Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №3 по курсу

«Операционные системы»

Группа: М8О-214Б-23

Студент: Гайдуков А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 12.12.24

Москва, 2024

Постановка задачи

Вариант 1.

Переделать первую лабораторную работу. Для общения между процессами необходимо использовать shared memory вместо pipe.

1 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс. Дочерний процесс является копией родительского процесса и начинает выполнение с той же точки, где был вызван fork().
- int shm_open(const char name, int oflag, mode_t mode); – создает или открывает объект разделяемой памяти. Возвращает файловый дескриптор, который может быть использован для работы с этим объектом.
- int ftruncate(int fd, off_t length); – изменяет размер файла или объекта разделяемой памяти, связанного с файловым дескриптором fd, до указанного размера length.
- void mmap(void addr, size_t length, int prot, int flags, int fd, off_t offset); – отображает файл или объект разделяемой памяти в адресное пространство процесса. Возвращает указатель на начало отображенной области.
- int munmap(void addr, size_t length); – удаляет отображение области памяти, начиная с адреса addr и размером length.
- sem_t sem_open(const char name, int oflag, mode_t mode, unsigned int value); – создает или открывает семафор с именем name. Возвращает указатель на семафор.
- int sem_wait(sem_t sem); – уменьшает значение семафора на 1. Если значение семафора равно 0, процесс блокируется до тех пор, пока семафор не станет больше 0.
- int sem_post(sem_t sem); – увеличивает значение семафора на 1. Если есть процессы, ожидающие этого семафора, один из них будет разблокирован.
- int sem_close(sem_t sem); – закрывает семафор, освобождая связанные с ним ресурсы.
- int sem_unlink(const char name); – удаляет семафор с именем name.
- int shm_unlink(const char name); – удаляет объект разделяемой памяти с именем name.
- int close(int fd); – закрывает файловый дескриптор fd, освобождая связанные с ним ресурсы.
- int execl(const char path, const char arg0, ... /, (char)0 /); – заменяет текущий образ процесса новым образом, загружаемым из файла, указанного в path. Аргументы передаются новому процессу.
- int open(const char pathname, int flags, mode_t mode); – открывает файл, указанный в pathname, с флагами flags и режимом доступа mode. Возвращает файловый дескриптор.

head.c:

1. Чтение входных данных:

- Если аргумент командной строки не передан, программа считывает имя файла из стандартного ввода.

- Затем программа считывает строку чисел из стандартного ввода.

2. Создание разделяемой памяти:

- Создается объект разделяемой памяти с именем `SHM_NAME` с помощью `shm_open`.

- Размер разделяемой памяти устанавливается с помощью `ftruncate`.

3. Отображение разделяемой памяти:

- Разделяемая память отображается в адресное пространство процесса с помощью `mmap`.

4. Создание семафора:

- Создается семафор с именем `SEM_NAME` с помощью `sem_open`. Начальное значение семафора — 0.

5. Создание дочернего процесса:

- С помощью `fork` создается дочерний процесс.

- В родительском процессе:

- Ожидается сигнал от семафора с помощью `sem_wait`.

- Данные из разделяемой памяти выводятся на экран.

- Освобождаются ресурсы: разделяемая память (`munmap`, `shm_unlink`) и семафор (`sem_close`, `sem_unlink`).

- В дочернем процессе:

- Запускается программа `summ` с помощью `execl`, передавая ей имя файла, имя разделяемой памяти и имя семафора.

6. Завершение программы:

- Родительский процесс завершает выполнение после вывода результата.

summ.c:

1. Проверка аргументов:

- Программа проверяет, что передано достаточно аргументов (имя файла, имя разделяемой памяти и имя семафора).

2. Чтение входных данных:

- Считывается строка чисел из стандартного ввода.

3. Открытие разделяемой памяти:

- Открывается объект разделяемой памяти с именем, переданным в аргументах, с помощью `shm_open`.

- Разделяемая память отображается в адресное пространство процесса с помощью `mmap`.

