Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №1 по курсу

«Операционные системы»

Группа: М8О-214Б-23

Студент: Кузин А.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.02.25

Москва, 2024

Постановка задачи

Вариант 10.

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 <endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- **int open(const char *pathname, int flags);** Открывает или создаёт файл по пути pathname с флагами flags.
- **int close(int fd);** Закрывает файловый дескриптор fd.
- **int pipe(int pipefd[2]);** Создаёт неименованный канал для взаимодействия процессов. Возвращает файловые дескрипторы: pipefd[0] для чтения и pipefd[1] для записи.
- **pid_t fork(void);** Создаёт дочерний процесс.
- **int dup2(int oldfd, int newfd);** Создаёт дубликат файлового дескриптора oldfd newfd.
- **int execv(const char *path, char *const argv[]);** Заменяет текущий образ процесса новым, загружая исполняемый файл по пути path с аргументами argv.
- **pid_t waitpid(pid_t pid, int *status, int options);** Приостанавливает выполение текущего процесса до тех пор, пока дочерний процесс pid не совершит выполнение, и возвращает его статус в status.
- **void exit(int status);** Завершает процесс с кодом завершения status.

Программа считывает имя файла из стандартного ввода, открывает его функцией open. Создаёт канал (pipe) с помощью функции pipe. Потом создаётся дочерний процесс с помощью fork. Дочерний процесс перенаправляет стандартный ввод на открытый файл с помощью dup2 и перенаправляет стандартный вывод на конец записи канала, чтобы отправлять данные обратно родительскому процессу. Выполняет исполняемый файл child.out с помощью execv.

Родительский процесс закрывает конец записи канала (pipe), считывает из канала числа, отправленные дочерним процессом, и передаёт их функции print_int для вывода на экран. Он ожидает завершения дочернего процесса с помощью waitpid. Затем проверяет код завершения дочернего процесса: если он завершился с ошибкой, выводит сообщение об этом и завершает работу с кодом ошибки.

Дочерний процесс считывает числа из перенаправленного стандартного ввода (файла, открытого родительским процессом) с помощью функции process_line.

- Если число составное, записывает его в стандартный вывод (который перенаправлен в канал).
- Если число отрицательное, простое или некорректное, завершает работу с кодом ошибки (сюда же относится 0).
- Если в потоке входных данных встречаются только составные числа, программа завершает работу с кодом успеха.

Код программы

main.c

```
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <wait.h>

int print_int(const int num) {
char buf[16];
char res[32];
int n = 0;
int sign = (num < 0);
int x = abs(num);
if (x == 0) {
write(STDOUT_FILENO, "0\n", 2);
return 0;
}
while (x) {
buf[n] = (x % 10) + '0';
x = x / 10;
n++;
}
if (sign) {
res[0] = '-';
}
for (int i = 0; i < n; i++) {
res[i + sign] = buf[n - i - 1];
}
res[n + sign] = '\n';
write(STDOUT_FILENO, res, (n + sign + 1));
return 0;
}

int read_line(char** buf, int* n){
char c;
int i = 0;
while(1) {
if (read(STDIN_FILENO, &c, sizeof(char)) == -1) {
return -1;
}
(*buf)[i] = c;
i++;
if (i >= *n) {
```

```
*buf = realloc(*buf, (*n) * 2 * sizeof(char));
*n = *n * 2;
}
if (c == '\n') {
(*buf)[i - 1] = '\0';
break;
}
}
return 0;
}

int main() {
char* buf = malloc(128 * sizeof(char));
int n = 128;
if (read_line(&buf, &n) == -1) {
char* msg = "failed to read file name\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

int file_fd = open(buf, O_RDONLY);
if (file_fd == -1) {
char* msg = "failed to open file\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

int pipe_fd [2];
if (pipe(pipe_fd) == -1) {
char* msg = "failed to create pipe\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}

__pid_t pid = fork();

if (pid == 0) {
if (dup2(file_fd, STDIN_FILENO) == -1) {
write(STDOUT_FILENO, "dup2 failed\n", 12);
exit(EXIT_FAILURE);
}
close(file_fd);

if (dup2(pipe_fd[1], STDOUT_FILENO) == -1) {
write(STDOUT_FILENO, "dup2 failed\n", 12);
exit(EXIT_FAILURE);
}
close(pipe_fd[1]);
close(pipe_fd[0]);

char *arg[] = {"./child.out", NULL};
if (execv("./child.out", arg) == -1) {
write(STDOUT_FILENO, "execv failed\n", 13);
exit(EXIT_FAILURE);
}
} else {
close(pipe_fd[1]);

int x;
int received = 0;
while (read(pipe_fd[0], &x, sizeof(int)) > 0) {
print_int(x);
received = 1;
}
close(pipe_fd[0]);

int status;
waitpid(pid, &status, 0);
```

```
if (WIFEXITED(status)) {
int exit_code = WEXITSTATUS(status);
if (exit_code != 0) {
char* msg = "ERROR: child exited with an error\n";
write(STDERR_FILENO, msg, strlen(msg));
exit(EXIT_FAILURE);
}
}
}
if (pid == -1) {
char* msg = "fail to fork\n";
write(STDOUT_FILENO, msg, strlen(msg));
exit(-1);
}
return 0;
}
```

