Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №4
по курсу «Операционные системы»
III Семестр

Вариант 17

Студент:	Короткевич Л. В.
Группа:	М80-208Б-19
Преподаватель:	Миронов Е.С
Оценка:	
Дата:	

Москва 2020

1. Постановка задачи

Цель работы:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных 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 программа (основной процесс) должен создать для решение 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.

Группа вариантов 5:

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Вариант 17:

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.

2. Метод решения

Используемые системные вызовы для выполнения работы:

int pipe(int filedes[3]);	pipe создает пару файловых описателей, указывающих на запись inode именованного канала, и помещает их в массив, на который указывает filedes. filedes[0] предназначен для чтения, а filedes[1] предназначен для записи, filedes[2] – информации об ошибках.
void exit(int status);	Функция exit() приводит к обычному завершению программы, и величина status & 0377 (least significant byte of status) возвращается процессу-родителю.

pid_t fork(void);	fork создает процесс-потомок, который отличается от родительского только значениями PID (идентификатор процесса) и PPID (идентификатор родительского процесса), а также тем фактом, что счетчики использования ресурсов установлены в 0. Блокировки файлов и сигналы, ожидающие обработки, не наследуются.
int close(int fd);	close закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно. Все блокировки, находящиеся на соответствующем файле, снимаются (независимо от того, был ли использован для установки блокировки именно этот файловый дескриптор).
int open(const char *pathname, int flags, mode_t mode);	Вызов open() используется, чтобы преобразовать путь к файлу в описатель файла. Если системный вызов завершается успешно, возвращенный файловый описатель является наименьшим описателем, который еще не открыт процессом. Новый описатель файла будет оставаться открытым при выполнении функции exec(2). Указатель устанавливается в начале файла.
int dup2(int oldfd, int newfd); **int dup(int oldfd);**	dup и dup2 создают копию файлового дескриптора oldfd.
pid_t waitpid(pid_t pid, int *status, int options);	Функция waitpid приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс, указанный в параметре pid, не завершит выполнение, или пока не появится сигнал, который либо завершает текущий процесс либо требует вызвать функцию-обработчик.
int open(const char *pathname, int flags);	Вызов open() используется, чтобы преобразовать путь к файлу в описатель файла (небольшое неотрицательно целое число, которое используется с вызовами read, write и т.п. при последующем вводе-выводе).
int fstat(int filedes, struct stat *buf);	fstat идентична stat, только возвращается информация об открытом файле, на который указывает filedes (возвращаемый open(2)), а не о file_name.
void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset);	Функция mmap отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start. Последний параметр (адрес) необязателен, и обычно бывает равен 0.
int munmap(void *start, size_t length);	Функция munmap () удаляет сопоставления для страниц в диапазоне [addr, addr + len), округляя аргумент len до следующего кратного размера страницы, возвращаемого sysconf (3C).

Программа запускается с двумя ключами в качестве названия файлов filename1, filename2 для записи первым и вторым ребенком соответственно. За ними следует считывание произвольного количества строк, надледащих обработке детьми. В зависимости от их длины, каждая строка обрабатывается первым или вторым ребенком, затем записывается в filename1/filename2.