4. Открытие семафора:

- Открывается семафор с именем, переданным в аргументах, с помощью `sem_open`.

5. Вычисление суммы чисел:

- Функция `sum_of_strnums` вычисляет сумму чисел, переданных в строке.

- Результат преобразуется в строку с помощью `int_to_str`.

6. Запись результата в файл:

- Открывается файл с именем, переданным в аргументах, с помощью `open`.

- Результат записывается в файл с помощью `write`.

7. Запись результата в разделяемую память:

- Результат копируется в разделяемую память с помощью `strcpy`.

8. Сигнализация родительскому процессу:

- Семафор увеличивается с помощью `sem_post`, чтобы сообщить родительскому процессу, что данные готовы.

9. Освобождение ресурсов:

- Разделяемая память освобождается с помощью `munmap`.

- Семафор закрывается с помощью `sem_close`.

10. Завершение программы:

- Программа завершает выполнение.

Программа демонстрирует использование разделяемой памяти, семафоров и межпроцессного взаимодействия*для выполнения задачи. Родительский процесс управляет дочерним процессом, который выполняет вычисления и передает результат обратно.

Код программы

head.c

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <sys/wait.h>
#include <errno.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include "mpio.h"

#define BUFSIZ 8192
#define SHMEM_SIZE 8192

#define SHM_NAME "/my_shm"
#define SEM_NAME "/my_sem"

int main(int argc, char* argv[]) {
    char file_name[BUFSIZ];

    if (argc < 2) {
        stdin_read(file_name, BUFSIZ);
    } else {
        strcpy(file_name, argv[1]);
    }
```

```
    char str[BUFSIZ];
    stdin_read(str, BUFSIZ);

    pid_t p;

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        myWrite("shm_open() error!\n");
        return -1;
    }

    if (ftruncate(shm_fd, SHMEM_SIZE) == -1) {
        myWrite("ftruncate() error!\n");
        return -1;
    }

    char* shm_buf = (char*)mmap(NULL, SHMEM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_buf == MAP_FAILED) {
        myWrite("mmap() error!\n");
        return -1;
    }

    sem_t* sem = sem_open(SEM_NAME, O_CREAT, 0666, 0);
    if (sem == SEM_FAILED) {
        myWrite("sem_open() error!\n");
        return -1;
    }

    p = fork();
    if (p < 0) {
        myWrite("Fork failed!\n");
        return -1;
    } else if (p > 0) {
        // Parent process
        sem_wait(sem);
        myWrite(shm_buf);
        myWrite("\n");
        munmap(shm_buf, SHMEM_SIZE);
        close(shm_fd);
        shm_unlink(SHM_NAME);
        sem_close(sem);
        sem_unlink(SEM_NAME);
        return 0;
    } else {
        // Child process
        int ret = execl("./summ", "./summ", file_name, SHM_NAME, SEM_NAME, NULL);
        if (ret == -1) {
            char* sstr = strerror(errno);
```

```
            myWrite(sstr);
            return -1;
        }
    }

    munmap(shm_buf, SHMEM_SIZE);
    close(shm_fd);
    shm_unlink(SHM_NAME);
    sem_close(sem);
    sem_unlink(SEM_NAME);
    return 0;
}
```

summ.c

```
#include <stdlib.h>
#include <ctype.h>
#include <unistd.h>
#include <string.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include "mpio.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/shm.h>
#include <fcntl.h>

#define BUFSIZ 1024

int sum_of_strnums(char* str, int len) {
    char num[100];
    int n_len = 0;
    int res = 0;
    for (int i = 0; i < len; i++) {
        char c = str[i];
        if (isdigit(str[i]) || str[i] == '-') {
            num[n_len++] = str[i];
        } else if ((isspace(str[i]) && n_len != 0)) {
            num[n_len] = '\0';
            res += atoi(num);
            n_len = 0;
        } else if (iscntrl(str[i])) {
            num[n_len] = '\0';
            res += atoi(num);
            n_len = 0;
            break;
        } else {
```

