_join(Thread[i],NULL);

return 0;

}

**Output:**



**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:**

