

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»

Выполнила: Ю. В. Павлова
Группа: М8О-207БВ-24
Преподаватель: Е. С. Миронов

Москва, 2025

Условие

Цель работы: Приобретение практических навыков в:

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

Задание: Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программы (основной процесс) должен создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант: 3

Описание варианта: Пользователь вводит команды вида: «число число число<конец строки»». 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.

Группа вариантов 1

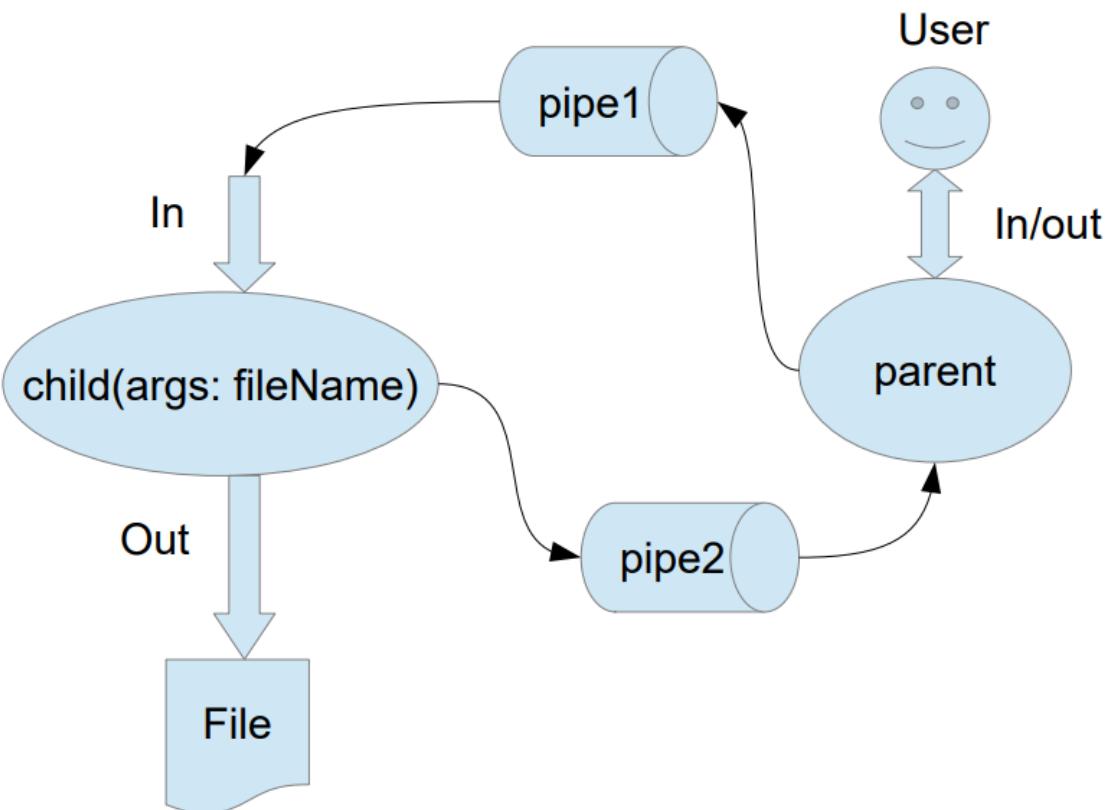


Рис. 1: Схема взаимодействия процессов

Метод решения

Общее описание алгоритма

Программа состоит из двух процессов: родительского (parent) и дочернего (child), которые взаимодействуют между собой с использованием:

- Общей памяти (shared memory) для передачи данных
- Сигналов для синхронизации работы

Алгоритм работы:

1. Родительский процесс создает общую память и запускает дочерний процесс
2. Родительский процесс запрашивает у пользователя ввод чисел
3. Введенные данные записываются в общую память
4. Родитель посыпает сигнал WORK дочернему процессу
5. Дочерний процесс, получив сигнал, читает данные из общей памяти
6. Дочерний процесс выполняет деление первого числа на последующие
7. Если обнаружено деление на ноль, дочерний процесс отправляет SIGTERM родителю и завершается
8. Результаты записываются в файл
9. Дочерний процесс отправляет сигнал CONFIRM родителю
10. Процесс повторяется до ввода команды "exit" или обнаружения деления на ноль

Для реализации использованы системные вызовы POSIX для работы с процессами, сигналами и общей памятью. Данный подход обеспечивает надежное межпроцессное взаимодействие с синхронизацией через сигналы и передачей данных через общую память.

