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-211Б-23

Студент: Акимов К.К.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 19.12.24

Москва, 2024

Постановка задачи

Вариант 6.

Постановка задачи Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
- pid_t getpid(void); – возвращает ID вызывающего процесса.
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- int dup2(int __fd, int __fd2); – копирует FD в FD2, закрыв FD2 если это требуется.
- int execv(const char *__path, char *const *__argv); – заменяет образ текущего процесса на образ нового процесса, определённого в пути path.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
- pid_t wait(int *__stat_loc); – используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.
- int shm_open(const char *name, int oflag, mode_t mode); – создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
- int shm_unlink(const char *name); – удаляется имя объекта разделяемой памяти и, как только все процессы завершили работу с объектом и отменили его распределение, очищают пространство и уничтожают связанную с ним область памяти.
- void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.

- int ftruncate(int fd, off_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
- int sem_wait(sem_t *sem); – уменьшает значение семафора на 1. Если семафор в данный момент имеет нулевое значение, то вызов блокируется до тех пор, пока либо не станет возможным выполнить уменьшение.
- int sem_post(sem_t *sem); – увеличивает значение семафора на 1.
- int sem_destroy(sem_t *sem); - уничтожает безымянный семафор, расположенный по адресу sem

Для выполнения данной лабораторной работы я изучил указанные выше системные вызовы, а также пример выполнения подобного задания.

Программа parent.c принимает путь к файлу с числами типа int в качестве аргумента командной строки. Открывается файл с помощью fopen() в режиме чтения, создается область разделяемой памяти с помощью shm_open() и ftruncate(), а также два семафора для синхронизации: sem_write (управляет записью данных) и sem_read (управляет чтением).

С помощью fork() создается дочерний процесс. Если это родитель, он читает строки из файла с помощью fgets(), копирует их в разделяемую память и сигнализирует дочернему процессу через sem_read. После завершения ввода родитель записывает пустую строку в память для сигнала об окончании и ждет завершения дочернего процесса через wait().

Дочерний процесс открывает разделяемую память и подключается к семафорам. Он ждет данных через sem_read, обрабатывает строки, суммируя числа, и выводит результат. Невалидные строки или числа вызывают сообщение об ошибке, но программа продолжает свое выполнение. После получения пустой строки дочерний процесс завершает работу, закрывая разделяемую память и семафоры.

Все системные вызовы проверяются на ошибки, ресурсы корректно освобождаются с помощью shm_unlink() и sem_unlink() после завершения программы.

Код программы

Parent.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/wait.h>
#include <limits.h>

#define SHM_NAME "/shared_memory"
#define SEM_WRITE "/sem_write"
#define SEM_READ "/sem_read"
#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 2) {
        write(STDERR_FILENO, "Error, there should be 2 arguments here\n", 40);
        exit(EXIT_FAILURE);
```

```
    }

    FILE *file = fopen(argv[1], "r");
    if (!file) {
        write(STDERR_FILENO, "Error with open file\n", 21);
        exit(EXIT_FAILURE);
    }

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        write(STDERR_FILENO, "Error creating shared memory\n", 29);
        fclose(file);
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, BUF_SIZE) == -1) {
        write(STDERR_FILENO, "Error setting shared memory size\n", 33);
        fclose(file);
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    char *shm_ptr = mmap(0, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        write(STDERR_FILENO, "Error mapping shared memory\n", 28);
        fclose(file);
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    sem_t *sem_write = sem_open(SEM_WRITE, O_CREAT, 0666, 1);
    sem_t *sem_read = sem_open(SEM_READ, O_CREAT, 0666, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        write(STDERR_FILENO, "Error creating semaphores\n", 26);
        fclose(file);
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid == 0) {
        // Child
        execl("./child", "./child", NULL);
        write(STDERR_FILENO, "Error executing child process\n", 30);
        exit(EXIT_FAILURE);
    } else if (pid > 0) {
        // Parent
        char buffer[BUF_SIZE];
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
            sem_wait(sem_write);
            strncpy(shm_ptr, buffer, BUF_SIZE);
            sem_post(sem_read);
        }
        // End of input
        sem_wait(sem_write);
        shm_ptr[0] = '\0';
```

```
        sem_post(sem_read);

        wait(NULL);
        fclose(file);

    } else {
        write(STDERR_FILENO, "Error with fork\n", 16);
        fclose(file);
        shm_unlink(SHM_NAME);
        sem_unlink(SEM_WRITE);
        sem_unlink(SEM_READ);
        exit(EXIT_FAILURE);
    }

