**Assignment 1**

**William Collins**

**040652633**

**CST8217**

**Lab Section 013**

**Mitch White**

**20 Oct 2010**

**PDL**

START prog1

OPEN files for reading

CHECK arguments

CREATE key for message queue and shared memory

CREATE semaphore set

SET value in semaphore

CREATE message queue

CREATE shared memory segment

WHILE reading data from files

WRITE to a buffer

END WHILE

SEND message to message queue

LOCK resource while writing to shared memory

WRITE message to shared memory

UNLOCK resource

END prog1

START prog2

CHECK arguments

CREATE keys

GET semaphore set

GET message queue

WRITE message queue to local string

LOCK access to the shared memory

WRITE local string to shared memory

UNLOCK access to shared memory

REMOVE semaphore from the system

END prog2

START prog3

CHECK arguments

WHILE file being opened exists

CHANGE file name

END WHILE

OPEN file for writing

CREATE keys

GET semaphore set

LOCK shared memory resources

GET shared memory segments

WHILE both files have more lines to write

WRITE a line from memory 1

WRITE a line from memory 2

END WHILE

REMOVE shared memory from the system

REMOVE semaphore set

END prog3

**Data Dictionary**

|  |  |  |
| --- | --- | --- |
| Variable | Type | Purpose |
| file1, file2 | FILE \* | Holds a pointer to the input data files |
| mymsg\_data1, 2 | struct mymesg | Structure holding message and type for the message going to the message queue |
| setval | union semun | Used to initialize the value of the semaphore set |
| shm\_p1, shm\_p2 | void \* | Holds the location of each section of shared memory |
| msqid | int | The message queue ID |
| semid | int | The semaphore set ID |
| shmid | int | The shared memory ID |
| semset | struct sembuf | Used to increment or decrement semaphore |
| key1, key2 | key\_t | The keys used to create the memory,queue,and semaphores |
| output | char [] | Name of output file where the combined data will go |
| fd | int | File descriptor returned from the open() function. Will be used to convert to a pointer. |

**Test Plan**

|  |  |  |
| --- | --- | --- |
| Case | Expected results | Results |
| Files don’t exist | Error message and program exits | matches |
| message queue exists with specified key | Error message and program exits | matches |
| semaphore set exists with specified key | error message and program exits | matches |
| shared memory exists with specified key | error message and program exits | matches |
| input files are binaries not text files | same as text files | matches |
| semaphore is locked when accessing shared memory | the process will wait for the semaphore to be unlocked and then access memory | matches |
| semaphore is free when access shared memory | process will access the memory right away | matches |
| size of data files exceed program specified limit | error message and program exits | matches |
| size of data files are okay | combines the two files | matches |

**Code**

/\*\*prog1

\* Assignment 1

\* By: William Collins

\* CST8217

\* Purpose - This program will take in two data files

\* sending one to shared memory, and one to a message queue

\*/

#include "header.h"

int main(int argc, char \*argv[])

{

FILE \*file1;

FILE \*file2;

struct mymesg mymsg\_data1; /\*Data file 1\*/

struct mymesg mymsg\_data2; /\*Data file 2\*/

struct msqid\_ds msq\_ds; /\*Message queue data structure\*/

struct shmid\_ds shm\_ds; /\*Shared memory data structure\*/

union semun setval;

char \*shm\_p;

key\_t key;

int msqid;

int shmid;

int semid;

struct sembuf semset = { 0, -1, 0 };

mymsg\_data2.mtype = TYPE; /\*Set message type\*/

setval.val = 1; /\*set value of semaphore set\*/

/\*check arguments\*/

if (argc != 3)

{

printf("syntax: ./prog1 file1 file2\n");

return 1;

}

/\*open file1 and file2\*/

if ((file1 = fopen(argv[1], "r")) == NULL)

{

perror("fopen1: ");

return 1;

}

if ((file2 = fopen(argv[2], "r")) == NULL)

{

perror("fopen2: ");

return 1;

}

/\*create key from file 1\*/

if ((key = ftok(argv[1], ID)) == -1)

{

perror("ftok: ");

return 1;

}

/\*set up semaphore set\*/

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

{

perror("semget: ");

return 1;

}

/\*initialize sem set to 1\*/

if (semctl(semid, 0, SETVAL, setval) == -1)

{

perror("semctl setval: ");

return 1;

}

/\*set up the shared memory to be used\*/

if ((shmid = shmget(key, sizeof(mymsg\_data1), 0666 | IPC\_CREAT | IPC\_EXCL)) == -1)

{

perror("shmget: ");

return 1;