3. Тестирование

```
[leo@pc final]$ gcc main.c -o main
[leo@pc final]$ gcc child.c -o child
[leo@pc final]$ cat test01.txt
“The unexamined life is not worth living” – Socrates
https://www.google.com/search?
q=philosophy+phrases&oq=philosophy+phrases&aqs=chrome..69i57j0l7.14207j0j9&sourceid=chr
ome&ie=UTF-8
123
...biba
boba...
“If God did not exist, it would be necessary to invent Him” – Voltaire
[leo@pc final]$ ./main out1 out2 <test01.txt; cat out1; cat out2;
Enter strings to process:
Child 1 exited, returned  0
Child 2 exited, returned  0
123
...bb
bb...
“Th nxmnd lf s nt wrth lvng” – Scrts
https://www.ggl.cm/srch?
q=phlsph+phrss&q=phlsph+phrss&qs=chrm..6957j0l7.14207j0j9&srcd=chrm&=TF-8
“f Gd dd nt xst, t wld b ncssr t nvnt Hm” – Vltr
[leo@pc final]$ cat test02.txt
a
b
AxAxAx322
15102001leonidvitalyevich
!@#$%^&*()_+alabama
[leo@pc final]$ ./main out1 out2 <test02.txt; cat out1; cat out2;
Enter strings to process:
Child 1 exited, returned  0
Child 2 exited, returned  0

b
xxx322
15102001lndvtlvch
!@#$%^&*()_+lbm
[leo@pc final]$ cat test03.txt
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaB
Baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1a2e3u4i5o6e
U lukomoria dub zeleniy
[leo@pc final]$ ./main out1 out2 <test03.txt; cat out1; cat out2;
Enter strings to process:
Child 1 exited, returned  0
Child 2 exited, returned  0
B
B
123456
 lkmr db zln
[leo@pc final]$ cat test04.txt
```

```
privet
kakdela
1oooo01
vladimir putin molodec
raz i dva
[leo@pc final]$ ./main out1 out2 <test04.txt; cat out1; cat out2;
Enter strings to process:
Child 1 exited, returned  0
Child 2 exited, returned  0
prvt
kkdl
101
rz  dv
vldmr ptn mldc
```

Strace:

strace — это утилита, отслеживающая системные вызовы, которые представляют собой механизм трансляции, обеспечивающий интерфейс между процессом и операционной системой (ядром). Эти вызовы могут быть перехвачены и прочитаны, что позволяет лучше понять, какую задачу процесс пытается сделать в заданное время. Перехватывая эти вызовы, мы можем добиться лучшего понимания поведения процессов, особенно если что-то пошло не так. Функциональность операционной системы, позволяющая отслеживать системные вызовы, называется ptrace. Strace вызывает ptrace и читает данные о поведении процесса, возвращая отчет.

Отображение всех вызовов:

```
[leo@pc final]$ strace ./main out1 out2 <test01.txt
execve("./main", ["./main", "out1", "out2"], 0x7ffcad2f8690 /* 69 vars */) = 0
brk(NULL)                               = 0x56225074b000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffc8b7f480) = -1 EINVAL (Invalid argument)
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) = 6
fstat(6, {st_mode=S_IFREG|0644, st_size=221255, ...}) = 0
mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 6, 0) = 0x7f232a63c000
close(6)                                = 0
openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 6
read(6, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\202\2\0\0\0\0\0"..., 832) = 832
pread64(6, "\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(6, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(6, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\207\360\21\247\344\314?\306\nT\320\323\335i\16t"...,
68, 880) = 68
fstat(6, {st_mode=S_IFREG|0755, st_size=2159552, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f232a63a000
pread64(6, "\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, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 6, 0) =
0x7f232a471000
```

```
mmap(0x7f232a497000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x26000) = 0x7f232a497000
mmap(0x7f232a5e4000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x173000) = 0x7f232a5e4000
mmap(0x7f232a630000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x1be000) = 0x7f232a630000
mmap(0x7f232a636000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f232a636000
close(6)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f232a46f000
arch_prctl(ARCH_SET_FS, 0x7f232a63b580) = 0
mprotect(0x7f232a630000, 12288, PROT_READ) = 0
mprotect(0x5622503e9000, 4096, PROT_READ) = 0
mprotect(0x7f232a69f000, 4096, PROT_READ) = 0
munmap(0x7f232a63c000, 221255)          = 0
openat(AT_FDCWD, "./input.txt", O_RDWR|O_CREAT|O_TRUNC, 0777) = 6
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x4), ...}) = 0
brk(NULL)                               = 0x56225074b000
brk(0x56225076c000)                     = 0x56225076c000
write(1, "Enter strings to process: \n", 27Enter strings to process:
) = 27
fstat(0, {st_mode=S_IFREG|0644, st_size=286, ...}) = 0
read(0, "\342\200\234The unexamined life is not wo"..., 4096) = 286
write(6, "\342\200\234The unexamined life is not wo"..., 59) = 59
write(6, "https://www.google.com/search?q="..., 130) = 130
write(6, "123\n", 4)                    = 4
write(6, "...biba\n", 8)                = 8
write(6, "boba...\n", 8)                = 8
write(6, "\342\200\234If God did not exist, it woul"..., 77) = 77
read(0, "", 4096)                       = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f232a63b850) = 4174
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLD, child_tidptr=0x7f232a63b850) = 4175
close(6)                                = 0
wait4(4174, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 4174
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4174, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
write(1, "Child 1 exited, returned  0\n", 28Child 1 exited, returned  0
) = 28
wait4(4175, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 4175
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4175, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
write(1, "Child 2 exited, returned  0\n", 28Child 2 exited, returned  0
) = 28
exit_group(0)                           = ?
+++ exited with 0 +++
```

