/\*

\* shm\_server : wait on a semaphore; do lookup ; use shared

\* memory for communication ; notify using same

\* semaphore (client raises by 2; server lowers

\* by 1 both to wait and to notify)

\*

\* argv[1] is the name of the local file

\* argv[2] is the key for semaphores + shared mem

\*/

#include <sys/stat.h>

#include <sys/errno.h>

#include <sys/types.h>

#include <fcntl.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/mman.h>

#include <errno.h>

#include <stdlib.h>

#include <string.h>

#include "dict.h"

int main(int argc, char \*\*argv) {

int shmid,semid;

long key;

int fd ;

Dictrec \* shm;

struct stat stbuff;

extern int errno;

unsigned short vals[2] = { 1 , 0 }; /\* Initial values of semaphores \*/

union semun { /\* Needed for semctl for setup. \*/

int val;

struct semid\_ds \*buf;

unsigned short \*array;

} setit ;

struct sembuf wait = {1,-1,SEM\_UNDO}; /\* used BOTH to wait & notify \*/

if (argc != 3) {

fprintf(stderr,"Usage : %s <dictionary source> <Resource/Key>\n",argv[0]);

exit(errno);

}

/\* Create & initialize shared memory & semaphores \*/

/\* Verify database resource is present. \*/

if (stat(argv[1],&stbuff) == -1)

DIE(argv[1]);

/\* Get key from commandline argument. \*/

key = strtol(argv[2],(char \*\*)NULL,0);

/\* Map one record's worth of shared memory.

\* The word of the sent value will be the request,

\* and the text of the result will be returned as the answer.

\*

\* Fill in code. \*/

if((shmid=shmget(key,sizeof(Dictrec),0666|IPC\_CREAT))==-1){

perror("shmget");

exit(1);

}

/\* Allocate a group of two semaphores. Use same key as for shmem.

\* Fill in code. \*/

if((semid=semget(key ,2,0666|IPC\_CREAT))==-1){

perror("semget");

exit(1);

}

/\* Get shared memory virtual address.

\* Fill in code. \*/

shm = shmat(shmid,NULL,0);

if(shm==MAP\_FAILED){

perror("shmat");

exit(1);

}

/\* Set up semaphore group. \*/

setit.array = vals;

/\* Fill in code. \*/

if(semctl(semid,0,SETALL,setit)==-1){

perror("semctl");

exit(1);

}

/\* Main working loop \*/

for (;;) {

/\* When we are first started, value is zero. Client sets to two to wake us up.

\* Try to decrement sem 1.

\* Then we will wait here until the semaphore is non-zero

\*

\* Fill in code. \*/

if(semop(semid,&wait,1)==-1){

perror("semop1");

exit(1);

}

/\* Do the lookup here. Write result directly into shared memory. \*/

switch(lookup(shm,argv[1]) ) {

case FOUND:

break;

case NOTFOUND:

strcpy(shm->text,"XXXX");

break;

case UNAVAIL:

DIE(argv[1]);

}

/\* Decrement again so that we will block at the top of the for loop again until a client wakes us up again.

\*

\* Fill in code. \*/

if(semop(semid,&wait,1)==-1){

perror("semop2");

exit(1);

}

} /\* end for \*/

}