Описание программы

Структура проекта

Проект состоит из следующих компонентов:

- `app/` - содержит главные файлы приложений (`parent_main.cpp`, `child_main.cpp`)
- `include/` - содержит заголовочные файлы (`child.h`, `os.h`, `parent.h`)
- `src/` - содержит реализацию классов (`child.cpp`, `os.cpp`, `parent.cpp`)
- `CMakeLists.txt` - файл для сборки проекта

Основные типы данных

- `os::ProcessHandle` - тип для представления идентификатора процесса (`pid_t`)
- `os::FileHandle` - тип для представления дескриптора файла (`int`)
- `os::SignalHandler` - тип указателя на функцию обработки сигналов
- `os::SharedMemory` - структура для работы с общей памятью, содержит указатель на память, размер, дескриптор файла и имя

Основные функции

- `os::CreateChildProcess()` - создает дочерний процесс
- `os::OpenShM()` и `os::CreateShM()` - работа с общей памятью
- `os::SetSignalHandler()` - установка обработчика сигналов
- `os::SendSignal()` - отправка сигнала процессу
- `os::WaitSignal()` - ожидание сигнала
- `Parent::CreateChild()` - создание дочернего процесса
- `Parent::Input()` - обработка пользовательского ввода
- `Child::ProcessDivision()` - выполнение деления и запись результатов

Результаты

Программа успешно реализует поставленную задачу. При запуске:

1. Родительский процесс запрашивает имя файла для вывода результатов
2. Затем запрашивает у пользователя числа для деления
3. Передает данные в дочерний процесс через общую память
4. Дочерний процесс выполняет вычисления и записывает результаты в файл

Пример работы:

- Пользователь вводит: 10 2 5
- В файл записывается:

```
Child: Запущен. Файл вывода: output.txt
10 / 2 = 5
10 / 5 = 2
---
```

- При вводе: 10 0 5
- В файл записывается:

```
Child: Запущен. Файл вывода: output.txt
КРИТИЧЕСКАЯ ОШИБКА: Деление на ноль!
Child: Получен SIGTERM. Завершение работы.
```

Выводы

В ходе выполнения лабораторной работы я освоила организацию взаимодействия между процессами в Linux с использованием общей памяти и сигналов. Основной задачей было реализовать передачу чисел от родительского процесса дочернему для выполнения операции деления с записью результатов в файл. Для обмена данными была применена технология POSIX shared memory, позволяющая процессам работать с одной областью памяти без копирования данных. Синхронизация выполнялась через пользовательские сигналы SIGUSR1 и SIGUSR2, что обеспечивало корректную последовательность операций. Особое внимание было уделено обработке критических ситуаций, таких как деление на ноль, с реализацией механизма аварийного завершения обоих процессов.

Исходная программа

parent_main.cpp

```
1 #include "parent.h"
2 #include <iostream>
3
4 int main() {
5     std::string filename;
6     std::cout << "Parent:                                     :" << std::endl;
7     std::getline(std::cin, filename);
8
9     if (filename.empty()) return 1;
10
11    parent::Parent p;
12    p.CreateChild(filename);
13    p.Input();
14    p.EndChild();
15
16    return 0;
17 }
```

Листинг 1: Файл parent_main.cpp

child_main.cpp

```
1 #include "child.h"
2 #include <iostream>
3 #include <string>
4
5 int main(int argc, char* argv[]) {
6     if (argc < 3) {
7         std::cerr << "Child:                                     ." << std::endl;
8         return 1;
9     }
10
11    child::Child c;
12    c.Input();
13    c.Calc();
14    c.Output();
15
16    return 0;
17 }
```

```

9 }
10
11 std::string filename = argv[1];
12 std::string shm_name = argv[2];
13
14 child::Child child_proc(filename, shm_name);
15 child_proc.ProcessDivision();
16 return 0;
17 }
```

