CIS 452
Lab 5 Report

Ashley Hendrickson
Muna Gigowski
Fall 2019

Shared Memory

Question One

What exactly is being output by Sample Program 1 (i.e. what is the meaning of the output values)?

OUTPUT:

value a: 0x7f566eeef000 value b: 0x7f566eef0000

Value ‘a’ is the address that the shared memory pointer is pointing to (which is the address of the attached shared memory segment)

Value ‘b’ is the memory address of the attached shared memory segment plus 4096, which was the specified size of the shared memory segment when it was first created with shmget earlier in the program

Question Two

Read the man pages; then describe the meaning / purpose of each argument used by the `shmget()` function call.

Key - The first value of shmget is the key value. The key value is compared to existing values that exist within the kernel for other shared memory segments.

Size - Shared memory segment size

Shmflg - The shmflg is a combination of operation permissions and control commands. After determining the value for the operation permissions, the desired flags can be specified. if the shflag specifies both IPC_CREAT and IPC_EXCL and a shared segment already exists for the key then shmget will fail with errno set to 0_EXCL

Question Three

Describe two specific uses of the `shmctl()` function call

Shmctl() is used for many purposes relating to controlling the resource that the kernel created. One use for shmctl() is that it can be used to mark the segment to be destroyed, after the last process detaches it, which is specified by using IPC_RMID for the cmd argument in the function call. Shmctl() can also be used to prevent swapping of the shared memory segment by locking it down, which is specified by using SHM_LOCK for the cmd argument in the function call.

Similarly, you can also unlock the segment and allow it to again be swapped out by using SHM_UNLOCK cmd**.**

Useful System Utilities

Question Four

Read the man pages, then use `shmctl()` to modify Sample Program 1 so that it prints out the size of the shared memory segment.

SOURCE CODE FOR REVISED SAMPLEPROGRAM1.C:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <signal.h>
#include <unistd.h>
#define FOO 4096

void sigHandler(int sigNum);
int main ()
{
   int shmId;
   char *shmPtr;
   struct shmid_ds sharedMem;


   if ((shmId = shmget (IPC_PRIVATE, FOO, IPC_CREAT|S_IRUSR|S_IWUSR)) < 0) {
      perror ("i can't get no..\n");
      exit (1);
   }

   shmctl(shmId,IPC_STAT, &sharedMem);
   printf("The size of the shared memory segment is %lu\n
",sharedMem.shm_segsz);

   if ((shmPtr = shmat (shmId, 0, 0)) == (void*) -1) {
      perror ("can't attach\n");
      exit (1);
   }

   //modify the print statement in Sample Program 1 to determine the ID of
the shared memory segment
   printf ("shared memory Id is %d\n", shmId);
   printf ("value a: %p\t value b: %p\n", (void *) shmPtr, (void *) shmPtr +
FOO);
   pause();
   if (shmdt (shmPtr) < 0) {
```

```
      perror ("just can't let go\n");
      exit (1);
   }
   if (shmctl (shmId, IPC_RMID, 0) < 0) {
      perror ("can't deallocate\n");
      exit(1);
   }

   return 0;
}
void sigHandler(int sigNum){
       printf("Exiting good by");
}
```

Question Five

Submit your script (Take a screenshot of commands).

Lab Programming Assignment (Readers and Writer)

Source Code

WRITER PROGRAM CODE:

```
#include <iostream>
#include <thread>
#include <pthread.h>
#include <string>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
#include<unistd.h>
#include <signal.h>
#include "DataSet.h"
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <unistd.h>
using namespace std;

const int shared_segment_size = sizeof(Dataset);

void my_handler(int num);
Dataset * shmptr;
int main(){
```

```
        Dataset * sharedMemory;
        int shmid;
        key_t key;
        //set up sigHandler to receive ^C signal and call custom signal
handler function
        struct sigaction sigIntHandler;
        sigIntHandler.sa_handler = my_handler;
        sigemptyset(&sigIntHandler.sa_mask);
        sigIntHandler.sa_flags = 0;
        sigaction(SIGINT, &sigIntHandler, NULL);

        // ftok to generate unique key
        if((key=ftok("./",1))<1){
                perror("Failed to assign shmid");
                exit(1);
        }

        // shmget returns an identifier in shmid
        shmid = shmget(key,shared_segment_size,IPC_CREAT | 0600);
        if(shmid < 1){
                perror("Failed to assign shmid");
        }