Отображение определенных вызовов:

Опция -e служит для отображения лишь определенных вызовов.

Например — отобразить только вызовы mmap:

```
[leo@pc final]$ strace -e mmap ./main out1 out2 <test01.txt
mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 6, 0) = 0x7f8bade31000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f8bade2f000
mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 6, 0) =
0x7f8badc66000
mmap(0x7f8badc8c000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x26000) = 0x7f8badc8c000
mmap(0x7f8baddd9000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x173000) = 0x7f8baddd9000
mmap(0x7f8bade25000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x1be000) = 0x7f8bade25000
mmap(0x7f8bade2b000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f8bade2b000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f8badc64000
Enter strings to process:
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4663, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned  0
Child 2 exited, returned  0
+++ exited with 0 +++
```

Отслеживание дочерних процессов:

Отслеживать дерево процессов целиком помогает флаг -f, с которым strace отслеживает системные вызовы в процессах-потомках. К каждой строке вывода при этом добавляется pid процесса, делающего системный вывод:

```
[leo@pc final]$ strace -f -e mmap ./main out1 out2 <test01.txt
mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 6, 0) = 0x7f29ffc01000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f29ffbff000
mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 6, 0) =
0x7f29ffa36000
mmap(0x7f29ffa5c000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x26000) = 0x7f29ffa5c000
mmap(0x7f29ffba9000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x173000) = 0x7f29ffba9000
mmap(0x7f29ffbf5000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x1be000) = 0x7f29ffbf5000
mmap(0x7f29ffbfb000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f29ffbfb000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f29ffa34000
Enter strings to process:
strace: Process 4697 attached
[pid  4697] mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 7, 0) = 0x7f72ad80e000
[pid  4697] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f72ad80c000
[pid  4697] mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 7, 0) =
0x7f72ad643000
```

```
[pid  4697] mmap(0x7f72ad669000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x26000) = 0x7f72ad669000
[pid  4697] mmap(0x7f72ad7b6000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 7, 0x173000) = 0x7f72ad7b6000
[pid  4697] mmap(0x7f72ad802000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x1be000) = 0x7f72ad802000
[pid  4697] mmap(0x7f72ad808000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f72ad808000
[pid  4697] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f72ad641000
[pid  4697] mmap(NULL, 286, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) =
0x7f72ad844000
[pid  4697] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4697, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
strace: Process 4696 attached
[pid  4696] mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 7, 0) = 0x7f19942ed000
[pid  4696] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f19942eb000
[pid  4696] mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 7, 0) =
0x7f1994122000
[pid  4696] mmap(0x7f1994148000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x26000) = 0x7f1994148000
[pid  4696] mmap(0x7f1994295000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 7, 0x173000) = 0x7f1994295000
[pid  4696] mmap(0x7f19942e1000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x1be000) = 0x7f19942e1000
[pid  4696] mmap(0x7f19942e7000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f19942e7000
[pid  4696] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f1994120000
[pid  4696] mmap(NULL, 286, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) =
0x7f1994323000
[pid  4696] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4696, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 1 exited, returned  0
Child 2 exited, returned  0
+++ exited with 0 +++
```