```
            myWrite("Not a number detected!\n");
            myWrite(str + i);
            return 0;
        }
    }
    return res;
}

char* int_to_str(int number, char* string) {
    char rev_num[100];
    int len = 0;
    for (int i = 0; number != 0; i++) {
        rev_num[i] = '0' + number % 10;
        number /= 10;
        len = i + 1;
    }
    for (int i = len - 1, j = 0; i >= 0; i--, j++) {
        string[j] = rev_num[i];
    }
    string[len] = '\0';
    return string;
}

int main(int argc, char* argv[]) {
    if (argc < 3) {
        myWrite("Wrong amount of arguments!\n");
        return -1;
    }

    char buf[BUFSIZ];
    int len = stdin_read(buf, BUFSIZ);
    buf[len] = '\0';

    char filename[100];
    strcpy(filename, argv[1]);

    int shm_fd = shm_open(argv[2], O_RDWR, 0666);
    if (shm_fd == -1) {
        myWrite("shm_open() error!\n");
        return -1;
    }

    char* shm_buf = (char*)mmap(NULL, BUFSIZ, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_buf == MAP_FAILED) {
        myWrite("mmap() error!\n");
        return -1;
    }
```

```
    sem_t* sem = sem_open(argv[3], 0);
    if (sem == SEM_FAILED) {
        myWrite("sem_open() error!\n");
        return -1;
    }

    int sum = sum_of_strnums(buf, strlen(buf) + 1);
    char res[100];
    int_to_str(sum, res);

    int dtr = open(filename, O_CREAT | O_TRUNC | O_RDWR , S_IRUSR | S_IWUSR | S_IRGRP |
S_IWGRP| S_IROTH | S_IWOTH);
    if (dtr == -1) {
        myWrite("File error!\n");
        return -1;
    }

    strcpy(shm_buf, res);
    munmap(shm_buf, BUFSIZ);
    close(shm_fd);

    write(dtr, res, strlen(res));
    write(dtr, "\n", 1);
    close(dtr);

    sem_post(sem);
    sem_close(sem);

    return 0;
}
```

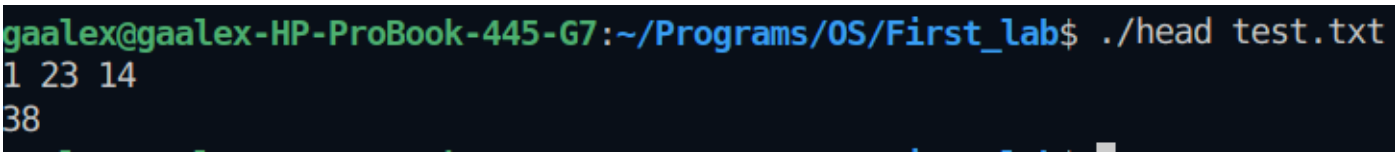
mpio.c

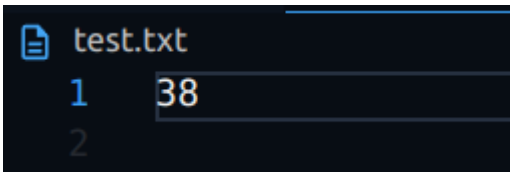
```
#ifndef MPIO
#define MPIO

#include <unistd.h>
#include <fcntl.h>
#include <string.h>

ssize_t stdin_read(void *buf, size_t cap)
{
       return read(STDIN_FILENO, buf, cap);
}

size_t stdout_write(const void *data, size_t len)
{
       return write(STDOUT_FILENO, data, len);
}
```

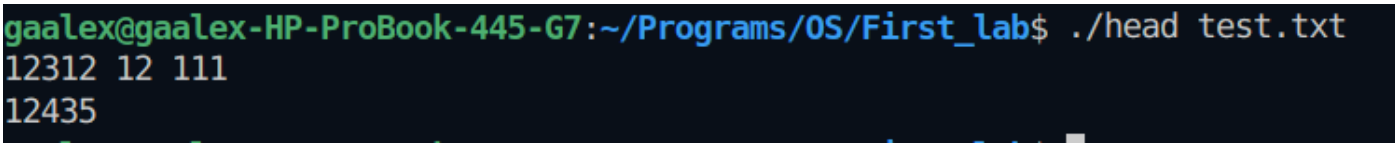
```
size_t myWrite(const void *data)
{
	return write(STDOUT_FILENO, (char*)data, strlen((char*)data));
}

