ADVANCED UNIX PROGRAMMING ASSIGNMENT REPORT

ASSIGNMENT 12

TEAM 9 — 林禾堃、馬毓昇、陳曦

Dec 2023

1. Code Implementation

Synopsis of *mmap:*

```
void *mmap(void addr[.length], size_t length, int prot, int flags, int fd, off_t offset);
```

Considering this, we know the arguments that *mmap* needs for the input file and the output file are
NULL, copysz, copysz, PROT_READ, MAP_SHARED, fdin, fsz, and
NULL, copysz, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, fsz respectively in this case, so we called *mmap* twice with these two sets of arguments and store the addresses in two pointers:

```
/* TODO: Copy the file using mmap here */
char *src_buf = mmap(NULL, copysz, PROT_READ, MAP_SHARED, fdin, fsz);
//close(fdin);
char *dst_buf = mmap(NULL, copysz, PROT_READ | PROT_WRITE, MAP_SHARED, fdout,
```

After this, we utilize *memcpy* to copy the content of the file from source to destination, and add the offset by *copysz*:

```
memcpy(dst_buf, src_buf, copysz);
fsz += copysz;
```

Performing this session in the while loop,

```
while (fsz < sbuf.st_size) {
```

the whole file can be copied via memory-mapped I/O.

2. Will closing the file descriptor invalidate the memory-mapped I/O?

No, it will not. Since *mmap* will map the file to the memory, we can perform the desired I/O via memory accessing despite the closing of the file descriptor. Additionally, considering in some cases, *close* might affect the functionality of the loop due to the time it takes, we want to remove any doubt about this. The reason why *close* will not pose a threat to our code is that the size of the source file used in this assignment isn't large, preventing the aforementioned issue from invalidating the memory-mapped I/O.