

Assignment12

Team18

112062519 廖思愷

112062636 游竣量

111065547 游述宇

- Explanations of implemented code

```
int main(int argc, char *argv[])
{
    int fdin, fdout;
    void *src, *dst;
    size_t copysz;
    struct stat sbuf;
    off_t fsz = 0;

    if (argc != 3)
        err_quit("usage: %s <fromfile> <tofile>", argv[0]);

    if ((fdin = open(argv[1], O_RDONLY)) < 0)
        err_sys("can't open %s for reading", argv[1]);

    if ((fdout = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, FILE_MODE)) < 0)
        err_sys("can't creat %s for writing", argv[2]);

    if (fstat(fdin, &sbuf) < 0) /* need size of input file */
        err_sys("fstat error");

    if (ftruncate(fdout, sbuf.st_size) < 0) /* set output file size */
        err_sys("ftruncate error");
```

變數解釋

- **int fdin, fdout**：檔案描述符。**fdin** 用於來源檔案，**fdout** 用於目標檔案。
- **void *src, *dst**：這兩個指標變數將在後面用於指向映射到記憶體中的檔案區域。**src** 用於來源檔案，**dst** 用於目標檔案。
- **size_t copysz**：這是一個表示大小的變數，用於確定每次映射和複製的數據量。

- **struct stat sbuf**：這是一個結構體，用於存儲檔案的狀態資訊，如檔案大小等。
- **off_t fsz = 0**：**off_t** 是表示檔案偏移量的數據類型。**fsz** 用於記錄已經複製的數據量，初始化為 0。

檔案前處理

- **if (argc != 3)**：檢查命令行參數的數量是否正確。
- **open(argv[1], O_RDONLY)**：使用 **open** 系統呼叫打開來源檔案為唯讀模式。**argv[1]** 是命令行中的第二個參數，代表來源檔案路徑。
- **open(argv[2], O_RDWR | O_CREAT | O_TRUNC, FILE_MODE)**：同樣使用 **open** 打開目標檔案，但是模式為可讀寫，如果不存在則創建，如果已存在則截斷。
- **fstat(fdin, &sbuf)**：使用 **fstat** 獲取來源檔案的大小等資訊。
- **ftruncate(fdout, sbuf.st_size)**：將目標檔案的大小設定為與來源檔案相同。

```
while (fsz < sbuf.st_size) {
    if ((sbuf.st_size - fsz) > COPYINCR)
        copysz = COPYINCR;
    else
        copysz = sbuf.st_size - fsz;

    /* TODO: Copy the file using mmap here */
    src = mmap(0, copysz, PROT_READ, MAP_PRIVATE, fdin, fsz);
    if (src == MAP_FAILED)
        err_sys("mmap error for input");
    dst = mmap(0, copysz, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, fsz);
    if (dst == MAP_FAILED)
        err_sys("mmap error for output");
    memcpy(dst, src, copysz);
    munmap(src, copysz);
    munmap(dst, copysz);
    fsz += copysz;
}
close(fdin);
close(fdout);
exit(0);
```

- **循環條件**：當 **fsz**（已複製的大小）小於來源檔案的總大小（**sbuf.st_size**）時，繼續執行循環。
- **計算複製大小**：**copysz** 變數用於確定這一次從來源檔案中要映射和複製多少數據。如果剩餘的數據量大於 **COPYINCR**（定義的最大複製量），則使用 **COPYINCR** 作為這次複製的大小；否則，使用剩餘的數據量。

- **映射來源檔案：**使用 `mmap` 將來源檔案的一部分映射到記憶體中。這裡的 `PROT_READ` 表示映射區域可被讀取，`MAP_PRIVATE` 表示對映射區域的修改不會影響原始檔案。
 - **映射目標檔案：**同樣使用 `mmap`，但此處的標誌是 `PROT_READ | PROT_WRITE`（可讀寫），並且使用 `MAP_SHARED`，表示對映射區域的修改將會反映到檔案本身。
 - **數據複製：**使用 `memcpy` 將數據從來源 (`src`) 複製到目標 (`dst`)，且對 `dst` memory 區塊的修改會反映到檔案本身。
 - **釋放映射：**用 `munmap` 釋放之前映射的兩個記憶體區域。
 - **更新已複製數據大小：**`fsz` 被更新，加上剛才複製的大小，為下一次循環做準備。
 - **關閉檔案：**使用 `close` 函數關閉先前打開的來源和目標檔案的檔案描述符。
 - **退出程式：**通過 `exit(0)` 終止程序。這裡的 `0` 表示程序正常結束。
-
- **Screenshot of result**

```
team18@~/Kyle_test/assignment12 $ ls
Makefile      assignment12  assignment12.c assignment12.o source.txt    try          try.c
team18@~/Kyle_test/assignment12 $ ./assignment12 source.txt dest.txt
team18@~/Kyle_test/assignment12 $ ls
Makefile      assignment12  assignment12.c assignment12.o dest.txt      source.txt    try          try.c
team18@~/Kyle_test/assignment12 $ diff source.txt dest.txt
team18@~/Kyle_test/assignment12 $
```