    munmap(shm_ptr, BUF_SIZE);
    shm_unlink(SHM_NAME);
    sem_unlink(SEM_WRITE);
    sem_unlink(SEM_READ);
    return 0;
}
```

Child.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <unistd.h>
#include <limits.h>

#define SHM_NAME "/shared_memory"
#define SEM_WRITE "/sem_write"
#define SEM_READ "/sem_read"
#define BUF_SIZE 1024

enum Errors {
    OK,
    ERROR
};

enum Errors Str_to_int(char *str, int *answer, int line) {
    char *endptr;
    long number = strtol(str, &endptr, 10);

    if (number > INT_MAX || number < INT_MIN || *endptr != '\0') {
        fprintf(stderr, "Error, incorrect value in %d line\n", line);
        return ERROR;
    }
    *answer = (int) number;
    return OK;
}

void remove_carriage_return(char *str) {
    char *ptr = strchr(str, '\r');
    if (ptr) *ptr = '\0';
}
```

```c
int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("Error opening shared memory");
        exit(EXIT_FAILURE);
    }

    char *shm_ptr = mmap(0, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("Error mapping shared memory");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_write = sem_open(SEM_WRITE, 0);
    sem_t *sem_read = sem_open(SEM_READ, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        perror("Error opening semaphores");
        exit(EXIT_FAILURE);
    }

    int line = 1;
    while (1) {
        sem_wait(sem_read);
        if (shm_ptr[0] == '\0') {
            break;
        }

        remove_carriage_return(shm_ptr);  // Удаляем '\r' перед обработкой
        char *token = strtok(shm_ptr, " ");
        int line_sum = 0;
        int valid_line = 1;

        while (token != NULL) {
            int num;
            if (Str_to_int(token, &num, line) != OK) {
                valid_line = 0;
                break;
            }
            line_sum += num;
            token = strtok(NULL, " ");
        }

        if (valid_line) {
            printf("Sum in %d line = %d\n", line, line_sum);
        }

        line++;
        sem_post(sem_write);
    }

    munmap(shm_ptr, BUF_SIZE);
    sem_close(sem_write);
    sem_close(sem_read);
    return 0;
}
```

Протокол работы программы

```
kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ gcc -o parent parent.c
kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ gcc -o child child.c
kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ ./parent file.txt
Sum in 1 line = 10
Sum in 2 line = 11
Sum in 3 line = 12
Error, incorrect value in 4 line
Sum in 5 line = 14
Sum in 6 line = 15
kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ strace ./parent file.txt
execve("./parent", ["./parent", "file.txt"], 0x7fff3f13e248 /* 28 vars */) = 0
brk(NULL)                               = 0x55df1ddba000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fad75da5000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=20231, ...}) = 0
mmap(NULL, 20231, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fad75da0000
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\0\220\243\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\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fad75b8e000
mmap(0x7fad75bb6000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fad75bb6000
mmap(0x7fad75d3e000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7fad75d3e000
mmap(0x7fad75d8d000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fad75d8d000
mmap(0x7fad75d93000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fad75d93000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fad75b8b000
arch_prctl(ARCH_SET_FS, 0x7fad75b8b740) = 0
set_tid_address(0x7fad75b8ba10)         = 614
set_robust_list(0x7fad75b8ba20, 24)     = 0
rseq(0x7fad75b8c060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fad75d8d000, 16384, PROT_READ) = 0
mprotect(0x55df1cbf5000, 4096, PROT_READ) = 0
mprotect(0x7fad75ddd000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fad75da0000, 20231)           = 0
getrandom("\x90\xfc\x76\xf4\x31\x22\xd1\x6c", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55df1ddba000
brk(0x55df1dddb000)                     = 0x55df1dddb000
openat(AT_FDCWD, "file.txt", O_RDONLY)  = 3
openat(AT_FDCWD, "/dev/shm/shared_memory", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC,
0666) = 4
ftruncate(4, 1024)                      = 0
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fad75da4000
openat(AT_FDCWD, "/dev/shm/sem.sem_write", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT
(No such file or directory)
getrandom("\xaf\x08\x9a\x86\xe9\xdb\xb8\x03", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.twWXNJ", 0x7ffe67586070, AT_SYMLINK_NOFOLLOW) = -1
```

```
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.twWXNJ",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\1\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", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fad75da3000
link("/dev/shm/sem.twWXNJ", "/dev/shm/sem.sem_write") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.twWXNJ")         = 0
close(5)                              = 0
openat(AT_FDCWD, "/dev/shm/sem.sem_read", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT
(No such file or directory)
getrandom("\x60\x41\xc2\xf4\xdb\x09\xca\x4a", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.yMpJR2", 0x7ffe67586070, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.yMpJR2",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 5
write(5, "\0\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", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fad75da2000
link("/dev/shm/sem.yMpJR2", "/dev/shm/sem.sem_read") = 0
fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0
unlink("/dev/shm/sem.yMpJR2")         = 0
close(5)                              = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fad75b8ba10) = 615
fstat(3, {st_mode=S_IFREG|0777, st_size=66, ...}) = 0
read(3, "2 3 5\r\n1 1 9\r\n12 0 0\r\n16 -2 -1 D"..., 4096) = 66
futex(0x7fad75da3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANYSum in 1 line = 10
) = 0
futex(0x7fad75da2000, FUTEX_WAKE, 1Sum in 2 line = 11
)    = 1
futex(0x7fad75da2000, FUTEX_WAKE, 1Sum in 3 line = 12
)    = 1
futex(0x7fad75da2000, FUTEX_WAKE, 1Error, incorrect value in 4 line
)    = 1
futex(0x7fad75da2000, FUTEX_WAKE, 1)    = 1
Sum in 5 line = 14
read(3, "", 4096)                     = 0
futex(0x7fad75da2000, FUTEX_WAKE, 1Sum in 6 line = 15
)    = 1
futex(0x7fad75da2000, FUTEX_WAKE, 1)    = 1
wait4(-1, NULL, 0, NULL)              = 615
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=615, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
close(3)                              = 0
munmap(0x7fad75da4000, 1024)          = 0
unlink("/dev/shm/shared_memory")      = 0
unlink("/dev/shm/sem.sem_write")      = 0
unlink("/dev/shm/sem.sem_read")       = 0
exit_group(0)                         = ?
+++ exited with 0 +++
kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$
```

Вывод

В процессе выполнения данной лабораторной работы я изучил новые системные вызовы на языке Си, которые позволяют эффективно работать с разделяемой памятью и семафорами. Освоил передачу данных между процессами через shared memory и управление доступом с использованием семафоров.