В данном случае будет полезной **фильтрация по группам вызовов:**

```
[leo@pc final]$ strace -f -e trace=%memory ./main out1 out2 <test01.txt
brk(NULL)                               = 0x5581b6fa5000
mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 6, 0) = 0x7f3d683b2000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f3d683b0000
mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 6, 0) =
0x7f3d681e7000
mmap(0x7f3d6820d000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x26000) = 0x7f3d6820d000
```

```
mmap(0x7f3d6835a000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x173000) = 0x7f3d6835a000
mmap(0x7f3d683a6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 6, 0x1be000) = 0x7f3d683a6000
mmap(0x7f3d683ac000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7f3d683ac000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f3d681e5000
mprotect(0x7f3d683a6000, 12288, PROT_READ) = 0
mprotect(0x5581b67b7000, 4096, PROT_READ) = 0
mprotect(0x7f3d68415000, 4096, PROT_READ) = 0
munmap(0x7f3d683b2000, 221255)          = 0
brk(NULL)                               = 0x5581b6fa5000
brk(0x5581b6fc6000)                     = 0x5581b6fc6000
Enter strings to process:
strace: Process 4713 attached
strace: Process 4714 attached
[pid  4713] brk(NULL)                   = 0x55dd79782000
[pid  4713] mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 7, 0) = 0x7fdcf19ba000
[pid  4714] brk(NULL)                   = 0x564dd37db000
[pid  4713] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7fdcf19b8000
[pid  4714] mmap(NULL, 221255, PROT_READ, MAP_PRIVATE, 7, 0) = 0x7f5c3efbc000
[pid  4713] mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 7, 0) =
0x7fdcf17ef000
[pid  4713] mmap(0x7fdcf1815000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x26000) = 0x7fdcf1815000
[pid  4713] mmap(0x7fdcf1962000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 7, 0x173000) = 0x7fdcf1962000
[pid  4713] mmap(0x7fdcf19ae000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x1be000) = 0x7fdcf19ae000
[pid  4713] mmap(0x7fdcf19b4000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fdcf19b4000
[pid  4713] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7fdcf17ed000
[pid  4714] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f5c3efba000
[pid  4713] mprotect(0x7fdcf19ae000, 12288, PROT_READ) = 0
[pid  4713] mprotect(0x55dd77eda000, 4096, PROT_READ <unfinished ...>
[pid  4714] mmap(NULL, 1868448, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 7, 0
<unfinished ...>
[pid  4713] <... mprotect resumed>)     = 0
[pid  4714] <... mmap resumed>)         = 0x7f5c3edf1000
[pid  4713] mprotect(0x7fdcf1a1d000, 4096, PROT_READ <unfinished ...>
[pid  4714] mmap(0x7f5c3ee17000, 1363968, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x26000 <unfinished ...>
[pid  4713] <... mprotect resumed>)     = 0
[pid  4714] <... mmap resumed>)         = 0x7f5c3ee17000
[pid  4713] munmap(0x7fdcf19ba000, 221255 <unfinished ...>
[pid  4714] mmap(0x7f5c3ef64000, 311296, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 7, 0x173000 <unfinished ...>
[pid  4713] <... munmap resumed>)       = 0
```

```
[pid  4714] <... mmap resumed>)         = 0x7f5c3ef64000
[pid  4714] mmap(0x7f5c3efb0000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 7, 0x1be000) = 0x7f5c3efb0000
[pid  4714] mmap(0x7f5c3efb6000, 12960, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5c3efb6000
[pid  4713] mmap(NULL, 286, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) =
0x7fdcf19f0000
[pid  4713] brk(NULL)                   = 0x55dd79782000
[pid  4713] brk(0x55dd797a3000 <unfinished ...>
[pid  4714] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0 <unfinished ...>
[pid  4713] <... brk resumed>)          = 0x55dd797a3000
[pid  4714] <... mmap resumed>)         = 0x7f5c3edef000
[pid  4713] munmap(0x7fdcf19f0000, 286) = 0
[pid  4714] mprotect(0x7f5c3efb0000, 12288, PROT_READ) = 0
[pid  4714] mprotect(0x564dd1a71000, 4096, PROT_READ) = 0
[pid  4713] +++ exited with 0 +++
[pid  4714] mprotect(0x7f5c3f01f000, 4096, PROT_READ <unfinished ...>
[pid  4712] --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4713,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
[pid  4714] <... mprotect resumed>)     = 0
Child 1 exited, returned  0
[pid  4714] munmap(0x7f5c3efbc000, 221255) = 0
[pid  4714] mmap(NULL, 286, PROT_READ|PROT_WRITE, MAP_SHARED, 7, 0) =
0x7f5c3eff2000
[pid  4714] brk(NULL)                   = 0x564dd37db000
[pid  4714] brk(0x564dd37fc000)         = 0x564dd37fc000
[pid  4714] munmap(0x7f5c3eff2000, 286) = 0
[pid  4714] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4714, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
Child 2 exited, returned  0
+++ exited with 0 +++
```

