Системне програмне забезпечення. Лабораторна робота 2

Мета роботи: практичне використання системного виклику mmap()

Частина 1: Обчислення контрольної суми файла

Приклад запуску:

```
# ./lab2 test.txt
Filename passed: test.txt
File size: 1496
Mapping address: 0x7ff4dc1ab000
Check sum: 120
```

Алгоритм роботи:

1. Отримати назву файла з аргументу виклику програми
2. Дізнатися розмір файла
3. Імплементувати відображення його в пам'ять тільки для читання і отримати вказівник на початок відображення
4. Обчислити контрольну суму як XOR усіх байтів файла та вивести її значення
5. Звільнити вказівник відображення

0. Підключити заголовочні файли

В даній лабораторній роботі використовуються наступні системні виклики: open(), close(), lstat(), mmap(), munmap(), exit(). Довідкову інформацію про них можна отримати виконавши команду

```
man 2 назва_системного_виклику
```

Або відкривши посилання
https://linux.die.net/man/2/open
https://linux.die.net/man/2/close
https://linux.die.net/man/2/lstat
https://linux.die.net/man/2/mmap

В секції синтаксичного опису приводиться перелік заголовочних файлів необхідних для використання системного виклику. Для даної лабораторної роботи список такий:

```
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/sysmacros.h>
```

```
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
```

1. Отримати назву файла з аргументу виклику програми

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

      if (argc != 2) {
            fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
            exit(EXIT_FAILURE);
      }
      printf("Filename passed: %s\n", argv[1]);

      return 0;
}
```

```
argc - кількість переданих аргументів в програму
argv[0] - назва бінарного файла програми
argv[1] - перший аргумент командного рядка
```

2. Дізнатися розмір файла

```
size_t get_file_size(const char *filename)
{
      struct stat sb;

      if (lstat(filename, &sb) == -1) {
            perror("lstat");
            exit(EXIT_FAILURE);
      }
      return sb.st_size;
}
```

```
struct stat sb - буфер для отримання інформації з системного виклику
lstat - системний виклик для отримання інформації про файл
```

3. Імплементувати відображення файла в пам’ять

```
const uint8_t *get_read_mapping(const char *filename, size_t size)
{
      int fd;
      void *p;
```

```
	fd = open(filename, O_RDONLY);
	if (fd == -1) {
		perror("open");
		exit(EXIT_FAILURE);
	}

	p = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
	if (p == MAP_FAILED) {
		perror("mmap");
		exit(EXIT_FAILURE);
	}

	close(fd);
	return (const uint8_t*)p;
}
```

fd - тимчасовий файловий дескриптор
p - адреса відображення
open - системний виклик відкриття файла
close - системний виклик закриття файла
mmap - системний виклик системний виклик створення відображення файла в пам’ять

Зверніть увагу, що файловий дескриптор закривається відразу після створення відображення. Для подальшого зчитування даних з файла достатньо вказівника на відображення в пам’ять.

4. Обчислити контрольну суму як XOR усіх байтів файла та вивести її значення

```
mapping = get_read_mapping(argv[1], size);

for (i = 0; i < size; i++)
	checksum ^= mapping[i];
```

Зверніть увагу, доступ до файла відбувається через вказівник mapping, тобто немає використання системного виклику read() для зчитування даних.

5. Звільнити вказівник відображення

```
void release_mapping(const uint8_t *p, size_t size)
{
	munmap((void*)p, size);
}
```

munmap - системний виклик для звільнення відображення в пам’ять

Частина 2: Заповнення текстового файла символом

Виконання роботи:

1. Створити довільний текстовий файл
2. За допомогою програми заповнити його символами
3. Переконатися що файл було змінено успішно.

Приклад виконання:

```
# echo "1234567890" > test.txt
# ./lab2_write test.txt
Filename passed: test.txt
File size: 11
Mapping address: 0x7fd41adc1000
Done
# cat test.txt
aaaaaaaaaaa
```

Алгоритм роботи:

1. Отримати назву файла з аргументу виклику програми
2. Дізнатися розмір файла
3. Імплементувати відображення його в пам’ять *для запис*у і отримати вказівник на початок відображення
4. *Заповнити файл довільним символом*
5. Звільнити вказівник відображення

3. Імплементувати відображення файла в пам’ять для запису

```
uint8_t *get_write_mapping(const char *filename, size_t size)
{
      int fd;
      void *p;

      fd = open(filename, O_RDWR);
      if (fd == -1) {
      perror("open");
      exit(EXIT_FAILURE);
      }

      p = mmap(NULL, size, PROT_READ | PROT_WRITE,
                                 MAP_SHARED, fd, 0);
      if (p == MAP_FAILED) {
      perror("mmap");
      exit(EXIT_FAILURE);
      }
```

```
      close(fd);
      return (uint8_t*)p;
}
```

Зверніть увагу на змінені аргументи системних викликів open та mmap.

4. Заповнити файл довільним символом

```
mapping = get_write_mapping(argv[1], size);

for (i = 0; i < size; i++)
      mapping[i] = 'a';
```

Доступ до файла відбувається через вказівник mapping, тобто немає використання системного виклику write() для запису даних в файл.