410410043 SP_Hw2

姓名：林秉燁

系級：資工二

學號：410410043

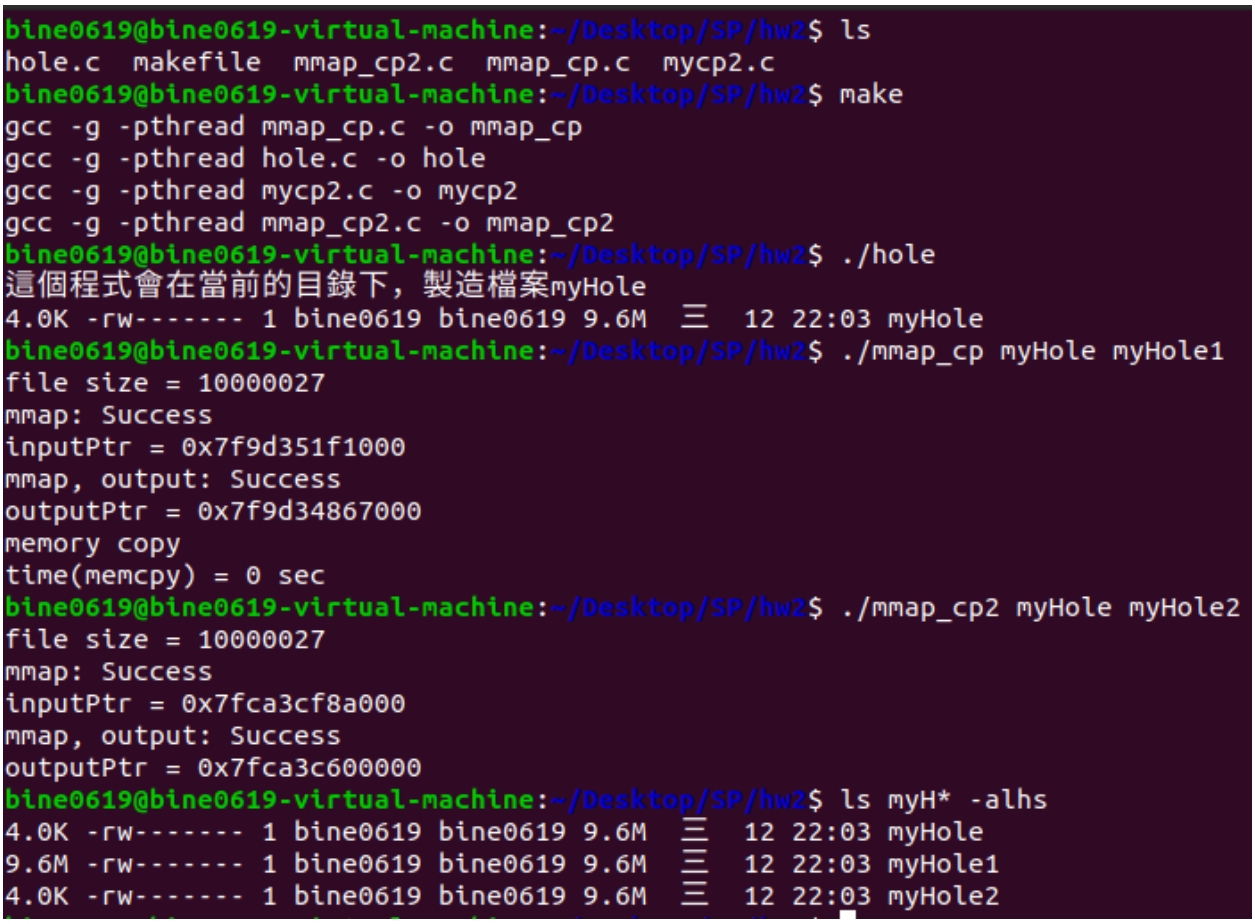
作業2 使用mmap進行檔案複製

以下是我mmap_cp2的程式碼

```
#define _GNU_SOURCE

#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <time.h>

int main(int argc, char* argv[]) {

    int inputFd, outputFd;
    char *inputPtr, *outputPtr;
    ssize_t numIn, numOut;
    ssize_t fileSize=0;
    //char buffer[BUF_SIZE];

    //只可讀取模式打開
    inputFd = open (argv [1], O_RDONLY);
    if (inputFd == -1) {
        perror ("cannot open the file for read"); exit(1); }

    //open後可對該檔案＊＊『可讀可寫』＊＊（因為mmap的需求），如果沒有該檔案，就建立該檔案。
    //如果要建立，設定該檔案的屬性為owner可讀可寫
    outputFd = open(argv[2], O_RDWR | O_CREAT, S_IRUSR| S_IWUSR);
    if(outputFd == -1){
        perror("canot open the file for write"); exit(1); }

    //lseek的回傳是該檔案的絕對位址，因此lseek(0, seek_end)相當於檔案大小
```

```
    //linux有專門讀取檔案大小的函數，但我習慣用這一個
    fileSize = lseek(inputFd, 0, SEEK_END);
    printf("file size = %ld\n", fileSize);

    //NULL，不指定映射到記憶體的哪個位置。通常不指定
    //filesize，將檔案中多少內容映射到記憶體
    //prot_read，只會對該段記憶體做讀取
    //MAP_SHARED，對mmap出的記憶體的所有修改讓整個系統裡的人都看到。因此底藏的檔案會跟著變更
    //inputFd從哪個檔案映射進來
    //0, 映射的起點為 0
    inputPtr = mmap(NULL, fileSize, PROT_READ, MAP_SHARED , inputFd , 0);
    perror("mmap");
    printf("inputPtr = %p\n", inputPtr);
    //assert(madvise(inputPtr, fileSize, MADV_SEQUENTIAL|MADV_WILLNEED|MADV_HUGEPAGE)==0);

    //ftruncate的名字是：縮小
    //實際上是設定檔案大小
    ftruncate(outputFd, fileSize);
    outputPtr = mmap(NULL, fileSize, PROT_WRITE, MAP_SHARED , outputFd , 0);
    perror("mmap, output");
    printf("outputPtr = %p\n", outputPtr);
