

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу
«Операционные системы»**

Студент: Друхольский А.К.

Группа: М8О-207Б-21

Вариант: 22

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2022

Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения

5. Исходный код
6. Демонстрация работы программы
7. Выводы

Репозиторий

<https://github.com/ssForz/OS-labs>

Постановка задачи

Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних

процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Общие сведения о программе

Оба процесса представлены в одном файле lab.cpp. Компилируется при помощи `make`

- `sem_open()` - инициализируем и открываем семафор
- `(char *)mmap(0, mapsize, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0)` — создаём memory map для передачи информации между процессами, настраивая определёнными флагами
- `munmap(mapped, mapsize)` — закрытие memory map
- `sem_close(semaphore)` — отключение семафора
- `sem_unlink(sem_file)` — закрываем семафор по данному имени

Общий метод и алгоритм решения

Главная идея в том, чтобы изначально создать mmap размера под нашу задачу. В таком случае не придётся перезаписывать конкретные участки памяти под новые строки. В начале каждого блока строки будет записывать длину строки, что нужно для упрощения алгоритма. Итак мы имеем mmap, в котором по порядку идут: размер-строка-размер-строка...

Исходный код

```
#include <iostream>
#include <string>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <unistd.h>
#include <fstream>
#include <errno.h>
#include <sys/mman.h>
#include <cstdio>
```

```
using namespace std;
```

```

int flaccess = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

int child_execute(string path, char *mapped, string sem_file) {

    int counter = 0;
    string filename = path;
    fstream cur_file;
    cur_file.open(filename, fstream::in | fstream::out | fstream::app);
    sem_t *semaphore = sem_open(sem_file.c_str(), 1);

    while (true) {
        if (sem_wait(semaphore) == -1) {
            perror("Semaphore error");
            exit(EXIT_FAILURE);
        }
        if (mapped[counter] == '!') {
            cout<<"ITS OVER"<<endl;
            break;
        }
        int cur_size = (int)mapped[counter];

        int start = counter;

        char str_array[cur_size];
        int i = 0;
        for (; counter < start + cur_size ; counter++) {
            str_array[i] = mapped[counter+1];
            i++;
        }
        string result_str;
        for (int j = cur_size - 1; j >= 0; j--) {
            cur_file << str_array[j];
        }
        cur_file<<endl;
        cout<<"Added result stroke to "<<filename<<endl;
        counter++;
    }
}

```

```

    }

    return 0;

}

int main()
{
    string current_str;
    string sem_file1 = "a.semaphore";
    string sem_file2 = "b.semaphore";
    string filename1, filename2;

    cout<<"Enter the name for first child file: ";
    cin>>filename1;
    cout<<endl;

    cout<<"Enter the name for second child file: ";
    cin>>filename2;
    cout<<endl;

    cout<<"Enter amount of strings: ";
    int cnt;
    cin>>cnt;
    const int mapsize = cnt*256;

    sem_t *semaphore1 = sem_open(sem_file1.c_str(), O_CREAT, flaccess, 0);
    if (semaphore1 == SEM_FAILED) {
        perror("Semaphore error");
        exit(EXIT_FAILURE);
    }
    sem_t *semaphore2 = sem_open(sem_file2.c_str(), O_CREAT, flaccess, 0);
    if (semaphore2 == SEM_FAILED) {
        perror("Semaphore error");
        exit(EXIT_FAILURE);
    }
}

