7. Virtual Memory

Memory-mapped files in Linux offer an efficient and elegant way to access file contents by mapping them into a process's memory space. This technique leverages the virtual memory system of the operating system to provide direct byte-level access to files, which can be highly advantageous for certain types of applications, especially those dealing with large data files.

Key Linux API functions used in conjunction with memory-mapped files include `mmap`, `munmap`, `mprotect`, and `open`:

1. `mmap`: This function is the cornerstone of memory-mapped file operations. It creates a new mapping in the virtual address space of the calling process. The contents of a file can be mapped directly into the process's memory, allowing the application to read and write to the file as if it were manipulating memory. This can lead to significant performance improvements as it reduces the number of I/O system calls and data copying.
2. `munmap`: To release the mapped memory, `munmap` is used. This function deallocates the memory segment that was previously mapped by `mmap`, ensuring that resources are freed and not leaked. It is crucial for maintaining the health and efficiency of the application.
3. `mprotect`: While working with memory-mapped files, there might be a need to change the access permissions of the mapped memory region (like read, write, or execute). `mprotect` serves this purpose, offering control over how the memory can be accessed, thus enhancing security and preventing unauthorized or unintended memory access.
4. `open`: Before mapping a file into memory, it must be opened. The `open` function is used for this purpose. It opens the file and returns a file descriptor, which `mmap` then uses to create the memory mapping. The `open` function also allows specifying the mode in which the file should be accessed (e.g., read-only, read-write).

A. Linux APIs for memory-mapped files

mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset);

1. start:
   This argument gives a preferred starting address for the mapping. If you pass NULL, the operating system will choose a suitable address automatically.
2. size_t:
   This is the number of bytes which to be mapped.
3. prot:
   This argument is used to control what kind of access is permitted.
   PROT_NONE: No access.
   PROT_READ: Read access.
   PROT_WRITE: Write access.
   PROT_EXEC: Execute access.
   You can combine these flags using the bitwise OR operator (|). For example, to specify read and write access, use PROT_READ | PROT_WRITE.
4. flags:
   This argument is used to control the nature of the map.
   Following are some common values of the flags:
   MAP_SHARED: Creates a shared mapping between processes.
   MAP_PRIVATE: Creates a private copy-on-write mapping.
   MAP_ANONYMOUS: Maps memory not associated with a file.
   MAP_FIXED: Forces the mapping to start at the specified address (if possible).
5. fd:
   This is the file descriptor of the file to be mapped.
6. offset
   For file mappings, this specifies the offset from the beginning of the file where the mapping should start. For other mappings, it should be set to 0.

munmap(void *start, size_t length);

1. Start

   The starting address of the memory region to unmap. This should be the same address that you received from the mmap() call.

2. length

   The length of the memory region to unmap in bytes. This should match the length you specified in the mmap() call.

open(const char* Path, int flags, mode_t mode);

1. Path

   This argument is the path of the file you try to open.

2. flags

   This argument controls how the file is opened and what operations are allowed.

   ex:

   O_RDONLY: Open the file for reading only.

   O_WRONLY: Open the file for writing only.

   O_RDWR: Open the file for both reading and writing.

   O_CREAT: Create the file if it doesn't exist.

3. mode

   This argument is used to open files with specific modes that define how the file should be accessed.

   ex:

   S_IRUSR: Read permission for the owner.

   S_IWUSR: Write permission for the owner.

   S_IXUSR: Execute permission for the owner.

mprotect(void *addr, size_t len, int prot);

1. addr

   This argument is the pointer to the starting address of the memory region whose protection attributes you want to change.

2. len

   This argument is the length (in bytes) of the memory region you want to change.

3. prot

   An integer representing the new protection attributes you want to set for the memory region.

   ex:

   PROT_NONE: No access is allowed (no read, write, or execute).

   PROT_READ: Read access is allowed.

   PROT_WRITE: Write access is allowed.

   PROT_EXEC: Execute access is allowed.

B. Implementation

In this assignment, you need to fill up the missing code in two programs `hw7_write1.c` and `hw7_write2.c`.

Part1 . hw7_write1.c (45%)

This program uses open(), mmap() and munmap() to create a shared file, and write some data to it.

Please complete the missing code on those lines of code with comments.

Warning: you must not modify code in the other lines of code !!!

Part2 . hw7_write2.c (15%)

This program is almost the same as hw7_write1.c. However, when we create the memory mapping via mmap, we set PROT_NONE to disable all types of accesses to the mapped pages. Then, we will use mprotect() to permit the write access.

Your task is to borrow code from hw7_write1.c and compete the call to mprotect().

Warning: you should only modify those lines of code marked with comments !!!

Part3 . Report (English or Chinese)

Provide brief answers to the following questions.

Q1(20%) : In Part 1, you used three APIs: mmap(), munmap(), and open(). Please explain the arguments passed to each API.

Q2(10%) : In Part2, you used the API mprotect(). Please explain the arguments passed to mprotect().

Q3(10%) : What advantages does memory-mapped I/O offer over standard file I/O (e.g. read, read, …)?

Hint: You can use the read1.c and read2.c to verify the correctness of your program.

C. Submission

The name of the zip file should be <student_id>.zip, and the structure of the file should be as the following:

```
<stduent_id>.zip
    |- <student_id>/
        |- hw7_write1.c
        |- hw7_write2.c
        |- hw7.pdf
```

For questions, please contact TA Mr. Lai <loseit7382.cs11@nycu.edu.tw>