Листинг 2: Файл child_main.cpp

os.cpp

```

1 #include "os.h"
2 #include <unistd.h>
3 #include <sys/wait.h>
4 #include <sys/mman.h>
5 #include <sys/stat.h>
6 #include <fcntl.h>
7 #include <signal.h>
8 #include <cstdlib>
9 #include <cstdio>
10 #include <iostream>
11
12 namespace os {
13
14     volatile sig_atomic_t signal_received = 0;
15     volatile sig_atomic_t terminated = 0;
16
17     void DefaultSignalHandler(int) {
18         signal_received = 1;
19     }
20
21     void TerminateHandler(int signum) {
22         terminated = 1;
23         signal_received = 1;
24     }
25
26     int CreateChildProcess(const std::string& exe_name, const std::string&
filename, const std::string& shm_name) {
27         pid_t pid = fork();
28         if (pid == -1) {
29             perror("fork failed");
30             return -1;
31         }
32
33         if (pid == 0) {
34             execl(exe_name.c_str(), exe_name.c_str(), filename.c_str(),
shm_name.c_str(), nullptr);
35             perror("execl failed");
36             Exit(1);
37         }
38
39         return pid;
40     }
41
42     void WaitForChild(ProcessHandle process) {
43         if (process > 0) {
```

```

44         waitpid(process, nullptr, 0);
45     }
46 }
47
48 ProcessHandle GetParentPID() {
49     return getppid();
50 }
51
52 SharedMemory CreateShM(const std::string& name, size_t size) {
53     shm_unlink(name.c_str());
54     int fd = shm_open(name.c_str(), O_CREAT | O_RDWR, 0666);
55     if (fd == -1) {
56         perror("shm create failed");
57         Exit(1);
58     }
59     if (ftruncate(fd, size) == -1) {
60         perror("ftruncate failed");
61         Exit(1);
62     }
63
64     char* ptr = (char*)mmap(0, size, PROT_READ | PROT_WRITE,
65     MAP_SHARED, fd, 0);
66     if (ptr == MAP_FAILED) {
67         perror("mmap failed");
68         Exit(1);
69     }
70
71     return {ptr, size, fd, name};
72 }
73
74 SharedMemory OpenShM(const std::string& name, size_t size) {
75     int fd = -1;
76     int attempts = 0;
77     const int max_attempts = 10;
78
79     while (attempts < max_attempts) {
80         fd = shm_open(name.c_str(), O_RDWR, 0666);
81         if (fd != -1) break;
82
83         if (errno == ENOENT) {
84             usleep(100000);
85             attempts++;
86         } else {
87             perror("shm open failed");
88             Exit(1);
89         }
90     }
91     if (fd == -1) {
92         perror("shm open failed");
93         Exit(1);
94     }
95
96     char* ptr = (char*)mmap(0, size, PROT_READ | PROT_WRITE,
97     MAP_SHARED, fd, 0);
98     if (ptr == MAP_FAILED) {
99         perror("mmap failed");
100        Exit(1);
101    }
102
103    return {ptr, size, fd, name};

```

```
102 }
103
104 void UnmapShM(SharedMemory& shm) {
105     if (shm.ptr) {
106         munmap(shm.ptr, shm.size);
107     }
108     if (shm.fd != -1) {
109         close(shm.fd);
110     }
111 }
112
113 void DestroyShM(SharedMemory& shm) {
114     UnmapShM(shm);
115     shm_unlink(shm.name.c_str());
116 }
117
118 void SendSignal(ProcessHandle pid, int signum) {
119     kill(pid, signum);
120 }
121
122 void WaitSignal() {
123     while (!signal_received && !terminated) {
124         pause();
125     }
126     signal_received = 0;
127 }
128
129 void SetSignalHandler(int signum, SignalHandler handler) {
130     struct sigaction sa;
131     sa.sa_handler = handler;
132     sigemptyset(&sa.sa_mask);
133     sa.sa_flags = 0;
134     sigaction(signum, &sa, nullptr);
135 }
136
137 void Exit(int code) {
138     _exit(code);
139 }
140 }
```

