**Memory Mapped Files**

**National University of Computer and Emerging Sciences, Lahore**

**Operating System Lab (Spring 2018)**

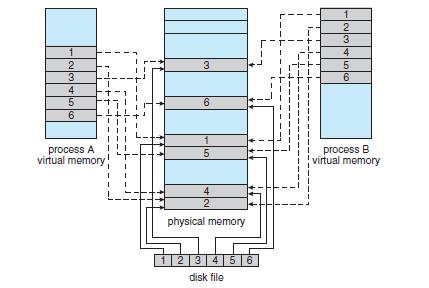


Fig: Memory mapped files

Memory mapping a file is accomplished by mapping a disk block to a page (or pages) in memory. Subsequent reads and writes to the file are handled as routine memory accesses. Manipulating files through memory rather than incurring the overhead of using the read() and write() system calls simplifies and speeds up file access and usage.

Reference: Abraham Silberschatz-Operating System Concepts (9th edition) Pg no. 430

Creating a memorymap

void \*mmap(void \**addr*, size\_t *len*, int *prot*, int *flags*, int *fildes*, off\_t *off*);

* **addr :** This is the address we want the file mapped into. The best way to use this is to set it to **(caddr\_t)0 or NULL** and let the OS choose it for you. If you tell it to use an address the OS doesn't

 like (for instance, if it's not a multiple of the virtual memory page size), it'll give you an error. 

 **len :** This parameter is the length of the data we want to map into memory. This can be any length

you want. (Aside: if *len* not a multiple of the virtual memory page size, you will get a blocksize

that is rounded up to that size. The extra bytes will be 0, and any changes you make to them will

 not modify the file.) 

* **prot:** The "protection" argument allows you to specify what kind of access this process has tothe memory mapped region. PROT\_READ, PROT\_WRITE, and PROT\_EXEC, for read, write, and execute permissions, respectively. The value specified here must be equivalent to the mode

 specified in the **open()** system call that is used to get the file descriptor. 

* **flags :** You'll want to set it to MAP\_SHARED if you're planning to share your changes to the filewith other processes, or MAP\_PRIVATE otherwise. If you set it to the latter, your process will get a copy of the mapped region, so any changes you make to it will not be reflected in the original

 file—thus, other processes will not be able to see them. 

 **fields :**This is where you put that file descriptor you opened earlier.

* **off :** This is the offset in the file that you want to start mapping from. A restriction: this*must*be

a multiple of the virtual memory page size. This page size can be obtained with a call to **getpagesize()**.

**Return value:**

 **mmap**() returns a pointer to the mapped area. On error, the value **MAP\_FAILED** is returned



**Munmap:**

* int munmap(void \**addr***, size\_t** *len***);** 

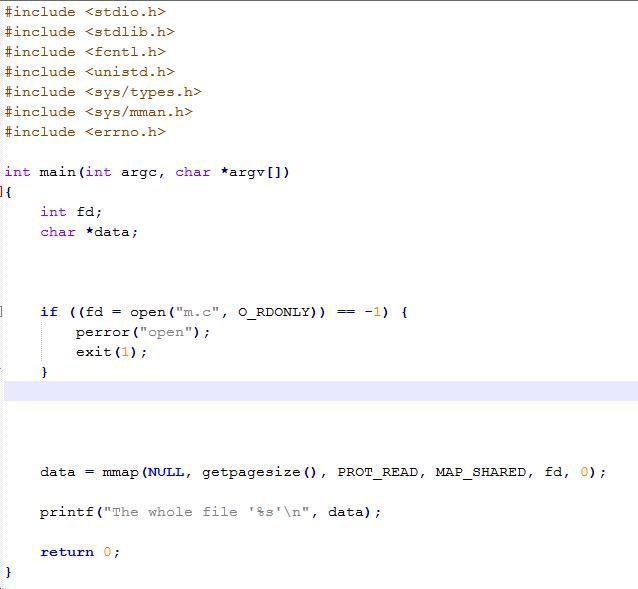


* On success, **munmap**() returns 0. On failure, it returns -1. 

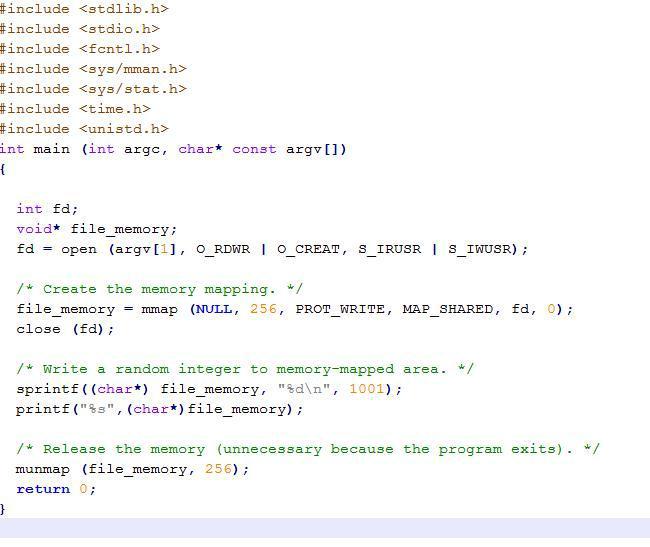
Header file to be included

**#include <sys/mman.h>**

For reading the memory map area



For writing in the memory mapped area



**Inlab Question:**

**Question 1:** Write C/C++ code for a program that takes as command line argument the file name and the substring to be found in file. Your program will make a memory map of the file and find the number of times the substring has occurred in the file. Create 2 threads for searching. First thread will search for substring in the first half, and the second thread will search for string in the second half of map. Whenever, a thread finds the string it increments the count of some shared variable “count”. Since count is being shared by both threads, you must synchronize the access using semaphore. After both threads have terminated, the main thread will print the count on the screen.

Example:

If the data in the file is “We went shopping on Sunday. There was hustle and bustle in the market. We also went shopping on Saturday.”, and the substring is “went shopping”; then your program must output 2.