}

/\*set up the message queue to be used\*/

if ((msqid = msgget(key, 0666 | IPC\_CREAT | IPC\_EXCL)) == -1)

{

perror("msgget: ");

return 1;

}

/\*read in from files write to destinations\*/

wread(file1, mymsg\_data1.msg);

wread(file2, mymsg\_data2.msg);

/\*send message to message queue\*/

if (msgsnd(msqid, &mymsg\_data2, MAX\_FILE, IPC\_NOWAIT) == -1)

{

perror("msgsnd: ");

return 1;

}

/\*decrement semeaphore to control access to shared memory\*/

if (semop(semid, &semset, 1) == -1)

{

perror("semop: ");

return 1;

}

/\*attach message to shared memory then detach\*/

if ((shm\_p = shmat(shmid, NULL, 0)) == (void \*)-1)

{

perror("shmat: ");

return 1;

}

strcpy(shm\_p, mymsg\_data1.msg);

if (shmdt(shm\_p) == -1)

{

perror("shmdt: ");

}

/\*increment semaphore to allow access again\*/

semset.sem\_op = 1;

if (semop(semid, &semset, 1) == -1)

{

perror("semop: ");

return 1;

}

/\*set up shared memory structure\*/

if (shmctl(shmid, IPC\_STAT, &shm\_ds) == -1)

{

perror("shmctl: ");

return 1;

}

/\*set up message queue structure\*/

if (msgctl(msqid, IPC\_STAT, &msq\_ds) == -1)

{

perror("msgctl: ");

return 1;

}

/\*display information about shared memory and message queue\*/

/\*

printf("message queue\n\*\*\*\*\*\*\*\*\*\*\*\*\*\nmsqid: %d\nlast message sent at: %s\n",msqid, ctime(&msq\_ds.msg\_stime));

printf("shared memory\n\*\*\*\*\*\*\*\*\*\*\*\*\*\nshmid: %d\nLast attach at: %s\n",shmid, ctime(&shm\_ds.shm\_atime));

\*/

return 0;

}

void wread(FILE \*file, char \*msg)

{

int c, i= 0;

while ((c = fgetc(file)) != EOF)

{

msg[i] = c;

i++;

}

msg[i] = '\0';

}

/\*\*prog2.c

\* Assignment 1

\* CST8217

\* By: William Collins

\* Purpose: Get a message from the queue and put it in shared memory.

\*/

#include "header.h"

int main(int argc, char \*argv[])

{

struct mymesg mymsg;

char \*shm\_p;

key\_t msg\_key;

key\_t shm\_key;

int msqid;

int shmid;

int semid;

struct sembuf semset = { 0, -1, 0 };

/\*check args\*/

if (argc != 3)

{

printf("syntax: ./prog2 file1 file2\n");

return 1;

}

/\*generate key\*/

if ((msg\_key = ftok(argv[1], ID)) == -1 || (shm\_key = ftok(argv[2], ID)) == -1)

{

perror("ftok: ");

return 1;

}

/\*set up semaphore set\*/

if ((semid = semget(msg\_key, 1, 0666L)) == -1)

{

perror("semget: ");

return 1;

}

/\*access message queue\*/

if ((msqid = msgget(msg\_key, 0666)) == -1)

{

perror("msgget ");

return 1;

}

/\*put message from queue into local variable\*/

if (msgrcv(msqid, &mymsg, MAX\_FILE, TYPE, IPC\_NOWAIT) == -1)

{

perror("msgrcv: ");

return 1;

}

/\*decrement semeaphore to control access to shared memory\*/

if (semop(semid, &semset, 1) == -1)

{

perror("semop: ");

return 1;

}

/\*get shared memory address\*/

if ((shmid = shmget(shm\_key, sizeof(mymsg), 0666 | IPC\_CREAT | IPC\_EXCL)) == -1)

{

perror("shmget: ");

return 1;

}

if ((shm\_p = shmat(shmid, NULL, 0)) == (void \*) -1)

{

perror("shmat: ");

return 1;

}

/\*put message from queue into shared memory\*/

strcpy(shm\_p, mymsg.msg);

/\*remove message queue from the system\*/

if (msgctl(msqid, IPC\_RMID, NULL) == -1)

{

perror("msgctl: ");

return 1;

}

/\*increment semaphore to allow access again\*/

semset.sem\_op = 1;

if (semop(semid, &semset, 1) == -1)

{

perror("semop: ");

return 1;

}

return 0;

}

/\*\*prog3.c

\* Assignment 1

\* CST8217

\* By: William Collins

\* Purpose: Accesses two shared memory locations and puts the data together.

