Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работ №4 по курсу

«Операционные системы»

Группа: М80 – 207Б-18

Студент: Цапков Александр Максимович

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____

Дата: _____

Москва, 2019.

Содержание

Постановка задачи.

Дочерний процесс представляет собой сервер по работе со стеками и принимает команды со стороны родительского процесса. Комуникация между процессами осуществляется с помощью файл маппинга

Общие сведения о программе

Программа состоит из одного исполняемого файла. В лабораторной работе были использованы следующие системные вызовы.

1. **mmap** — для создания и отображения файла
2. **pthread_mutex_lock** – для блокировка мьютекса.
3. **pthread_mutex_unlock** – для разблокировки мьютекса

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. В родительском процессе отобразить файл для передачи данных. (Приводим мы файл к типу структуры, в которой есть 1 мьютекс и массив с данными для передачи).
2. Запустить дочерний процесс с помощью fork().
3. Отличить родительский процесс от дочернего и запустить функции нужные именно для них.
4. При работе и считывание использовать мьютексы для блокировки процессов, чтобы добиться правильной последовательности выполнения (читать только после того как записали, поочередно).
5. При записи записать в 0-ю ячейку даты информацию о выводе, и, исходя из нее, прочесть еще нужное количество ячеек.

Основные файлы программы.

Файл main.cpp

```
#include "stack.h"
#include "sort.h"
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <pthread.h>

typedef struct
{
   int stor[1024];
   pthread_mutex_t mutex;
} shared_data;

static shared_data* data = NULL;

void initialise_shared()
{
   int prot = PROT_READ | PROT_WRITE;
   int flags = MAP_SHARED | MAP_ANONYMOUS;
   data = (shared_data*)mmap(NULL, sizeof(shared_data), prot, flags, -1, 0);
   if (data == MAP_FAILED) {
      exit(-3);
   }

   pthread_mutexattr_t attr;
   pthread_mutexattr_init(&attr);
   pthread_mutexattr_setpshared(&attr, PTHREAD_PROCESS_SHARED);
   pthread_mutex_init(&data->mutex, &attr);
}

int run_child()
{
   char c;
```

```
int st;
int val;
Stack *A[10];
for (int i = 0; i < 10; i++) {
   A[i] = NULL;
}
while (1){
   sleep(1);
   pthread_mutex_lock(&data->mutex);
   scanf("%c", &c);
   switch (c) {
      case 'c':
         scanf("%d", &st);
         if (st > 9 || st < 0) {
            data->stor[0] = -1;
            break;
         }
         A[st] = stack_create();
         data->stor[0] = 0;
         break;
      case 'd':
         scanf("%d", &st);
         if (st > 9 || st < 0) {
            data->stor[0] = -1;
            break;
         }
         if (A[st] != NULL) {
            stack_delete(&A[st]);
            data->stor[0] = 0;
         } else {
            data->stor[0] = -1;
         }
         break;
      case 'i':
         scanf("%d", &st);
         if (st > 9 || st < 0) {
            data->stor[0] = -1;
```

```
            break;
        }
        if (A[st] == NULL) {
            data->stor[0] = -1;
            break;
        }
        while (scanf("%d", &val)) {
            stack_push(A[st], val);
            c = getchar();
            if (c == '\n') {
                break;
            }
        }
        data->stor[0] = 0;
        break;
    case 'o':
        scanf("%d", &st);
        if (st > 9 || st < 0) {
            data->stor[0] = -1;
            break;
        }
        if (A[st] == NULL) {
            data->stor[0] = -1;
            break;
        }
        if (!stack_is_empty(A[st])) {
            data->stor[0] = 1;
            data->stor[1] = stack_pop(A[st]);
        } else {
            data->stor[0] = -1;
        }
        break;
    case 's':
        scanf("%d", &st);
        if (st > 9 || st < 0) {
            data->stor[0] = -1;
            break;
```

```
    }
    if (A[st] == NULL) {
        data->stor[0] = -1;
        break;
    }
    if (stack_size(A[st]) < 0) {
        data->stor[0] = -1;
        break;
    }
    sort(A[st]);
    data->stor[0] = 0;
    break;
case 'p':
    scanf("%d", &st);
    if (st > 9 || st < 0) {
        data->stor[0] = -1;
        break;
    }
    if (A[st] == NULL) {
        data->stor[0] = -1;
        break;
    }
    data->stor[0] = stack_size(A[st]);
    stack_print(A[st], data->stor);
    break;
case 'q':
    for (int i = 0; i < 10; i++) {
        if (A[i] != NULL) {
            stack_delete(&A[i]);
        }
    }
    data->stor[0] = -2;
    pthread_mutex_unlock(&data->mutex);
    return 0;
    break;
default:
    data->stor[0] = -1;
```

```
            break;
        }
        while(c != '\n') {
            c = getchar();
        }
        pthread_mutex_unlock(&data->mutex);
    }
}

void run_parent()
{
    while (true) {
        sleep(1);
        pthread_mutex_lock(&data->mutex);
        std::cout << "Mesage no: " << data->stor[0] << '\n';
        if (data->stor[0] == -2) {
            break;
        }
        for (int i = 1; i <= data->stor[0]; i++) {
            //read(pipeFdFromServ[0], &inMsg, sizeof(int));
            std::cout << data->stor[i] << '\n';
        }
        pthread_mutex_unlock(&data->mutex);
    }
}

int main(int argc, char** argv)
{
    initialise_shared();

    pid_t serverPid = fork();
    if (serverPid < 0) {
        std::cout << "Cannot create server procces\n";
        exit(-1);
    } else if (serverPid == 0) {
        std::cout << "Create server procces\n";
        run_child();
```

```
    } else {
        run_parent();
    }

    munmap(data, sizeof(data));
    return 0;
}
```

Пример работы программы

```
iPad-Alex:~/Documents/Study/Labs/OS/sem3os/2st# ./main.out
Create server procces
c1
Mesage no: 0
c3
Mesage no: 0
i 1 0 1 2 3 4 5 322
Mesage no: 0
i 3 8 5 4 0 -3 545
Mesage no: 0
i 2 3 4 33
Mesage no: -1
i1 6 7 8
Mesage no: 0
p1
Mesage no: 10
0
1
2
3
4
5
322
6
7
8
p
3
Mesage no: 6
8
5
4
 0
-3
545
s
3
Mesage no: 0
p
3
Mesage no: 6
-3
0
```

```
4
5
8
545
o
3
Mesage no: 1
545
o
3
Mesage no: 1
8
p
3
Mesage no: 4
-3
0
4
5
q
```

Вывод

В данной работе я научился использовать новый для меня способ общения процессов — через файл маппинг.. я считаю что это достаточно удобный способ, так как работа с уже готовым отображенным файлом происходит как с обычной внутрипрограммной переменной. С другой стороны у данного подхода есть и минусы по сравнению с пайпами, так как отображение файла не решает нам проблему критических участков памяти, и для контроля программы нам приходится использовать сторонние методы (к примеру мьютексы). Но все же это еще один инструмент, и в задачах которые того требуют, отображение файла может быть идеальным выбором.