William Shreeve
Matt Kennedy
CIS452 Lab 5

1. The two values being printed are shmPtr and shmPtr + FOO. shmPtr is a pointer being used to define the address of the shared memory, and FOO is 4096, which was declared to be the size of the shared memory space. Therefore, shmPtr is the address of the shared memory space, which is called ‘value a’, and shmPtr + FOO is the end address of the shared memory space, which is called ‘value b’.
2. From the man pages: The shmget() function shall return the shared memory identifier associated with key. To expand on this, there are 3 arguments the function takes. First is the key. In this example, IPC_PRIVATE means that the shared memory is only accessible to children of the process. The next argument is the size, which is just the requested size of the shared memory segment. Lastly is the flags. Here, IPC_CREAT says to create the segment if it doesn’t already exist in the kernel. The next two, S_IRUSR and S_IWUSR, give read and write permission, respectively, to the owner of the segment.
3. shmctl() performs the control operation specificed by the second parameter on the shared memory segment whose ID is given as the first parameter.
    a. IPC_STAT is one of the commands. This copies the information from the shared memory segment into a structure, which is defined by the third parameter, which takes a struct of shmid_ds type.
        i. Note: The shmid_ds type is a struct for shared memory segments.
    b. IPC_RMID marks the shared memory to be destroyed. The segment will only be destroyed after the last process detaches it, which is told by the shm_nattch value of the shmid_ds struct. When shm_nattch is equal to zero, the segment will be destroyed. (The third parameter, *buf, will be ignored in this case.)

```
struct shmid_ds buf;
if (shmctl (shmId, IPC_RMID | IPC_STAT, &buf) < 0) {
        perror ("can't deallocate\n");
        exit(1);
}
printf("Size is %zu\n", buf.shm_segsz);
return 0;
}
```

4.

Script started on 2019-02-14 08:27:24-05:00 [TERM="xterm" TTY="/dev/pts/0" COLUMNS="96" LINES="52"]
]0;shreevew@eos10:~/CIS452-Labs/Lab5[shreevew@eos10 Lab5]$ ./a.out
value a: 0x7f3c99694000 value b: 0x7f3c99695000
ID: **66519062**
^C

