Lab 8b/9a/9b – Memory Mapped files

Operating Systems CS SH3 Term 2, Winter 2018

Dr. Neerja Mhaskar

Labs that are not scheduled for a Lab Test are not mandatory. These are practice labs, designed to help you on your assignments.

Lab Format: The practice labs will be posted a day before or on the day of the lab on the course website. You can choose to solve it beforehand and come in with your solutions and check the correctness of your solution with your TA.

The TAs will also be available to answer any questions you might have on your assignments.

Solutions to practice labs will not be posted online.

Outline

In this lab you are going to study Memory Mapping files in C.

https://www.safaribooksonline.com/library/view/linux-system-programming/0596009585/ch04s03.html - has a good explanation for memory mapped files and the mmap() system call.

You are to write a C program that opens a binary file named **numbers.bin** and maps it to a memory region using the system call **`mmap()`**.The 'numbers.bin' can be downloaded from Avenue -> Content -> Practice labs. This file contains **ten** 4-byte integers in binary format.

After memory mapping this file you are to read the contents from this memory mapped region; that is, read one integer at a time, and copy it to an integer array (names `intArray`) using the **`memcpy()`** function. Finally you are to loop through the `intArray` array to add all the numbers of the array and output the sum on the console.

In particular your program needs to do the following:

1. To be able to use mmap() system call and the memcpy() function you need to add the below header files in addition to your standard input output header file.

```
#include <sys/mman.h> /*For mmap() function*/
#include <string.h> /*For memcpy function*/
```

2. Other useful header files are listed below. They enable you to use the **`open()`** system call which is used to open a new file and obtain its file descriptor.

```
#include <fcntl.h> /*For file descriptors*/
```

```
#include <stdlib.h> /*For file descriptors*/
```

3. Define global variables for the integer array and a signed character pointer to store the starting address of the memory mapped file. E.g.:

```
int intArray[MEMORY_SIZE];
signed char *mmapfptr;
```

You can define 'MEMORY_SIZE' as a macro definition. It is the total number of bytes you will be copying from numbers.bin file. Sample code:

```
#define INT_SIZE 4 // Size of integer in bytes
#define INT_COUNT 10
#define MEMORY_SIZE INT_COUNT * INT_SIZE
```

4. Open the file (**numbers.bin**) using the **`open()`** system call. Since you will be simply reading this file use the **`O_RDONLY`** option. E.g.:

```
int mmapfile_fd = open("numbers.bin", O_RDONLY);
```

5. Use the **`mmap()`** system call to memory map this file. E.g.:
`mmapfptr = mmap(0, MEMORY_SIZE, PROT_READ, MAP_PRIVATE, mmapfile_fd, 0);`

6. Retrieve the contents of the memory mapped file (using a loop) and store it in the integer array using **`memcpy()`** function. Sample code to use `memcpy()` is:

```
memcpy(intArray + i, mmapfptr + 4*i, INT_SIZE);
```

`INT_SIZE` = the size of the contents in bytes to be copied from the memory mapped file to **`intArray`**. Since we are reading only 4 bytes (size of an integer) at a time, `INT_SIZE` = 4.

7. Unmap the memory mapped file using the **`unmap()`** system call. E.g.:
`munmap(mmapfptr, MEMORY_SIZE);`

8. Loop through `intArray` to add all numbers in the array, and output this sum to the console.

9. Compile your program without errors.

10. Show the program output to your TA.

Sample Program output: **./lab8b9a9b**

```
Sum of numbers = 92
```