int mp_fopen(char* filename, int flags, mode_t mode){
	int res = open(filename, flags, O_RDWR);
	if(!res){
		res = open(filename, flags, O_CREAT);
	}
	return res;
}

int mp_fclose(int fd){
	return close(fd);
}

#endif
```

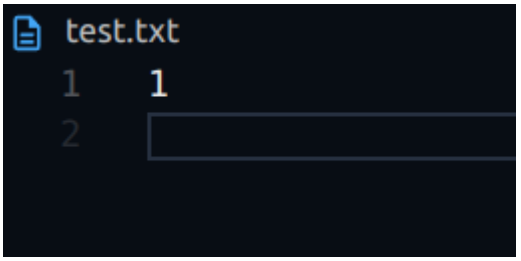
Протокол работы программы

Тестирование:









Strace:

```
strace ./head
execve("./head", ["./head"], 0x7ffde6a7fea0 /* 49 vars */) = 0
brk(NULL)                               = 0x557aeaaea000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffce7652a20) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd196f33000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=85427, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 85427, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd196f1e000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) =
832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48,
848) = 48
pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) =
68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64)
= 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd196cf5000
mprotect(0x7fd196d1d000, 2023424, PROT_NONE) = 0
mmap(0x7fd196d1d000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7fd196d1d000
mmap(0x7fd196eb2000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fd196eb2000
mmap(0x7fd196f0b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7fd196f0b000
mmap(0x7fd196f11000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fd196f11000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd196cf2000
arch_prctl(ARCH_SET_FS, 0x7fd196cf2740) = 0
set_tid_address(0x7fd196cf2a10)         = 5268
set_robust_list(0x7fd196cf2a20, 24)     = 0
rseq(0x7fd196cf30e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fd196f0b000, 16384, PROT_READ) = 0
mprotect(0x557aaf6f8000, 4096, PROT_READ) = 0
mprotect(0x7fd196f6d000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fd196f1e000, 85427)           = 0
read(0, 123
```

```
"123\n", 8192)                  = 4
read(0, 1 23 321 213
"1 23 321 213\n", 8192)         = 13
openat(AT_FDCWD, "/dev/shm/my_shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 8192)                      = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fd196f31000
openat(AT_FDCWD, "/dev/shm/sem.my_sem", O_RDWR|O_NOFOLLOW) = -1 ENOENT (Нет такого
файла или каталога)
getrandom("\x68\x58\x70\xdb\xd7\x4d\xee\x1b", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.wxGIFp", 0x7ffce764e720, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.wxGIFp", O_RDWR|O_CREAT|O_EXCL, 0666) = 4
write(4, "\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fd196f6c000
link("/dev/shm/sem.wxGIFp", "/dev/shm/sem.my_sem") = 0
newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
getrandom("\x31\x3a\xc3\xd9\xa8\x8f\x41\xfe", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x557aeaaea000
brk(0x557aeab0b000)                     = 0x557aeab0b000
unlink("/dev/shm/sem.wxGIFp")           = 0
close(4)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fd196cf2a10) = 5269
futex(0x7fd196f6c000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANY
) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=5269, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
write(1, "", 0)                         = 0
write(1, "\n", 1
)                       = 1
munmap(0x7fd196f31000, 8192)            = 0
close(3)                                = 0
unlink("/dev/shm/my_shm")               = 0
munmap(0x7fd196f6c000, 32)              = 0
unlink("/dev/shm/sem.my_sem")           = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Я научился писать приложения с несколькими процессами и с общением между ними при помощи shared memory. В ходе работы над лабораторной работой я не столкнулся с какими-либо проблемами.