Листинг 3: Файл os.cpp

strace output


```

119 10241 close(3) = 0
120 10241 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
   -1, 0) = 0x7ff8a9adf000
121 10241 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
   -1, 0) = 0x7ff8a9adc000
122 10241 arch_prctl(ARCH_SET_FS, 0x7ff8a9adc740) = 0
123 10241 set_tid_address(0x7ff8a9adca10) = 10241
124 10241 set_robust_list(0x7ff8a9adca20, 24) = 0
125 10241 rseq(0x7ff8a9add060, 0x20, 0, 0x53053053) = 0
126 10241 mprotect(0x7ff8a9dc9000, 16384, PROT_READ) = 0
127 10241 mprotect(0x7ff8a9bc8000, 4096, PROT_READ) = 0
128 10241 mprotect(0x7ff8a9e08000, 4096, PROT_READ) = 0
129 10241 mprotect(0x7ff8aa076000, 45056, PROT_READ) = 0
130 10241 mprotect(0x55b0bfcb6000, 4096, PROT_READ) = 0
131 10241 mprotect(0x7ff8aa0c6000, 8192, PROT_READ) = 0
132 10241 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=
   RLIM64_INFINITY}) = 0
133 10241 munmap(0x7ff8aa088000, 21863) = 0
134 10241 futex(0x7ff8aa0847bc, FUTEX_WAKE_PRIVATE, 2147483647) = 0
135 10241 getrandom("\xb7\xff\x09\x72\xfd\x52\x5e\xc5", 8, GRND_NONBLOCK) = 8
136 10241 brk(NULL) = 0x55b0d7e6b000
137 10241 brk(0x55b0d7e8c000) = 0x55b0d7e8c000
138 10241 openat(AT_FDCWD, "t.txt", O_WRONLY|O_CREAT|O_APPEND, 0666) = 3
139 10241 lseek(3, 0, SEEK_END) = 393
140 10241 fstat(3, {st_mode=S_IFREG|0644, st_size=393, ...}) = 0
141 10241 openat(AT_FDCWD, "/dev/shm/lab3_shm", O_RDWR|O_NOFOLLOW|O_CLOEXEC) =
   4
142 10241 mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0
   x7ff8aa08d000
143 10241 rt_sigaction(SIGUSR1, {sa_handler=0x55b0bfcb3293, sa_mask=[], sa_flags=SA_RESTORER, sa_restorer=0x7ff8a9c0f330}, NULL, 8) = 0
144 10241 pause(<unfinished ...>
145 10205 <... clock_nanosleep resumed>NULL) = 0
146 10205 write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
   \321\207\320\270\321\201\320\273\320\260: '\320\264\320\265"..., 93) =
   93
147 10205 read(0, "10 5 2\n", 1024) = 7
148 10205 kill(10241, SIGUSR1) = 0
149 10241 <... pause resumed> = ? ERESTARTNOHAND (To be
   restarted if no handler)
150 10205 pause(<unfinished ...>
151 10241 --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=10205, si_uid
   =1000} ---
152 10241 rt_sigreturn({mask=[]}) = -1 EINTR (Interrupted system
   call)
153 10241 write(3, "Child:
   \320\227\320\260\320\277\321\203\321\211\320\265\320\275.
   \320\244\320\260\320\271\320\273 "..., 78) = 78
154 10241 getppid() = 10205
155 10241 kill(10205, SIGUSR2) = 0
156 10205 <... pause resumed> = ? ERESTARTNOHAND (To be
   restarted if no handler)
157 10205 --- SIGUSR2 {si_signo=SIGUSR2, si_code=SI_USER, si_pid=10241, si_uid
   =1000} ---
158 10241 pause(<unfinished ...>
159 10205 rt_sigreturn({mask=[]}) = -1 EINTR (Interrupted system
   call)
160 10205 read(0, "exit\n", 1024) = 5
161 10205 kill(10241, SIGUSR1) = 0
162 10241 <... pause resumed> = ? ERESTARTNOHAND (To be

```

```

        restarted if no handler)
163 10205 kill(10241, SIGUSR1 <unfinished ...>
164 10241 --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=10205, si_uid
      =1000} ---
165 10205 <... kill resumed>                      = 0
166 10241 rt_sigreturn({mask=[]}) <unfinished ...>
167 10205 wait4(10241,  <unfinished ...>
168 10241 <... rt_sigreturn resumed>           = -1 EINTR (Interrupted system
      call)
169 10241 --- SIGUSR1 {si_signo=SIGUSR1, si_code=SI_USER, si_pid=10205, si_uid
      =1000} ---
170 10241 rt_sigreturn({mask=[]})           = -1 EINTR (Interrupted system
      call)
171 10241 write(3, "Child:
      \320\237\320\276\320\273\321\203\321\207\320\265\320\275\320\260
      \320\272\320\276\320\274\320\260"..., 66) = 66
172 10241 getpid()                         = 10241
173 10241 write(3, "Child: \320\240\320\260\320\261\320\276\321\202\320\260
      \320\267\320\260\320\262\320\265\321\200\321\210"..., 51) = 51
174 10241 close(3)                          = 0
175 10241 munmap(0x7ff8aa08d000, 4096)    = 0
176 10241 close(4)                          = 0
177 10241 exit_group(0)                    = ?
178 10241 +++ exited with 0 ===+
179 10205 <... wait4 resumed>NULL, 0, NULL) = 10241
180 10205 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=10241,
      si_uid=1000, si_status=0, si_utime=2 /* 0.02 s */, si_stime=2 /* 0.02 s
      */} ---
181 10205 munmap(0x7fa69d2b5000, 4096)    = 0
182 10205 close(3)                          = 0
183 10205 unlink("/dev/shm/lab3_shm")       = 0
184 10205 exit_group(0)                    = ?
185 10205 +++ exited with 0 ===+

```

Листинг 4: strace вывод программы