Unix Assignment12

Team19 108072244 邱煒甯 108032053 陳凱揚 112065525 簡佩如

Implementation

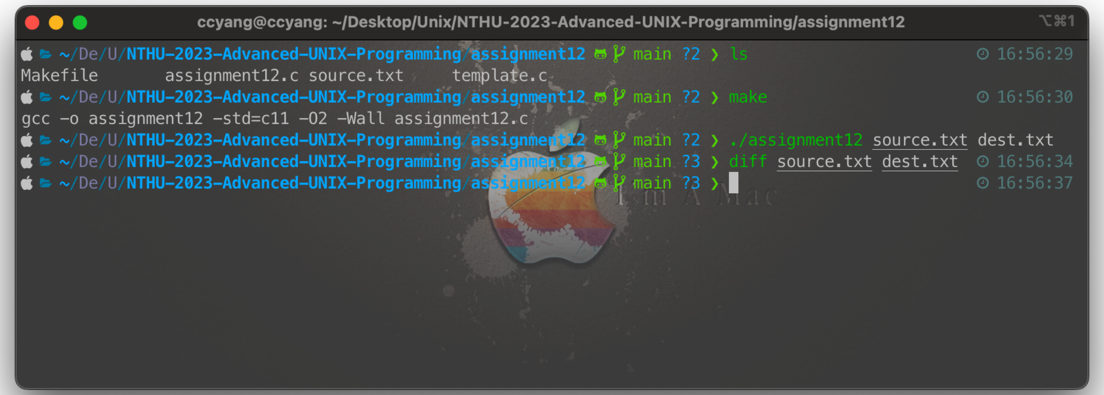
In the each iteration of the while loop, when copying `copysz` bytes starting from the offset `fsz` from the source file to the destination file, we must calculate the `page_offset` for the source file. This is necessary because the offset provided for `mmap()` must be page aligned. The `page_offset` is the last offset that is page aligned and comes before the offset `fsz`.

For the source file, we use the `PROT_READ` and `MAP_PRIVATE` flags in the `mmap` function to ensure that the operation is read-only. For the destination file, we use `PROT_READ`, `PROT_WRITE`, and `MAP_SHARED` to make the changes visible to other processes.

For both source file and destination file, we also need to add the extra bytes between the `page_offset` and `fsz` to the length provided for `mmap()`. The size is calculated as `fsz - page_offset`. When using memcpy, we skip this size by adding the size to both the source and destination pointers. Finally, we use `munmap` to unmap the memory.

```c
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 */
    // Find the page_offset which is page aligned and comes before the fsz
    off_t page_offset = fsz & ~(sysconf(_SC_PAGE_SIZE) - 1);

    // Memory-map the input file and output file
    if ((src = mmap(NULL, copysz + fsz - page_offset, PROT_READ, MAP_PRIVATE, fdin, page_offset)) == MAP_FAILED)
        err_sys("mmap error for input");
    if ((dst = mmap(NULL, copysz + fsz - page_offset, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, page_offset)) == MAP_FAILED)
        err_sys("mmap error for output");

    // Copy the data
    memcpy(dst + fsz - page_offset, src + fsz - page_offset, copysz);

    // Unmap the memory
    if (munmap(src, copysz + fsz - page_offset) < 0)
        err_sys("munmap error for input");
    if (munmap(dst, copysz + fsz - page_offset) < 0)
        err_sys("munmap error for output");

    fsz += copysz;
}

// Close the file descriptors
close(fdin);
close(fdout);
```

Result



```
ccyang@ccyang: ~/Desktop/Unix/NTHU-2023-Advanced-UNIX-Programming/assignment12                ⌥⌘1
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment12  main ?2 ❯ ls                    16:56:29
Makefile       assignment12.c source.txt     template.c
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment12  main ?2 ❯ make                  16:56:30
gcc -o assignment12 -std=c11 -O2 -Wall assignment12.c
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment12  main ?2 ❯ ./assignment12 source.txt dest.txt
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment12  main ?3 ❯ diff source.txt dest.txt  16:56:34
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment12  main ?3 ❯                       16:56:37
```

Question

- Try to close the input file after calling `mmap` and answer the following question in the report: Will closing the file descriptor invalidate the memory-mapped I/O?

  No, closing the file descriptor doesn't invalidate the memory-mapped region itself since it doesn't remove the reference count to the file which incremented by the `mmap` function. We need to use `munmap` to stop the mapping.