# Reader’s Preference Code

## Writer

#include <iostream>

#include <fstream>

#include <string>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <inttypes.h>

using namespace std;

#define SHM\_KEY 9876 // the shared process id

#define SEMKEY 1234

struct sembuf vsembuf, psembuf, vsembuf1, psembuf1; // struct with (p, v) being down, up

int main()

{

union semun{ // like struct but all members share the same location

int val;

struct semid\_ds \*buf;

ushort myArray[0];

} arg;

string ip;

ofstream myFile; // makes an ofstream object to read from myFile

int shmid = shmget(SHM\_KEY, 256, 0777|IPC\_CREAT); // 0777 is the permission, IPC\_CREAT creates a shared mem block

int \*readerCount = (int\*)shmat(shmid, 0, 0); // starting addr of 1, 0 is full R/W

int semid = semget(SEMKEY, 2, 0777|IPC\_CREAT); // Creates two semaphores

readerCount = 0;

psembuf1.sem\_num=1; // resource

psembuf1.sem\_op=-1;

psembuf1.sem\_flg=SEM\_UNDO;

vsembuf1.sem\_num=1;

vsembuf1.sem\_op=1;

vsembuf1.sem\_flg=SEM\_UNDO;

arg.val = 1;

semctl(semid, 0, SETVAL, arg);

semctl(semid, 1, SETVAL, arg);

while(1){

cout << "\nWriter1: \n";

semop(semid, &psembuf1, 1); // lock file for writer

myFile.open ("myFile.txt", ios::out | ios::app); // ::app appends the myFile (new line)

if(myFile.is\_open()){

cout << "Please enter something\n";

cin >> ip;

myFile << ip << endl;

myFile.close();

}

semop(semid, &vsembuf1, 1); // unlock file for writers if no readers request

}

return 0;

}

## Reader

#include <iostream>

#include <fstream>

#include <string>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <inttypes.h>

using namespace std;

#define SHM\_KEY 9876 // the shared process id

#define SEMKEY 1234

struct sembuf vsembufR, psembufR, vsembufF, psembufF; // Reader Mutex and Resource Mutex

int main()

{

union semun{ // like struct but all members share the same location

int val;

struct semid\_ds \*buf;

ushort myArray[0];

} arg;

string op;

ifstream myFile; // makes an ifstream object to read from myFile

int shmid = shmget(SHM\_KEY, 256, 0777|IPC\_CREAT);// 0777 is the permission, IPC\_CREAT creates a shared mem block

int semid = semget(SEMKEY, 2, 0777|IPC\_CREAT); // Creates two semaphores

int \*readerCount = (int\*)shmat(shmid, 0, 0); // starting addr of 1, 0 is full R/W

int pause; // used as later for pausing program

\*readerCount = 0;

psembufR.sem\_num=0; // init mutex members

psembufR.sem\_op=-1; // what value to use with semop

psembufR.sem\_flg=SEM\_UNDO; // SEM\_UNDO makes kernel increment for process after decrementing semap

vsembufR.sem\_num=0;

vsembufR.sem\_op=1;

vsembufR.sem\_flg=SEM\_UNDO;

psembufF.sem\_num=1; // resource

psembufF.sem\_op=-1;

psembufF.sem\_flg=SEM\_UNDO;

vsembufF.sem\_num=1;

vsembufF.sem\_op=1;

vsembufF.sem\_flg=SEM\_UNDO;

arg.val = 1; // sets to binary semap

semctl(semid, 0, SETVAL, arg); // initialises semaphore 0. Do this once only

semctl(semid, 1, SETVAL, arg);

while(1){

cout << "Reader1:\n";

pause = getchar(); // pause prevents error

semop(semid, &psembufR, 1); // lock reader mutex

\*readerCount++;

if(\*readerCount == 1){ // is this first reader

cout << "Locking reader from writers\n";

semop(semid, &psembufF, 1); // lock resource from writers if 1st reader

}

semop(semid, &vsembufR, 1); // unlock reader mutex

// Critical Section

myFile.open ("myFile.txt", ios::out | ios::app); // ::app appends the myFile (new line)

if(myFile.is\_open()){

while(getline(myFile, op)){

cout << op << endl;

}

myFile.close();

}

semop(semid, &psembufR, 1); // lock reader mutex, can use readerCount

\*readerCount--;

if(\*readerCount == 0) { // is this the last reader

semop(semid, &vsembufF, 1); // unlock resource mutex for writers

cout << "Locking reader mutex for readers\n";

}

semop(semid, &vsembufR, 1); // unlock reader mutex (release)

semop(semid, &vsembufF, 1); // unlock resource mutex

}

return 0;

}

# Writer’s Preference Code

## Writer

#include <iostream>

#include <fstream>

#include <string>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <inttypes.h>

using namespace std;

#define SHM\_KEY 9876 // the shared process id

#define SEMKEY 1234

struct sembuf vsembufR, psembufR, vsembufW, psembufW; // reader and writer semaphores

struct sembuf vsembufF, psembufF, vsembufT, psembufT; // resource (file) and read try semaphores

int main()

{

union semun{ // like struct but all members share the same location

int val;

struct semid\_ds \*buf;

ushort myArray[0];

} arg;

string ip;

ofstream myFile; // makes an ifstream object to read from myFile

int shmid = shmget(SHM\_KEY, 256, 0777|IPC\_CREAT); // 0777 is the permission, IPC\_CREAT creates a shared mem block

int \*writerCount = (int\*)shmat(shmid, 0, 0); // starting addr of 1, 0 is full R/W

int semid = semget(SEMKEY, 2, 0777|IPC\_CREAT); // Creates two semaphores

int pause;

