

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-209БВ-24

Студент: Галич А.П.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 10.12.25

Москва 2025

Постановка задачи

Вариант 3.

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы

mmap() – отображает файл в память для межпроцессного взаимодействия

fork() – создает дочерний процесс

ftruncate() – устанавливает размеры файла

open – открывает/создает файл

waintpid() – проверяет завершение дочернего процесса

kill – отправляет сигналы между процессами

pause – приостанавливает до получения сигнала

execl – запускает программу child

munmap – освобождает отображенную память

fgets – чтение ввода пользователя

sscanf – парсинг чисел из строки

Алгоритм работы:

Родительский процесс начинает выполнение с запроса имени файла для сохранения результатов у пользователя. После получения имени файла он создаёт и инициализирует файл shared.dat для разделяемой памяти с помощью системных вызовов open и ftruncate, затем отображает этот файл в своё адресное пространство при помощи mmap с флагом MAP_SHARED. Далее родитель создаёт дочерний процесс через вызов fork, который немедленно заменяет свой образ на программу child с помощью execl, передавая в качестве

аргументов имя файла разделяемой памяти, имя выходного файла и собственный идентификатор процесса.

Дочерний процесс инициализирует свою работу, открывая файл разделяемой памяти в режиме только для чтения и отображая его в память через mmap. Он также открывает указанный выходной файл для записи результатов. Затем процесс устанавливает обработчики сигналов SIGUSR1 для обработки новых данных и SIGTERM для аварийного завершения, после чего переходит в состояние ожидания сигналов, используя системный вызов pause.

Родительский процесс входит в цикл обработки пользовательского ввода. Каждую введённую строку чисел он копирует в область разделяемой памяти и отправляет дочернему процессу сигнал SIGUSR1, уведомляя о готовности данных для обработки. После отправки сигнала родитель немедленно проверяет состояние дочернего процесса неблокирующим вызовом waitpid с флагом WNOHANG, что позволяет ему продолжать работу без ожидания.

Дочерний процесс, получив сигнал SIGUSR1, читает строку чисел из разделяемой памяти и выполняет цепочное деление первого числа на все последующие. Ход вычислений и результат записываются в выходной файл. При обнаружении деления на ноль дочерний процесс фиксирует ошибку в файле, отправляет родительскому процессу сигнал SIGTERM и завершает свою работу.

Родительский процесс, получив сигнал SIGTERM или обнаружив завершение дочернего процесса через waitpid, выходит из цикла обработки ввода. Перед завершением родитель освобождает ресурсы: отменяет отображение разделяемой памяти с помощью mmap, закрывает файловый дескриптор и ожидает окончательного завершения дочернего процесса через wait, обеспечивая корректное завершение работы всей системы.

Код программы

parent.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <string.h>
#include <signal.h>

#define SHARED_SIZE 4096

int main() {
    char buffer[1024];
    char outfile[100];
```

```
printf("Введите имя файла для результатов: ");
fgets(outfile, sizeof(outfile), stdin);
outfile[strcspn(outfile, "\n")] = 0;

FILE* test = fopen(outfile, "w");
if (!test) {
    perror("fopen");
    exit(1);
}
fclose(test);

int fd = open("shared.dat", O_CREAT | O_RDWR, 0666);
if (fd < 0) {
    perror("open");
    exit(1);
}

ftruncate(fd, SHARED_SIZE);

char* shared_data = mmap(NULL, SHARED_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
0);
if (shared_data == MAP_FAILED) {
    perror("mmap");
    close(fd);
    exit(1);
}

pid_t pid = fork();

if (pid == 0) {
    char parent_pid_str[20];
    sprintf(parent_pid_str, "%d", getppid());

    execl("./child", "child", "shared.dat", outfile, parent_pid_str, NULL);
    perror("execl");
    exit(1);
}

printf("Введите числа:\n");

while (1) {
    if (fgets(buffer, sizeof(buffer), stdin) == NULL)
        break;

    strcpy(shared_data, buffer);

    kill(pid, SIGUSR1);

    int status;
    pid_t res = waitpid(pid, &status, WNOHANG);
    if (res != 0) {
        printf("Дочерний процесс завершён.\n");
        break;
    }
}
```