Статистика системных вызовов:

С помощью опции -c — можно получить наглядную статистику выполнения программы:

```
[leo@pc final]$ strace -cf ./main out1 out2 <test01.txt
Enter strings to process:
strace: Process 4740 attached
strace: Process 4741 attached
Child 1 exited, returned  0
Child 2 exited, returned  0
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 42.88    0.000331          30        11           write
 42.10    0.000325         162         2           clone
  7.51    0.000058          29         2           wait4
  4.79    0.000037           7         5           read
  2.72    0.000021           2         9           close
  0.00    0.000000           0        12           fstat
  0.00    0.000000           0        26           mmap
```

```
  0.00    0.000000           0        9           mprotect
  0.00    0.000000           0        5           munmap
  0.00    0.000000           0        9           brk
  0.00    0.000000           0       12            pread64
  0.00    0.000000           0        3         3 access
  0.00    0.000000           0        2           dup2
  0.00    0.000000           0        3           execve
  0.00    0.000000           0        6         3 arch_prctl
  0.00    0.000000           0       11           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000772           6      127         6 total
```

4. Листинг программы

main.c:

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

pid_t create_child(char *filename, int type)
{
    pid_t pid = fork();
    if (pid < 0)
    {
        perror("Fork err");
        exit(1);
    }
    if (pid == 0)
    {
        int file_out = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0777);
        if (file_out < 0)
        {
            perror("File err");
            exit(1);
        }

        int new_out = dup2(file_out, STDOUT_FILENO);
        close(file_out);
        if (new_out < 0)
        {
            perror("Duping child stdout err");
            exit(1);
        }

        if (execlp("./child", "child", type ? "1" : "0", NULL) < 0)
        {
            perror("Child execl err");
            exit(1);
        }
    }
    return pid;
}

char *ufgets(FILE *stream)
{
    unsigned int maxlen = 128, size = 128;
    char *buffer = (char *)malloc(maxlen);

    if (buffer != NULL) /* NULL if malloc() fails */
    {
        int ch = EOF;
        int pos = 0;

        /* Read input one character at a time, resizing the buffer as necessary */
        while ((ch = fgetc(stream)) != '\n' && ch != EOF && !feof(stream))
        {
            buffer[pos++] = ch;
```

```
            if (pos == size) /* Next character to be inserted needs more memory */
            {
                size = pos + maxlen;
                buffer = (char *)realloc(buffer, size);
            }
        }
        buffer[pos] = '\0'; /* Null-terminate the completed string */
    }
    return buffer;
}

int main(int argc, char *argv[])
{
    if (argc < 3) {
        printf("Usage: ./main <filename1> <filename2>\n");
        exit(1);
    }

    // output files of child1, child2
    char *filename1 = argv[1];
    char *filename2 = argv[2];

    // input
    int input = open("./input.txt", O_RDWR | O_CREAT | O_TRUNC, 0777);
    if (input < 0) {
        perror("Couldn't open input file\n");
        exit(1);
    }

    char *msg;
    printf("Enter strings to process: \n");
    while ((msg = ufgets(stdin)) && msg[0] != '\0')
    {
        msg[strlen(msg)] = '\n';
        write(input, msg, strlen(msg));
    }

    pid_t pid1, pid2;
    pid1 = create_child(filename1, 0);
    pid2 = create_child(filename2, 1);

    close(input);

    // processes cleanup
    int statusChild1, statusChild2;
    waitpid(pid1, &statusChild1, 0);
    if (WIFEXITED(statusChild1))
    {
        printf("Child 1 exited, returned  %d\n", WEXITSTATUS(statusChild1));
    }
    else
    {
        fprintf(stderr, "Something is wrong with 1st child process\n");
    }
    waitpid(pid2, &statusChild2, 0);
    if (WIFEXITED(statusChild1))
    {
        printf("Child 2 exited, returned  %d\n", WEXITSTATUS(statusChild2));
    }
    else
    {
        fprintf(stderr, "Something is wrong with 2nd child process\n");
    }