\*writerCount = 0;

psembufR.sem\_num=0; // init reader mutex members

psembufR.sem\_op=-1;

psembufR.sem\_flg=SEM\_UNDO;

vsembufR.sem\_num=0;

vsembufR.sem\_op=1;

vsembufR.sem\_flg=SEM\_UNDO;

psembufF.sem\_num=1; // resource

psembufF.sem\_op=-1;

psembufF.sem\_flg=SEM\_UNDO;

vsembufF.sem\_num=1;

vsembufF.sem\_op=1;

vsembufF.sem\_flg=SEM\_UNDO;

psembufW.sem\_num=0; // writer

psembufW.sem\_op=-1;

psembufW.sem\_flg=SEM\_UNDO;

vsembufW.sem\_num=0;

vsembufW.sem\_op=1;

vsembufW.sem\_flg=SEM\_UNDO;

psembufT.sem\_num=1; // try

psembufT.sem\_op=-1;

psembufT.sem\_flg=SEM\_UNDO;

vsembufT.sem\_num=1;

vsembufT.sem\_op=1;

vsembufT.sem\_flg=SEM\_UNDO;

arg.val = 1;

semctl(semid, 0, SETVAL, arg); // initialises semaphore 0. Do this once only

semctl(semid, 1, SETVAL, arg);

while(1){

cout << "Writer1:\n";

semop(semid, &psembufW, 1); // lock writer to prevent race conditions

\*writerCount++;

if(\*writerCount == 1){ // if 1st writer

semop(semid, &psembufT, 1); // lock the try semap (from 1st reader)

}

semop(semid, &vsembufW, 1); // unlock writer to release entry section

semop(semid, &psembufF, 1); // lock resource from other writers

// Critical Section

myFile.open ("myFile.txt", ios::out | ios::app);// ::app appends the myFile (new line)

if(myFile.is\_open()){

cout << "Please enter something\n";

cin >> ip;

myFile << ip << endl;

myFile.close();

}

semop(semid, &vsembufF, 1); // unlock file for other writers

semop(semid, &psembufW, 1); // lock writer to reserve exit section

writerCount--;

if(writerCount == 0){ // if last writer

semop(semid, &vsembufT, 1); // unlock the try semap, if last writer

}

semop(semid, &vsembufW, 1); // unlock writer mutex, release exit section

}

}

## Reader

#include <iostream>

#include <fstream>

#include <string>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <inttypes.h>

using namespace std;

#define SHM\_KEY 9876 // the shared process id

#define SEMKEY 1234

struct sembuf vsembufR, psembufR, vsembufW, psembufW; // reader and writer semaphores

struct sembuf vsembufF, psembufF, vsembufT, psembufT; // resource (file) and read try semaphores

int main()

{

union semun{ // like struct but all members share the same location

int val;

struct semid\_ds \*buf;

ushort myArray[0];

} arg;

string op;

ifstream myFile; // makes an ifstream object to read from myFile

int shmid = shmget(SHM\_KEY, 256, 0777|IPC\_CREAT);// 0777 is the permission, IPC\_CREAT creates a shared mem block

int \*readerCount = (int\*)shmat(shmid, 0, 0); // starting addr of 1, 0 is full R/W

int semid = semget(SEMKEY, 2, 0777|IPC\_CREAT); // Creates two semaphores

int pause;

\*readerCount = 0;

psembufR.sem\_num=0; // init reader mutex members

psembufR.sem\_op=-1;

psembufR.sem\_flg=SEM\_UNDO;

vsembufR.sem\_num=0;

vsembufR.sem\_op=1;

vsembufR.sem\_flg=SEM\_UNDO;

psembufF.sem\_num=1; // resource

psembufF.sem\_op=-1;

psembufF.sem\_flg=SEM\_UNDO;

vsembufF.sem\_num=1;

vsembufF.sem\_op=1;

vsembufF.sem\_flg=SEM\_UNDO;

psembufW.sem\_num=0; // writer

psembufW.sem\_op=-1;

psembufW.sem\_flg=SEM\_UNDO;

vsembufW.sem\_num=0;

vsembufW.sem\_op=1;

vsembufW.sem\_flg=SEM\_UNDO;

psembufT.sem\_num=1; // try

psembufT.sem\_op=-1;

psembufT.sem\_flg=SEM\_UNDO;

vsembufT.sem\_num=1;

vsembufT.sem\_op=1;

vsembufT.sem\_flg=SEM\_UNDO;

arg.val = 1;

semctl(semid, 0, SETVAL, arg); // initialises semaphore 0. Do this once only

semctl(semid, 1, SETVAL, arg);

while(1){

cout << "\nReader1: \n";

pause = getchar(); // prevent errors

semop(semid, &psembufT, 1); // lock reader try mutex (trying to enter)

semop(semid, &psembufR, 1); // lock reader mutex (avoid race condition)

\*readerCount++;

if(\*readerCount == 1) // is this first reader

semop(semid, &psembufF, 1); // lock resource from writers if 1st reader

semop(semid, &vsembufR, 1); // unlock reader mutex (for other readers)

semop(semid, &vsembufT, 1); // unlock try mutex (done accessing file)

// Critical Section

myFile.open ("myFile.txt", ios::out | ios::app); // ::app appends the myFile (new line)

if(myFile.is\_open()){

while(getline(myFile, op)){

cout << op; // reads

}

myFile.close();

}

semop(semid, &psembufR, 1); // lock reader mutex (avoid race)

\*readerCount--;

if(\*readerCount == 0) // is this the last reader

semop(semid, &vsembufF, 1); // unlock resource

semop(semid, &vsembufR, 1); // unlock reader mutex

}

}