```
        }

        munmap(shared_data, SHARED_SIZE);
        close(fd);

        wait(NULL);
        return 0;
    }
}
```

child.c

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

#define SHARED_SIZE 4096

static char* shared_data = NULL;
static int should_exit = 0;
static FILE* outfile = NULL;
static pid_t parent_pid = 0;

void handle_signal(int sig) {
    if (sig == SIGTERM) {
        should_exit = 1;
        return;
    }

    if (sig != SIGUSR1 || !shared_data) return;

    char buffer[SHARED_SIZE];
    strcpy(buffer, shared_data);

    int first, num;
    int result;
    int count = 0;

    char* ptr = buffer;

    if (sscanf(ptr, "%d", &first) != 1)
        return;

    result = first;
    count++;

    fprintf(outfile, "%d", first);
```

```
while (*ptr && *ptr != ' ') ptr++;
while (*ptr == ' ') ptr++;

while (sscanf(ptr, "%d", &num) == 1) {
    fprintf(outfile, " / %d", num);

    if (num == 0) {
        fprintf(outfile, " - Деление на 0! Завершение работы.\n");
        fflush(outfile);

        // Завершаем родителя
        kill(parent_pid, SIGTERM);
        should_exit = 1;
        return;
    }

    result /= num;
    count++;

    while (*ptr && *ptr != ' ') ptr++;
    while (*ptr == ' ') ptr++;
}

if (count > 1)
    fprintf(outfile, " = %d\n", result);
else
    fprintf(outfile, "\n");

fflush(outfile);
}

int main(int argc, char* argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: child <shared_file> <output_file> <parent_pid>\n");
        return 1;
    }

    const char* shared_name = argv[1];
    const char* outname = argv[2];
    parent_pid = atoi(argv[3]);

    outfile = fopen(outname, "w");
    if (!outfile) {
        perror("fopen");
        return 1;
    }

    int fd = open(shared_name, O_RDONLY);
    if (fd < 0) {
        perror("open shared file");
        fclose(outfile);
        return 1;
    }
```

```

shared_data = mmap(NULL, SHARED_SIZE, PROT_READ, MAP_SHARED, fd, 0);
if (shared_data == MAP_FAILED) {
    perror("mmap");
    close(fd);
    fclose(outfile);
    return 1;
}

close(fd);

signal(SIGUSR1, handle_signal);
signal(SIGTERM, handle_signal);

while (!should_exit)
    pause();

munmap(shared_data, SHARED_SIZE);
fclose(outfile);

return 0;
}

```

Протокол работы программы

```

kishaki@416:~/lab_OS3$ ./parent
Введите имя файла для результатов: result.txt
Введите числа:
100 2 5
100 2
100 0
Terminated

```

```

result.txt
1 100 / 2 / 5 = 10
2 100 / 2 = 50
3 100 / 0 - Деление на 0! Завершение работы.
4

```

Тестирование:

```

kishaki@416:~/lab_OS3$ strace -f ./parent
execve("./parent", ["/./parent"], 0x7ffd7d08b1e8 /* 29 vars */) = 0
brk(NULL) = 0x5b716582a000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7b5cc87b4000
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

```


[pid 42820] pread64(4, "\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\0\0...", 784, 64) = 784

[pid 42820] fstat(4, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 42820] pread64(4, "\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...", 784, 64) = 784

[pid 42820] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 0x7e1979200000

[pid 42820] mmap(0x7e1979228000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x7e1979228000

[pid 42820] mmap(0x7e19793b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1b0000) = 0x7e19793b0000

[pid 42820] mmap(0x7e19793ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1fe000) = 0x7e19793ff000

[pid 42820] mmap(0x7e1979405000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7e1979405000

[pid 42820] close(4) = 0

[pid 42820] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e19795cd000

[pid 42820] arch_prctl(ARCH_SET_FS, 0x7e19795cd740) = 0

[pid 42820] set_tid_address(0x7e19795cda10) = 42820

[pid 42820] set_robust_list(0x7e19795cda20, 24) = 0

[pid 42820] rseq(0x7e19795ce060, 0x20, 0, 0x53053053) = 0

[pid 42820] mprotect(0x7e19793ff000, 16384, PROT_READ) = 0

[pid 42820] mprotect(0x5c0d63ee0000, 4096, PROT_READ) = 0

[pid 42820] mprotect(0x7e1979612000, 8192, PROT_READ) = 0

[pid 42820] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 42820] munmap(0x7e19795d0000, 37043) = 0

[pid 42820] getrandom("\x97\xc7\xc1\x9d\x32\x77\x08\xcc", 8, GRND_NONBLOCK) = 8

[pid 42820] brk(NULL) = 0x5c0d74601000

[pid 42820] brk(0x5c0d74622000) = 0x5c0d74622000

[pid 42820] openat(AT_FDCWD, "result.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4