    // memory cleanup
    free(msg);

    return 0;
}
```

child.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <stdint.h>
#include <fcntl.h>
```

```
#include <sys/wait.h>
#include <sys/mman.h>
#include "sys/stat.h"

#define INBUFSIZE 300 // Buffer size

int isVowel(char t)
{
    t = tolower(t);
    if (t == 'a' || t == 'e' || t == 'i' || t == 'o' || t == 'u' || t == 'y')
        return 1;
    return 0;
}

char *ufgetl(char *stream)
{
    unsigned int maxlen = 128, size = 128;
    char *buffer = (char *)malloc(maxlen);

    if (buffer != NULL) /* NULL if malloc() fails */
    {
        int ch = EOF;
        int pos = 0;

        /* Read input one character at a time, resizing the buffer as necessary */
        while ((ch = *(stream + pos)) != '\n' && ch != EOF)
        {
            buffer[pos++] = ch;
            if (pos == size) /* Next character to be inserted needs more memory */
            {
                size = pos + maxlen;
                buffer = (char *)realloc(buffer, size);
            }
        }
        buffer[pos] = '\0'; /* Null-terminate the completed string */
    }
    return buffer;
}

void PrintNoVowels(char *msg, int type) {
    int len = strlen(msg);
    if (len > 10 && type || len <= 10 && !type) {
        for (int i = 0; i < len; ++i)
            if (!isVowel(msg[i]))
                printf("%c", msg[i]);
        printf("\n");
    }
    return;
}

int main(int argc, char *argv[])
{
    int type = atoi(argv[1]);

    int fd = open("./input.txt", O_RDWR, 0777);
    if (fd < 0) {
        perror("Couldn't open input file");
        exit(1);
    }

    struct stat st;
    if (fstat(fd, &st) < 0)
    {
        perror("Couldn't get the input file size");
        exit(1);
    }

    uint8_t *byte_ptr = mmap(NULL,
                             st.st_size,
                             PROT_READ | PROT_WRITE, MAP_SHARED,
                             fd, 0);
    if (byte_ptr < 0) {
        perror("Couldn't mmap file");
        exit(1);
    }

    int len;
    char *msg;
    int pos = 0;
```

```
    while ( (msg = ufgetl(byte_ptr + pos)) && msg[0] != '\0' )
    {
        PrintNoVowels(msg, type);
        pos += strlen(msg);
        free(msg);
        if (byte_ptr[pos] == '\n') pos++;
    }

    if(munmap(byte_ptr, st.st_size) != 0){
        perror("Couldn't munmap ptr");
        exit(1);
    }

    return 0;
}
```

Вывод

По мере выполнения данной лабораторной я освежил в памяти работу с файловыми дескрипторами, создание процессов внутри программы, и, главное, научился отображать файлы в память.

Основные преимущества отображения перед простым чтением и записью в файл:

- Нет необходимости постоянно помнить текущую позицию файла и вовремя её передвигать на позицию, откуда будет производиться чтение или куда будет идти запись.
- Каждый вызов смены/чтения текущей позиции, записи/чтения — это системный вызов, который приводит к потере времени.
- Для работы через чтение/запись всё равно приходится выделять буфера определённого размера, таким образом, в общем виде работа состоит из трёх этапов: чтение в буфер -> модификация данных в буфере -> запись в файл. При отображении же работа состоит только из одного этапа: модификации данных в определённой области памяти.

Пожалуй, наиболее общий случай, когда применяется отображение файлов на память, — загрузка процесса в память. Другой общеупотребимый случай использования отображений — создание разделяемых несколькими процессами фрагментов памяти.

Использование файлов, отображенных на память, — это один из наиболее популярных и безопасных (без возникновения исключительных ситуаций) способов сделать память доступной нескольким процессам. Два или более приложений могут одновременно отобразить один и тот же физический файл на свою память и обратиться к этой памяти.