由 diff 的結果可知，source.txt 與 dest.txt 內容一樣，複製成功。

- Will closing the file descriptor invalidate the memory-mapped

I/O?

不會，解釋如下：

在呼叫 mmap 後關閉 source file 及 dest file (修改後的程式碼片段)

```
while (fsz < sbuf.st_size) {
    if ((sbuf.st_size - fsz) > COPYINCR)
        copysz = COPYINCR;
    else
        copysz = sbuf.st_size - fsz;

    /* TODO: Copy the file using mmap here */
    src = mmap(0, copysz, PROT_READ, MAP_PRIVATE, fdin, fsz);
    close(fdin);
    if (src == MAP_FAILED)
        err_sys("mmap error for input");
    dst = mmap(0, copysz, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, fsz);
    close(fdout);
    if (dst == MAP_FAILED)
        err_sys("mmap error for output");
    memcpy(dst, src, copysz);
    munmap(src, copysz);
    munmap(dst, copysz);
    fsz += copysz;
}
exit(0);
```

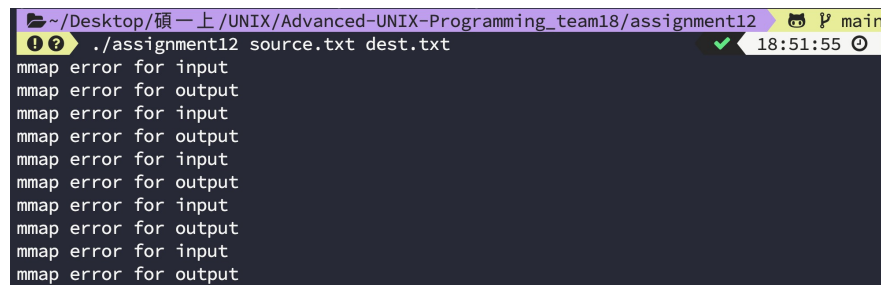
結果

```
team18@~/Kyle_test/assignment12 $ ls
Makefile      assignment12  assignment12.c  assignment12.o  source.txt    try           try.c
team18@~/Kyle_test/assignment12 $ ./try source.txt dest.txt
team18@~/Kyle_test/assignment12 $ ls
Makefile      assignment12  assignment12.c  assignment12.o  dest.txt      source.txt    try           try.c
team18@~/Kyle_test/assignment12 $ diff source.txt dest.txt
team18@~/Kyle_test/assignment12 $ |
```

仍然可以讀取 source file 並寫入 dest file。

所以，關閉 file descriptor 後 memory-mapped I/O 仍然有效，這表示仍然可以對 mapping 的記憶體區域進行存取，寫入 mapping 的記憶體區域也會反映到設有 MAP_SHARED flag 的檔案。

備註：mmap 必須在 fd 關閉之前設定好，也就是說對於上面的範例，只有第一次 while 有效，如果 source file 過大，while 需要多輪才能處理完所有資料，則從第二輪開始 mmap 會找不到對應的 fd 而報錯。

A terminal window with a dark background. The title bar shows the path ~/Desktop/碩一上 /UNIX/Advanced-UNIX-Programming_team18/assignment12 and a terminal icon with the text 'main'. The prompt is './assignment12 source.txt dest.txt' followed by a green checkmark icon and the time '18:51:55'. The output consists of ten lines of error messages: 'mmap error for input', 'mmap error for output', 'mmap error for input', 'mmap error for output', 'mmap error for input', 'mmap error for output', 'mmap error for input', 'mmap error for output', 'mmap error for input', and 'mmap error for output'.

```
~/Desktop/碩一上 /UNIX/Advanced-UNIX-Programming_team18/assignment12 main  
./assignment12 source.txt dest.txt ✓ 18:51:55  
mmap error for input  
mmap error for output  
mmap error for input  
mmap error for output  
mmap error for input  
mmap error for output  
mmap error for input  
mmap error for output  
mmap error for input  
mmap error for output
```