        //Attach struct to shared memory
        sharedMemory = (Dataset* ) shmat (shmid,NULL, 0);

        // sets sharedMemorys values

        sharedMemory->shmid=shmid;
        sharedMemory->writerTurn=true;
        sharedMemory->n=0;
        sharedMemory-> numTimesRead=0;
        memset(sharedMemory->userInput, '\000', sizeof(sharedMemory-
>userInput));

        shmptr=sharedMemory;
        if(sharedMemory==(Dataset*)-1){
        perror("shmat failed ");
        exit(1);
        }
        // shmat to attach to shared memory
        //char *str = (char*) shmat(shmid,(void*)0,0);
cout<< "printing the shmid:"<<shmid << "\n";

        while(1) {
                if(sharedMemory->writerTurn) {
                        cout << "Please provide data to be written into shared
memory: ";
                        cin >> sharedMemory->userInput;
                        printf("Data written into memory: %s\n",sharedMemory-
>userInput);
                        sharedMemory->writerTurn = false;
                }
        }
}

//When user enters ^C, print final stats before exiting the program
```

```
void my_handler(int shmid) {
        shmid=shmptr->shmid;
        //detach from shared memory
        if(shmdt(shmptr)==-1){
        perror("failed to detach");
        }

        if(shmctl(shmid,IPC_RMID,NULL)==-1){
        perror("failed to remove shared memory");
        }
cout<< "exiting writer "<<endl;
exit(0);
}
```

READER PROGRAM CODE:

```
#include<iostream>
#include<thread>
#include <pthread.h>
#include <string>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <errno.h>
#include<unistd.h>
#include <signal.h>
#include <sys/stat.h>
#include "DataSet.h"

using namespace std;

const int shared_segment_size = sizeof(Dataset);

void my_handler(int shmid);
Dataset* shmptr;
int main(){

        Dataset* sharedMemory;

//Dataset* sharedMemory = &sharedMem;

        int shmid;
        key_t key;
        //set up sigHandler to receive ^C signal and call custom signal
handler function
        //Fancy Singal handler
        struct sigaction sigIntHandler;
        sigIntHandler.sa_handler = my_handler;
        /*
         * The sigemptyset() function is part of a family of functions that
manipulate signal sets. Signal sets are data objects that let
         * a thread keep track of groups of signals. For example, a thread
might create a signal set to record which signals it is
         * blocking, and another signal set to record which signals are
pending. */
```

```
        sigemptyset(&sigIntHandler.sa_mask);

        sigIntHandler.sa_flags = 0;

        sigaction(SIGINT, &sigIntHandler,NULL);

        // ftok to generate unique key
        signal(SIGSEGV,my_handler);
        if((key = ftok(".",1))<1){
                perror("IPC error: ftok");
                exit(1);
        }

        // shmget returns an identifier in shmid
        shmid = shmget(key,shared_segment_size, S_IRUSR|S_IWUSR);
        if(shmid < 1){
                perror("Failed to assign shmid");
                exit(1);
        }
        cout<< "printing the shmid:"<<shmid << endl;
        //Attach struct to shared memory
        sharedMemory = (Dataset*) shmat(shmid, NULL, 0);


        shmptr=sharedMemory;
//      if(sharedMemory->n==0)
                sharedMemory->writerTurn=0;

        bool myTurn = true;
        sharedMemory->n++;
        while(1) {

//              cout<<"Shared Memory "<<sharedMemory->userInput<<endl;
                // last one out shut the lights. This check for the last one
out
                if(sharedMemory->n ==sharedMemory->numTimesRead){
                        sharedMemory->numTimesRead = 0;
                        sharedMemory->writerTurn = true;
                }

                //provents a process from entering the critical section once
it has entered once
                if(sharedMemory->writerTurn==true){
                        myTurn = true;

                }

                // only print out onces
                if(sharedMemory->writerTurn==false && myTurn) {
                        myTurn=false;
                        sharedMemory->numTimesRead=sharedMemory-
>numTimesRead+1;
                        cout<<"Shared Memory "<<sharedMemory->userInput<<endl;
                        //printf("Other side: %s\n", sharedMemory->userInput);

                }
        }
```

```
}


//When user enters ^C, print final stats before exiting the program
void my_handler(int shmid) {
        cout<<"Exsiting reader"<<endl;
        //notify everyone your leaving
        shmptr->n--;

        shmid=shmptr->shmid;
        //detach from shared memory
        shmdt(shmptr);
        // destroy the shared memory
        exit(0);
}
```

Sample Output