CE4206 Assignment 4

Conor Egan
13138782
22/April/2016

Requirements:

Complete three separate exercises demonstrating UNIX memory concepts. Each exercise requires two programs to be created.

The first exercise will create a shared memory object and will write a simple string message into that object. Another program memReader.c will open the memory object and will read the message from that object.

The second exercise is similar to the first, it requires the transfer of an array of data instead of a simple string message using a shared memory object.

The third exercise requires the creation and modification of a small memory mapped file.

Description of Solution:

Exercise 1:
Using write() instead of memcpy required me to replace the parameter “shared_msg” with “mfd”, since the write() function takes a file descriptor as it's first argument where memcpy has a memory area reference as it's first argument. This was all that was required to complete memWriter1.c.
For memReader1.c, reporting on the page size of the SHM simply required printing the value found using the sizeof() function.
A write command identical to that used in memWriter1.c was inserted to cause a deliberate error (since file is read-only).

Exercise 2:
To create an array of 10000 bytes I first declared the array and the used memset() to fill it with 10000 'A's. I then used memcpy() to copy this array to the SHM shared_msg.
I set the SHM object size to 20000 pages, taking a “better safe than sorry” approach in allocating more than enough memory. In a system where resources are less available this number would be reduced. This was all that was required for memWriter2.c.

To allow memReader2.c to read this SHM, I updated it's object size to 20000. I then used a for loop in concert with the putchar() function to display the first five and last three values of the array.

Exercise 3:
The program mappedFile.c was used as a base for most of this exercise. To modify it for use with exercise 3, I edited the open() function and the mmap function to allow read and write access to the SHM. To modify the contents of the file, I used sscanf() to access the first character in the file and then used memcpy() to modify it. I then used values returned from the fstat() function to print the file size and the i-node number. This completed memWriter3.c.
I also used mappedFile.c as a base for memReader3.c, however I opened the file as read-only. To print the contents of the file, I used a for loop in conjunction with putchar().

Testing and Results:

memWriter1.c output:

```
conor@conor-Latitude-E6410:~$ gcc memWriter1.c -o memWriter1 -lrt
conor@conor-Latitude-E6410:~$ ./memWriter1
I have created a lucky shared memory object:  don1337
Shared memory is now allocated 100 bytes
Message content is: University of Limerick

 Hit any key to finish!
```

memReader1.c output (no deliberate error):

```
conor@conor-Latitude-E6410:~$ gcc memReader1.c -o memReader1 -lrt
conor@conor-Latitude-E6410:~$ ./memReader1
Opened the shared memory object (read only): don1337
Shared memory size allocated is 100 bytes
The message content actually read is: .... University of Limerick
The page size for this system is 4096 bytes.
conor@conor-Latitude-E6410:~$
```

memReader1.c output(with deliberate error):

```
conor@conor-Latitude-E6410:~$ ./memReader1
Opened the shared memory object (read only): don1337
Segmentation fault (core dumped)
conor@conor-Latitude-E6410:~$
```

memWriter2.c output:

```
conor@conor-Latitude-E6410: ~
conor@conor-Latitude-E6410:~$ gcc memWriter2.c -o memWriter2 -lrt
conor@conor-Latitude-E6410:~$ ./memWriter2
I have created a lucky shared memory object:  don1337
Shared memory is now allocated 20000 bytes
Hit any key to finish!
```

memReader2.c output:

```
conor@conor-Latitude-E6410:~$ gcc memReader2.c -o memReader2 -lrt
conor@conor-Latitude-E6410:~$ ./memReader2
Opened the shared memory object (read only): don1337
First five characters in array:
A
A
A
A
A
Last three characters in array:
A
A
A
conor@conor-Latitude-E6410:~$
```