]0;shreevew@eos10:~/CIS452-Labs/Lab5[shreevew@eos10 Lab5]$ ipcs

------ Message Queues --------
key msqid owner perms used-bytes messages

------ Shared Memory Segments --------
key shmid owner perms bytes nattch status
0x00000000 64126976 shreevew 600 524288 2 dest
0x00000000 64028673 shreevew 600 524288 2 dest
0x00000000 37453826 deleonl 600 524288 2 dest
0x00000000 38109187 deleonl 600 4194304 2 dest
0x00000000 37748740 deleonl 600 524288 2 dest
0x00000000 37650437 deleonl 600 524288 2 dest
0x00000000 37879814 deleonl 600 524288 2 dest
0x00000000 37978119 deleonl 600 524288 2 dest
0x00000000 38010888 deleonl 600 33554432 2 dest
0x00000000 38273033 deleonl 600 2097152 2 dest
0x00000000 38174730 deleonl 600 33554432 2 dest
0x00000000 64225291 shreevew 600 524288 2 dest
0x00000000 64323596 shreevew 600 524288 2 dest
0x00000000 64520205 shreevew 600 524288 2 dest
0x00000000 64618510 shreevew 600 4194304 2 dest
0x00000000 64782351 shreevew 600 524288 2 dest
0x00000000 64684048 shreevew 600 67108864 2 dest
0x00000000 64815121 shreevew 600 1048576 2 dest
0x00000000 64946194 shreevew 600 1048576 2 dest
0x00000000 64880659 shreevew 600 2572288 2 dest
0x00000000 64978964 shreevew 600 2572288 2 dest
0x00000000 65798165 shreevew 600 12288 2 dest
0x00000000 66519062 shreevew 600 4096 0
0x00000000 65437719 shreevew 600 1167360 2 dest
0x00000000 65536024 shreevew 600 1167360 2 dest
0x00000000 65699865 shreevew 600 36864 2 dest
0x00000000 65732634 shreevew 600 36864 2 dest
0x00000000 65830939 shreevew 600 12288 2 dest
0x00000000 66486301 shreevew 600 524288 2 dest
0x00000000 66289695 shreevew 600 106496 2 dest
0x00000000 66322464 shreevew 600 106496 2 dest

```
------ Semaphore Arrays --------
key     semid           owner           perms           nsems

]0;shreevew@eos10:~/CIS452-Labs/Lab5[shreevew@eos10 Lab5]$ ipcrm -m
66519062
]0;shreevew@eos10:~/CIS452-Labs/Lab5[shreevew@eos10 Lab5]$ ipcs

------ Message Queues --------
key     msqid           owner           perms           used-bytes   messages

------ Shared Memory Segments --------
key     shmid           owner           perms           bytes  nattch          status
0x00000000 64126976   shreevew   600          524288         2         dest
0x00000000 64028673   shreevew   600          524288         2         dest
0x00000000 37453826   deleonl   600     524288         2         dest
0x00000000 38109187   deleonl   600     4194304        2         dest
0x00000000 37748740   deleonl   600     524288         2         dest
0x00000000 37650437   deleonl   600     524288         2         dest
0x00000000 37879814   deleonl   600     524288         2         dest
0x00000000 37978119   deleonl   600     524288         2         dest
0x00000000 38010888   deleonl   600     33554432   2          dest
0x00000000 38273033   deleonl   600     2097152        2         dest
0x00000000 38174730   deleonl   600     33554432   2          dest
0x00000000 64225291   shreevew   600          524288         2         dest
0x00000000 64323596   shreevew   600          524288         2         dest
0x00000000 64520205   shreevew   600          524288         2         dest
0x00000000 64618510   shreevew   600          4194304        2         dest
0x00000000 64782351   shreevew   600          524288         2         dest
0x00000000 64684048   shreevew   600          67108864   2          dest
0x00000000 64815121   shreevew   600          1048576        2         dest
0x00000000 64946194   shreevew   600          1048576        2         dest
0x00000000 64880659   shreevew   600          2572288        2         dest
0x00000000 64978964   shreevew   600          2572288        2         dest
0x00000000 65798165   shreevew   600          12288          2         dest
0x00000000 65437719   shreevew   600          1167360        2         dest
0x00000000 65536024   shreevew   600          1167360        2         dest
0x00000000 65699865   shreevew   600          36864          2         dest
0x00000000 65732634   shreevew   600          36864          2         dest
0x00000000 65830939   shreevew   600          12288          2         dest
```

```
0x00000000 66486301   shreevew   600        524288     2          dest
0x00000000 66289695   shreevew   600        106496     2          dest
0x00000000 66322464   shreevew   600        106496     2          dest

------ Semaphore Arrays --------
key     semid      owner      perms      nsems

]0;shreevew@eos10:~/CIS452-Labs/Lab5[shreevew@eos10 Lab5]$ exit
exit

Script done on 2019-02-14 08:27:50-05:00 [COMMAND_EXIT_CODE="0"]
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>

#define FOO 4096
#define SIZE 1024

int main ()
{
        int count, shmId, loopFlag = -1;
        char *shmPtr = (char*)malloc(SIZE), *data = (char*)malloc(SIZE);


        key_t key = ftok("shmfile",65);
        if ((shmId = shmget (key, FOO, IPC_CREAT|S_IRUSR|S_IWUSR)) < 0)
        {
                perror("could not get");
                exit(1);
        }
        if ((shmPtr = shmat (shmId, 0, 0)) == (void*) -1) {
                perror ("can't attach\n");
                exit (1);
        }

        while (loopFlag)
        {
```

```
30          while (loopFlag)
31          {
32                  //wait for value to be available
33                  //wait for lock to be 0
34                  //set lock to 0
35                  //read value
36                  //increment count
37                  //set lock to 1
38                  //while word hasn't changed, sleep
39                  while (shmPtr[SIZE] == 0);
40
41                  while (strcmp(data, shmPtr) == 0)
42                          strcpy(data, shmPtr);
43
44                  shmPtr[SIZE] = 1;
45                  count = (int)*(shmPtr + SIZE + sizeof(int)) + 1;
46
47                  shmPtr[SIZE + sizeof(int)] = count;
48                  shmPtr[SIZE] = 0;
49                  strcpy(data, shmPtr);
50                  printf("word is %s\n", data);
51          }
52  //      printf("What the heck goes here %s\n", shmPtr + SIZE);
53
54          //      }
55
56          printf ("value a: %p\t value b: %p\n", (void *) shmPtr, (void *) shmPtr + FOO);
57          if (shmdt (shmPtr) < 0) {
58
59                  perror ("just can't let go\n");
60                  exit (1);
61          }
62          if (shmctl (shmId, IPC_RMID, 0) < 0) {
63                  perror ("can't deallocate\n");
64                  exit(1);
65          }
66          free(data);
67          return 0;
68  }
```

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <signal.h>

#define FOO 4096
#define SIZE 1024

char* shmPtr, *data;
int shmId;
void sigHandler(int);
int main ()
{
        int loopFlag = 1;
        data = (char*)malloc(SIZE);
        signal(SIGINT, sigHandler);

        key_t key = ftok("shmfile",65);
        printf("made it past here\n");
        if ((shmId = shmget (key, FOO, IPC_CREAT|S_IRUSR|S_IWUSR)) < 0) {
                perror ("i can't get no..\n");
                exit (1);
        }
        if ((shmPtr = shmat (shmId, 0, 0)) == (void*) -1) {
                perror ("can't attach\n");
                exit (1);
        }

        printf ("value a: %p\t value b: %p, %d\n", (void *) shmPtr, (void *) shmPtr + FOO, shmId);
```

```
35          while (loopFlag)
36          {
37                  printf ("Please input your word: ");
38                  scanf ("%s", data);
39                  strcpy(shmPtr, data);
40                  shmPtr[SIZE] = '1'; // lock: 1 - available, 0 - not
41                  shmPtr[SIZE + sizeof(int)] = 0; // how many readers have accessed data
42                  printf("Word is in shared memory. Please wait . . .\n");
43                  while (shmPtr[SIZE + sizeof(int)] != 2);
44                  shmPtr[SIZE + sizeof(int)] = 0;
45          }
46
47          if (shmdt (shmPtr) < 0) {
48
49                  perror ("just can't let go\n");
50                  exit (1);
51          }
52          if (shmctl (shmId, IPC_RMID, 0) < 0) {
53                  perror ("can't deallocate\n");
54                  exit(1);
55          }
56          free(data);
57          return 0;
58  }
59
60  void sigHandler(int sigNum)
61  {
62          printf("\n\nUser exiting.\n");
63          if (shmdt (shmPtr) < 0) {
64
65                  exit (1);
66          }
67          if (shmctl (shmId, IPC_RMID, 0) < 0) {
68                  exit(1);
69          }
70          free(data);
71          exit(0);
72  }
```