    //madvise(inputPtr, fileSize, MADV_SEQUENTIAL|MADV_WILLNEED|MADV_HUGEPAGE);

    // 開始進行檔案複製
    off_t data_off = 0, hole_off = 0, cur_off = 0;
    long long int blockSize;
    while(1){
        // 找到資料起點跟洞的起點
        cur_off = lseek(inputFd, cur_off, SEEK_DATA);
        data_off = cur_off;
        cur_off = lseek(inputFd, cur_off, SEEK_HOLE);
        hole_off = cur_off;
        // 藉此確定資料的大小
        blockSize = hole_off - data_off;
        // 在利用這個大小去複製
        memcpy(outputPtr + data_off, inputPtr + data_off, blockSize);
        lseek(outputFd, cur_off, SEEK_SET);
        if((lseek(outputFd, 0, SEEK_CUR)) == fileSize) break;
    }

    assert(munmap(inputPtr, fileSize) == 0);
    assert(munmap(outputPtr, fileSize) == 0);

    assert(close (inputFd) == 0);
    assert(close (outputFd) == 0);

    return (EXIT_SUCCESS);
}
```

以下為從建立檔案到檢查的步驟



先利用make產生執行檔

再執行hole 產生myHole

接下來分別用mmap_cp 跟 mmap_cp2 做一次複製

在將原本與複製出來的兩個檔案展示出來即可發現

原本myHole 9.6M的大小只佔了磁碟的 4.0K

然而因為mmap_cp無法有效處理有洞的檔案

所以仍會佔用9.6M的大小

但利用mmap_cp2所複製出來的myHole2 會跟原本的myHole一樣只佔了4.0K

以下是我比較mycp2 跟 mmap_cp2

```
bine0619@bine0619-virtual-machine:~/Desktop/SP/hw2$ time ./mycp2 myHole myHole1

real    0m0.002s
user    0m0.000s
sys     0m0.002s
bine0619@bine0619-virtual-machine:~/Desktop/SP/hw2$ time ./mmap_cp2 myHole myHole2
file size = 50010000502
mmap: Success
inputPtr = 0x7f2058038000
mmap, output: Success
outputPtr = 0x7f14b32f7000

real    0m0.005s
user    0m0.000s
sys     0m0.004s
```

原先的hole.c所產生的myHole所進行的比較不明顯

因此改寫了hole.c 讓資料跟hole都比較大 才有一些些差異

會發現mmap_cp2還是會略慢mycp2一些，若資料量越來越大，差異就越明顯