memWriter3.c output:

```
conor@conor-Latitude-E6410: ~
conor@conor-Latitude-E6410:~$ gcc memWriter3.c -o memWriter3 -lrt
conor@conor-Latitude-E6410:~$ ./memWriter3 testFile.txt
File size in bytes is: 32
Contents of file:
This is a test for memWriter3.c
First Character in File: T
Character: X
Xhis is a test for memWriter3.c

File information report on: testFile.txt
=====================================
File size:                32 bytes
The inode:                3811328
conor@conor-Latitude-E6410:~$
```

memReader3.c output:

```
conor@conor-Latitude-E6410: ~
conor@conor-Latitude-E6410:~$ gcc memReader3.c -o memReader3 -lrt
conor@conor-Latitude-E6410:~$ ./memReader3 testFile.txt
File size in bytes is: 32
Contents of file:
 Xhis is a test for memWriter3.c
conor@conor-Latitude-E6410:~$
```

I created the file testFile.txt which contained basic text for exercise 3.

Statement of Completion:

All programs worked as required.

Question:

I agree with the statement that using a memory mapped file is orders of magnitude faster than accessing the file system for IPC.
Using gettimeofday() to find the program execution time for memWriter3.c and memReader3.c resulted in an execution time of 107 microseconds for memWriter3.c versus 65 microseconds for memReader3.c. This supports the above view.

Source Code:

memWriter1.c

```
/*********************************************************************
Program: memWriter1.c

A simple example on shared memory.
Create/open a shared memory object, map to that object and write to it. Unlink object.
Uses write() instead of memcpy() to write to the shared memory object

Donal Heffernan 6/November/2014     Updated 22/October/2015
Edited by Conor Egan (13138782)     18 April 2016
*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

#define SHARED_OBJ_PATH  "don1337"  // pathname to shared object
#define MESSAGE_SIZE      100     // maximum length the message

char message[MESSAGE_SIZE];

int main() {

int mfd; //file descriptor for the shared object
int seg_size = (sizeof(message));   //shared object sized to store message
char *shared_msg;

// create the shared memory object with shm_open()
mfd = shm_open(SHARED_OBJ_PATH, O_CREAT | O_RDWR, S_IRWXU | S_IRWXG);
    if (mfd < 0) {
        perror("error in shm_open()");
        exit(1);
    }

printf("I have created a lucky shared memory object:  %s\n", SHARED_OBJ_PATH);

ftruncate(mfd, seg_size) ; // define size of shared memory object

// map the shared memory object to this process
shared_msg = mmap(NULL, seg_size, PROT_READ | PROT_WRITE, MAP_SHARED, mfd, 0);
    if (shared_msg == NULL) {
        perror("error in mmap()");
        exit(1);
    }

printf("Shared memory is now allocated %d bytes\n", seg_size);
```

```
write(mfd, "University of Limerick", 30); //put something onto the memory

printf("Message content is: %s\n\n Hit any key to finish!\n", shared_msg) ;
getchar() ; // wait for user to hit any key

shm_unlink(SHARED_OBJ_PATH) ;  // unlink - better to add error check

return 0 ;
}
```

memReader1.c

```
/*************************************************************************
Program: memReader1.c

Open a shared a shared memory object, map to that object and read it.

Donal Heffernan 6/November/2014   Updated 22/October/2015
Edited by Conor Egan 22 April
*************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

#define SHARED_OBJ_PATH   "don1337"   // pathname to shared object
#define MESSAGE_SIZE      100  // maximum length the message

char message[MESSAGE_SIZE];

int main() {

int mfd;  //file descriptor for the shared object
int seg_size = (sizeof(message));   //shared object sized to store message
char *shared_msg;

// open the shared memory object for reading  only
mfd = shm_open(SHARED_OBJ_PATH, O_RDONLY, S_IRWXU | S_IRWXG);
   if (mfd < 0) {
      perror("error shm_open(): forgot to run memWriter?");
      exit(1) ;
   }

printf("Opened the shared memory object (read only): %s\n", SHARED_OBJ_PATH);

// map the shared memory object to this process
   shared_msg = mmap(NULL, seg_size, PROT_READ, MAP_SHARED, mfd, 0);
   if (shared_msg == NULL) {
      perror("error in  mmap()") ;
```

```
        exit(1);
    }
memcpy(shared_msg, "This should cause an error", 30); //should cause an error, this program has
opened memory object as read-only

printf("Shared memory size allocated is %d bytes\n", seg_size);
printf("The message content actually read is: .... %s\n", shared_msg);
printf("The page size for this system is %ld bytes.\n", sysconf(_SC_PAGESIZE));



return 0;
}
```