\*/

#include "header.h"

int main(int argc, char \*argv[])

{

FILE \*file; /\*file stream to send data to file\*/

struct sembuf semset = { 0, -1, 0 };

char \*shm\_p1; /\*pointer to shared memory1\*/

char \*shm\_p2; /\*pointer to shared memory2\*/

char output[32] = "output1";

key\_t key1;

key\_t key2;

int shmid1;

int shmid2;

int semid;

int fd; /\*file descriptor for the combined data\*/

int i;

/\*check args\*/

if (argc != 3)

{

printf("syntax: ./prog3 file1 file2");

return 1;

}

/\*open file with the name output1. if output1 exists call it output2..etc\*/

for (i = 2; ;i++)

{

if ((fd = open(output, O\_RDWR | O\_EXCL | O\_CREAT, S\_IRWXU | S\_IRGRP | S\_IROTH )) == -1)

{

if (errno == EEXIST)

{

output[strlen(output)-1] = '\0';

sprintf(output, "%s%d", output, i);

}

else

{

perror("open: ");

return 1;

}

}

else

{

break;

}

}

if ((file = fdopen(fd, "r+")) == NULL)

{

perror("fdopen: ");

return 1;

}

/\*generate keys\*/

if ((key1 = ftok(argv[1], ID)) == -1 || (key2 = ftok(argv[2], ID)) == -1)

{

perror("ftok: ");

return 1;

}

/\* get semephore set\*/

if ((semid = semget(key1, 1, 0666)) == -1)

{

perror("semget: ");

return 1;

}

/\* decrement semaphore set while memory in use\*/

if (semop(semid, &semset, 1) == -1)

{

perror("semop decrement: ");

return 1;

}

/\*access and attach shared memory 1\*/

if ((shmid1 = shmget(key1, MAX\_FILE, 0666)) == -1)

{

perror("shmget1: ");

return 1;

}

if ((shm\_p1 = shmat(shmid1, NULL, 0)) == (void \*) -1)

{

perror("shmat1: ");

return 1;

}

/\*access and attach shared memory 2\*/

if ((shmid2 = shmget(key2, MAX\_FILE, 0666)) == -1)

{

perror("shmget2: ");

return 1;

}

if ((shm\_p2 = shmat(shmid2, NULL, 0)) == (void \*) -1)

{

perror("shmat2: ");

return 1;

}

/\*send information to file\*/

while (shm\_p1[0] != '\0' || shm\_p2[0] != '\0')

{

if (shm\_p1[0] != '\0')

{

while (shm\_p1[0] != '\n')

{

fputc(shm\_p1[0], file);

shm\_p1++;

}

shm\_p1++;

fputc('\n', file);

}

if (shm\_p2[0] != '\0')

{

while (shm\_p2[0] != '\n')

{

fputc(shm\_p2[0], file);

shm\_p2++;

}

shm\_p2++;

fputc('\n', file);

}

}

/\*remove shared memory\*/

if (shmctl(shmid1, IPC\_RMID, NULL) == -1)

{

perror("shmctl1: ");

return 1;

}

if (shmctl(shmid2, IPC\_RMID, NULL) == -1)

{

perror("shmctl2: ");

return 1;

}

/\*release the semaphore\*/

semset.sem\_op = 1;

if (semop(semid, &semset, 1) == -1)

{

perror("semop release: ");

return 1;

}

/\*remove semapohore set\*/

if (semctl(semid, 0, IPC\_RMID, NULL) == -1)

{

perror("semctl remove: ");

return 1;

}

return 0;

}

**Output**

Script started on Wed Oct 20 17:54:52 2010

Willy-> ./prog1 DATA1 DATA2

Willy-> ./prog2 DATA1 DATA2

Willy-> ./prog3 DATA1 DATA2

Willy-> cat output1

THE I AM CANADIAN RANT

Hey,

I'm not a lumberjack, or a Furtrader,

and I don't live in an igloo,

or eat blubber

or own a dogsled.

And I don't know Jimmy, Sally or Suzy from Canada,

although I'm certain they're really, really nice.

I have a prime minister... not a president,

I speak English and French, not American

and I pronounce it About, not A-boot.

I can proudly sew my country's flag on my backpack,

I believe in peacekeeping, not policing,

diversity not assimilation,

and that the beaver is a truly proud and noble animal.

A toque is a hat,

a chesterfield is a couch,

and it IS pronounced Zed,

not Zee... ZED!!

Canada is the 2nd largest land mass,

the 1st nation of hockey,

and the best part of North America.

My name is Joe...

and...I......AM......CANADIAN!

Thank you.

Willy-> clear

Willy-> exit

exit

Script done on Wed Oct 20 17:55:27 2010