```

```
char *mapped1 = (char *)mmap(0, mapsize, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
```

```
char *mapped2 = (char *)mmap(0, mapsize, PROT_READ | PROT_WRITE, MAP_SHARED |
MAP_ANONYMOUS, -1, 0);
```

```
pid_t f_id1 = fork();
```

```
if (f_id1 == -1) {
```

```
    cout<<"Fork error with code -1 returned in the parent, no child_1 process is created"<<endl;
```

```
    exit(EXIT_FAILURE);
```

```
} else if (f_id1 == 0) { //child1
```

```
    string childfile = filename1;
```

```
    child_execute(childfile, mapped1, sem_file1);
```

```
    return 0;
```

```
}
```

```
pid_t f_id2 = fork();
```

```
if (f_id2 == -1) {
```

```
    cout<<"Fork error with code -1 returned in the parent, no child_2 process is created"<<endl;
```

```
    exit(EXIT_FAILURE);
```

```
} else if (f_id2 == 0) { //child2
```

```
    string childfile = filename2;
```

```
    child_execute(childfile, mapped2, sem_file2);
```

```
    return 0;
```

```
}
```

```
if (f_id1 != 0 && f_id2 != 0){ //parent
```

```
    cout<<endl;
```

```
    int stat_counter1 = 0;
```

```
    int stat_counter2 = 0;
```

```
    for (int i = 0; i < cnt+1; i++) {
```

```
        if (i == cnt) {
```

```
            mapped1[stat_counter1] = '!';
```

```
            mapped2[stat_counter2] = '!';
```

```
            sem_post(semaphore1);
```

```
            sem_post(semaphore2);
```

```

        break;
    }

    cin>>current_str;
    while (current_str.size() >= 20) {
        cout<<"Wrong size of string, try again (<20): "<<endl;
        cin>>current_str;
    }
    if (rand() % 6 < 5) {
        for (int j = 0; j < current_str.size()+1; j++) {
            if (j == 0) {
                mapped1[stat_counter1] = (char)current_str.size();
                continue;
            }
            mapped1[stat_counter1 + j] = current_str[j - 1];
        }
        sem_post(semaphore1);
        stat_counter1 += current_str.size() + 1;

    } else {

        for (int j = 0; j < current_str.size()+1; j++) {
            if (j == 0) {
                mapped2[stat_counter2] = (char)current_str.size();
                continue;
            }
            mapped2[stat_counter2 + j] = current_str[j - 1];
        }
        sem_post(semaphore2);
        stat_counter2 += current_str.size() + 1;
    }
}

munmap(mapped1, mapsize);
munmap(mapped2, mapsize);
sem_close(semaphore1);
sem_close(semaphore2);

```

```

sem_unlink(sem_file1.c_str());
sem_unlink(sem_file2.c_str());
return 0;
}

```

Демонстрация работы программы

```

alex@saddtype: ~/os-labs/OS-labs/lab-4
-- Generating done
-- Build files have been written to: /home/alex/os-labs/OS-labs/lab-4
alex@saddtype:~/os-labs/OS-labs/lab-4$ make
Scanning dependencies of target main
[ 50%] Building CXX object CMakeFiles/main.dir/src/lab.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
alex@saddtype:~/os-labs/OS-labs/lab-4$ ls
CMakeCache.txt  cmake_install.cmake  main  report
CMakeFiles      CMakeLists.txt       Makefile  src
alex@saddtype:~/os-labs/OS-labs/lab-4$ ./main
Enter the name for first child file: 1.txt
Enter the name for second child file: 2.txt
Enter amount of strings: 8
123
Added result stroke to 1.txt
fdsja
Added result stroke to 1.txt
jj2k1
Added result stroke to 1.txt
fdfsfs
Added result stroke to 1.txt
543
Added result stroke to 2.txt
zfdjds
Added result stroke to 1.txt
kfdsls
Added result stroke to 1.txt
fdsks
ITS OVER
Added result stroke to 1.txt
ITS OVER
alex@saddtype:~/os-labs/OS-labs/lab-4$

```

Выводы

В процессе работы я познакомился с memoxy tar как со способом передачи информации между процессами. Я решил использовать ANON tar, что позволило обойтись без file descriptor, но я столкнулся с проблемой. При передаче указателя между процессами он меняется. Это решается, если оба процесса выполняются одним исполняемым файлом.