memWriter2.c

```
/*********************************************************************
Program: memWriter2.c

A simple example on shared memory.
Create/open a shared memory object, map to that object and write to it. Unlink object.

Donal Heffernan 6/November/2014     Updated 22/October/2015
Edited by Conor Egan 22 April
*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

#define SHARED_OBJ_PATH  "don1337"  // pathname to shared object
#define MESSAGE_SIZE      20000    // maximum length the message

char message[MESSAGE_SIZE];

int main() {

int mfd; //file descriptor for the shared objects
int seg_size = (sizeof(message));   //shared object sized to store message
char *shared_msg;

char array[10000];

//fill array with 10000 A's
memset(array, 'A', sizeof(array));


// create the shared memory object with shm_open()
mfd = shm_open(SHARED_OBJ_PATH, O_CREAT | O_RDWR, S_IRWXU | S_IRWXG);
```

```
    if (mfd < 0) {
        perror("error in shm_open()");
        exit(1);
    }

printf("I have created a lucky shared memory object:  %s\n", SHARED_OBJ_PATH);

ftruncate(mfd, seg_size) ; // define size of shared memory object

// map the shared memory object to this process
shared_msg = mmap(NULL, seg_size, PROT_READ | PROT_WRITE, MAP_SHARED, mfd, 0);
    if (shared_msg == NULL) {
        perror("error in mmap()");
        exit(1);
    }

printf("Shared memory is now allocated %d bytes\n", seg_size);

memcpy(shared_msg, array, 10000); //put something onto the memory

printf("Hit any key to finish!\n") ;
getchar() ; // wait for user to hit any key

shm_unlink(SHARED_OBJ_PATH) ;  // unlink - better to add error check

return 0 ;
}
```

memReader2.c

```
/*************************************************************************
Program: memReader2.c

Open a shared a shared memory object, map to that object and read it.

Donal Heffernan 6/November/2014   Updated 22/October/2015
Edited by Conor Egan 22 April
*************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

#define SHARED_OBJ_PATH   "don1337"   // pathname to shared object
#define MESSAGE_SIZE      20000  // maximum length the message

char message[MESSAGE_SIZE];

int main() {
```

```
int mfd;  //file descriptor for the shared object
int seg_size = (sizeof(message));   //shared object sized to store message
char *shared_msg;
off_t  len;

// open the shared memory object for reading  only
mfd = shm_open(SHARED_OBJ_PATH, O_RDONLY, S_IRWXU | S_IRWXG);
   if (mfd < 0) {
       perror("error shm_open(): forgot to run memWriter?");
       exit(1) ;
   }

printf("Opened the shared memory object (read only): %s\n", SHARED_OBJ_PATH);

// map the shared memory object to this process
   shared_msg = mmap(NULL, seg_size, PROT_READ, MAP_SHARED, mfd, 0);
   if (shared_msg == NULL) {
       perror("error in  mmap()") ;
       exit(1);
   }

// print out the contents of memory that represents the file contents
printf("First five characters in array:\n");
for (len = 0; len < 5; len++){
           putchar (shared_msg[len]);
       printf("\n");
}
printf("Last three characters in array:\n");
for (len = (sizeof(shared_msg) - 3); len < sizeof(shared_msg); len++){
           putchar (shared_msg[len]);
       printf("\n");
}


return 0;
}
```