[pid 42820] openat(AT_FDCWD, "shared.dat", O_RDONLY) = 5

[pid 42820] mmap(NULL, 4096, PROT_READ, MAP_SHARED, 5, 0) = 0x7e19795d9000

[pid 42820] close(5) = 0

```
[pid 42820] rt_sigaction(SIGUSR1, {sa_handler=0x5c0d63ede369, sa_mask=[USR1],  
sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7e1979245330}, {sa_handler=SIG_DFL,  
sa_mask=[], sa_flags=0}, 8) = 0  
  
[pid 42820] rt_sigaction(SIGTERM, {sa_handler=0x5c0d63ede369, sa_mask=[TERM],  
sa_flags=SA_RESTORER|SA_RESTART, sa_restorer=0x7e1979245330}, {sa_handler=SIG_DFL,  
sa_mask=[], sa_flags=0}, 8) = 0  
  
[pid 42820] pause(100 2 5  
<unfinished ...>  
  
[pid 42807] <... read resumed>"100 2 5\n", 1024) = 8  
  
[pid 42807] kill(42820, SIGUSR1)      = 0  
  
[pid 42820] <... pause resumed>      = ? ERESTARTNOHAND (To be restarted if no handler)  
  
[pid 42807] wait4(42820, <unfinished ...>  
  
[pid 42820] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=42807, si_uid=1000}  
---  
  
[pid 42807] <... wait4 resumed>0x7ffde6380780, WNOHANG, NULL) = 0  
  
[pid 42807] read(0, <unfinished ...>  
  
[pid 42820] fstat(4, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0  
  
[pid 42820] write(4, "100 / 2 / 5 = 10\n", 17) = 17  
  
[pid 42820] rt_sigreturn({mask=[]})    = -1 EINTR (Interrupted system call)  
  
[pid 42820] pause(10 2 5  
<unfinished ...>  
  
[pid 42807] <... read resumed>"10 2 5\n", 1024) = 7  
  
[pid 42807] kill(42820, SIGUSR1)      = 0  
  
[pid 42820] <... pause resumed>      = ? ERESTARTNOHAND (To be restarted if no handler)  
  
[pid 42807] wait4(42820, 0x7ffde6380780, WNOHANG, NULL) = 0  
  
[pid 42820] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=42807, si_uid=1000}  
---  
  
[pid 42807] read(0, <unfinished ...>  
  
[pid 42820] write(4, "10 / 2 / 5 = 1\n", 15) = 15  
  
[pid 42820] rt_sigreturn({mask=[]})    = -1 EINTR (Interrupted system call)  
  
[pid 42820] pause(10 0  
<unfinished ...>  
  
[pid 42807] <... read resumed>"10 0\n", 1024) = 5  
  
[pid 42807] kill(42820, SIGUSR1)      = 0  
  
[pid 42820] <... pause resumed>      = ? ERESTARTNOHAND (To be restarted if no handler)
```

```
[pid 42807] wait4(42820, 0x7ffde6380780, WNOHANG, NULL) = 0
[pid 42820] --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=42807, si_uid=1000}
---
[pid 42807] read(0, <unfinished ...>
[pid 42820] write(4, "10 / 0 - \320\224\320\265\320\273\320\265\320\275\320\270\320\265
\320\275\320\260 0! "..., 67) = 67
[pid 42820] kill(42807, SIGTERM <unfinished ...>
[pid 42807] <... read resumed>0x5b716582a6b0, 1024) = ? ERESTARTSYS (To be restarted if
SA_RESTART is set)
[pid 42820] <... kill resumed>)      = 0
[pid 42807] --- SIGTERM {si_signo=SIGTERM, si_code=SI_USER, si_pid=42820,
si_uid=1000} ---
[pid 42820] rt_sigreturn({ mask=[] })   = -1 EINTR (Interrupted system call)
[pid 42807] +++ killed by SIGTERM +++
munmap(0x7e19795d9000, 4096)      = 0
close(4)                          = 0
exit_group(0)                     = ?
+++ exited with 0 +++
Terminated
```

Вывод

В ходе выполнения лабораторной работы была успешно реализована система межпроцессного взаимодействия, основанная на разделяемой памяти и сигналах. Родительский процесс и дочерний процесс организовали эффективный конвейер обработки данных, где родитель отвечает за ввод числовых выражений от пользователя, а дочерний процесс выполняет их вычисление.