Programming Assignment 5

- **The due day: April 30.**
- **No credit for a program that does not compile or does not run.**

This assignment requires you to write a multi-processed program that works with a *memory-mapped file*. With the help of virtual memory, memory mapping approach allows a disk file to be initially mapped into memory through demand paging and then to be manipulated subsequently via memory accesses rather than disk file I/O. This can lead to improved performance (refer to pp. 430-433 of the textbook for details). Your C program will spawn two child processes that manipulate a memory-mapped file. Make sure that your submitted program compiles and runs on Athena.

1. In your program, the parent process takes a file name from the command line, creates a file with read/write permissions using the given file name, and initializes the file with the following lower-case string:

```
with the mmap call, a disk file gets mapped into memory through demand\n
paging, i.e., a page sized portion of the file is initially read into\n
memory through a page fault. subsequent reads and writes to that portion\n
of the file are handled as routine memory accesses. this improves file\n
processing performance. in this assignment, the parent process memory maps\n
this disk file first, and then creates two child processes that each make\n
some changes to this file. when both child processes are done, changes\n
made in memory are synch'ed to the disk file.\n
```

The parent process then memory maps the file into the memory using the **`mmap()`** system call in the following manner (for more information on **`mmap()`** call, please refer to the on-line manual via "`man mmap`"). Afterwards, the parent creates two child processes, waits for the child processes to finish before it exits.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <signal.h>
#include <string.h>

struct stat buf;
char  *mm_file,
   ……

fstat(fd, &buf);   //used to determine the size of the file
if ((mm_file = mmap(0, (size_t) buf.st_size, PROT_READ|PROT_WRITE,
                    MAP_SHARED, fd, 0)) == (caddr_t) - 1) {
 fprintf(stderr, "mmap call fails\n");……
}
```

2. The first child process converts the content of the file to its uppercase equivalence through string manipulation, and then uses the **`msync`** system call to synchronize the content of mapped memory with physical storage (disk) before exiting (for more information on **`msync()`** call, please refer to the on-line manual via "`man msync`"). The following **`printf`** statements must be included in the Child-1 process.

```
printf("Child 1 %d reads: \n %s\n", getpid(), mm_file);
  {convert file content to uppercase string};
msync(0, (size_t) buf.st_size, MS_SYNC);
printf("Child 1 %d reads again: \n %s\n", getpid(), mm_file);
```

3. The second child process, again through string manipulation, makes the following changes to the file after the first child converted the string into its uppercase equivalence:

- to use a hyphen to replace the space between "**PAGE**" and "**SIZED**" (**PAGE-SIZED**); and
- to replace occurrence of "**MAPPED**" with "**LOADED**".

```
sleep(1);  //so that child 2 will perform its task after child 1 finishes
printf("Child 2 %d reads: \n %s\n", getpid(), mm_file);
  {use a hyphen to replace the space between "PAGE" and "SIZED"};
  {replace "MAPPED" with "LOADED"};
printf("Child 2 %d reads again: \n %s\n", getpid(), mm_file);
```

The above two **printf** statements must be included in the Child-2 process.

4. Submission Requirements. Your program must include adequate commenting (**points deduction for programs with inadequate comments**). Compile your program:

```
gcc   prog5.c  -o  prog5
```

Submit your source code to **Program 5 Submission** in SacCT.