memWriter3.c

```
/*********************************************************************
Program: memWriter3.c

A simple example on shared memory.
Create/open a shared memory object, map to that object and write to it. Unlink object.

Donal Heffernan 6/November/2014     Updated 22/October/2015
Edited by Conor Egan 22 April
*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
```

```
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/time.h>

#define SHARED_OBJ_PATH  "don1337"  // pathname to shared object
#define MESSAGE_SIZE      100     // maximum length the message

int main (int argc, char *argv[])
{
        struct stat st;
        off_t  len;
        char *pt;
        int fd;
           char character;
           unsigned long t1;
           unsigned long t2;
           unsigned long t;
           unsigned long iterations = 0;
           struct timeval startVal;
           struct timeval endVal;

           gettimeofday(&startVal, NULL);

        fd = open (argv[1], O_RDWR); // open file for reading

        fstat (fd, &st); // get stat info to 'st' structure
        printf("File size in bytes is: %lu\n", st.st_size); // print file's size

        // map file to memory - full length of the file

           pt = mmap(NULL, 100, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
        // print out the contents of memory that represents the file contents
           printf("Contents of file:\n");
        for (len = 0; len < st.st_size; len++)
                putchar (pt[len]);

           //Modify the first character of the file
           sscanf (pt, "%c", &character);
        printf ("First Character in File: %c\n", character);

           character = 'X';
        printf("Character: %c\n", character);

           memcpy(&pt[0], &character, 1);


           // print out the contents of memory that represents the file contents
        for (len = 0; len < st.st_size; len++)
                putchar (pt[len]);
```

```
        close(fd); // close file descriptor

        printf("\nFile information report on: %s\n",argv[1]);
        printf("==================================\n");
        printf("File size: \t\t%lu bytes\n",st.st_size);

        printf("The inode: \t\t%lu\n",st.st_ino);

        gettimeofday(&endVal, NULL);

        //subtract loop start time from loop end time
        t1 = startVal.tv_sec * 1000000 + startVal.tv_usec;
        t2 = endVal.tv_sec * 1000000  + endVal.tv_usec;
        t = t2 -t1;

        printf("Execution time: %lu microseconds\n", t);

        // the file will automatically unmap on program exit.

    return 0;
}
```

memReader3.c

```
/*********************************************************************
Program: memReader3.c

Open a shared a shared memory object, map to that object and read it.

Donal Heffernan 6/November/2014   Updated 22/October/2015
Edited by Conor Egan 22 April
*********************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/time.h>

#define SHARED_OBJ_PATH  "don1337"  // pathname to shared object
#define MESSAGE_SIZE      100     // maximum length the message

int main (int argc, char *argv[])
{
    struct stat st;
    off_t  len;
    char *pt;
    int fd;
        char character;
```

```
    unsigned long t1;
    unsigned long t2;
    unsigned long t;
    unsigned long iterations = 0;
    struct timeval startVal;
    struct timeval endVal;

    gettimeofday(&startVal, NULL);

  fd = open (argv[1], O_RDONLY); // open file for reading

  fstat (fd, &st); // get stat info to 'st' structure
  printf("File size in bytes is: %lu\n", st.st_size); // print file's size

  // map file to memory - full length of the file

    pt = mmap(NULL, 20000, PROT_READ, MAP_SHARED, fd, 0);
  // print out the contents of memory that represents the file contents
  printf("Contents of file:\n ");
    for (len = 0; len < st.st_size; len++)
        putchar (pt[len]);


    close(fd); // close file descriptor

    // the file will automatically unmap on program exit.

    gettimeofday(&endVal, NULL);

    //subtract loop start time from loop end time
    t1 = startVal.tv_sec * 1000000 + startVal.tv_usec;
    t2 = endVal.tv_sec * 1000000  + endVal.tv_usec;
    t = t2 -t1;

    printf("Execution time: %lu microseconds\n", t);

  return 0;
}
```