child.c

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

#define BUFFER_SIZE 512

int is_prime(int n) {
if (n < 2) {
return 0;
}
for (int i = 2; i * i <= n; i++){
if (n % i == 0) {
return 0;
}
}
return 1;
}

int is_num(char c) {
return (c >= '0') && (c <= '9');
}

int is_space(char c) {
return (c == ' ') || (c == '\t') || (c == '\n');
}

int process_line(int* res) {
char buffer[BUFFER_SIZE];
int index = 0;
char c;
int sign = 1;
*res = 0;

while(read(STDIN_FILENO, &c, sizeof(char)) > 0) {
if (c == '-') {
if (index != 0) {
return -1;
}
sign = -1;
} else if(is_num(c)) {
*res = *res * 10 + (c - '0');
} else if (is_space(c)) {
break;
} else {
return -1;
}
index++;
}
```

```
*res *= sign;
return (index > 0) ? 0 : -1;
}

int main() {
int n;
int has_composite_numbers = 0;

while (1) {
if (n <= 0 || is_prime(n)) {
exit(EXIT_FAILURE);
} else {
has_composite_numbers = 1;
if (write(STDOUT_FILENO, &n, sizeof(int)) != sizeof(int)) {
exit(EXIT_FAILURE);
}
}
if (process_line(&n) < 0) {
if (has_composite_numbers) {
exit(EXIT_SUCCESS);
} else {
exit(EXIT_FAILURE);
}
}
}
return 0;
}
```

Протокол работы программы

Тестирование:

1. Все числа составные

$cat > a.txt
8 12
16
26
87 100 24
4
$ cat > run.txt
a.txt
$ ./main.out < run.txt
8
12
16
26
87
100
24
4

2. Встречается простое число

$ cat > a.txt
8 12
3 5 7

16
4
$ ./main.out < run.txt
8
12
ERROR: child exited with an error

3. Встречается отрицательное число

$ cat > a.txt

8 12 4
-4
16
$ ./main.out < run.txt
8
12
4
ERROR: child exited with an error

4. Встречается 0(не простое и не составное, поэтому считается невалидным)

$ cat > a.txt
4 10
0
15 3
$ ./main.out < run.txt
4
10
ERROR: child exited with an error

Strace:

$ strace -f ./main.out < run.txt
execve("./main.out", ["./main.out"], 0x7ffc919cc588 /* 63 vars */) = 0
brk(NULL) = 0x56987420e000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc90953210) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72eb2f2a1000
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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58791, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 58791, PROT_READ, MAP_PRIVATE, 3, 0) = 0x72eb2f292000
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)
= 0x72eb2f000000

 mprotect(0x72eb2f028000, 2023424, PROT_NONE) = 0

 mmap(0x72eb2f028000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x72eb2f028000

 mmap(0x72eb2f1bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1bd000) = 0x72eb2f1bd000

 mmap(0x72eb2f216000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x72eb2f216000

 mmap(0x72eb2f21c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x72eb2f21c000

 close(3)                                = 0

 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x72eb2f28f000
 arch_prctl(ARCH_SET_FS, 0x72eb2f28f740) = 0
 set_tid_address(0x72eb2f28fa10)         = 8074
 set_robust_list(0x72eb2f28fa20, 24)     = 0
 rseq(0x72eb2f2900e0, 0x20, 0, 0x53053053) = 0
 mprotect(0x72eb2f216000, 16384, PROT_READ) = 0
 mprotect(0x569873a06000, 4096, PROT_READ) = 0
 mprotect(0x72eb2f2db000, 8192, PROT_READ) = 0

 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
 munmap(0x72eb2f292000, 58791)           = 0
 getrandom("\xee\x54\x08\x5f\xc9\xac\x0d\x4b", 8, GRND_NONBLOCK) = 8
 brk(NULL)                               = 0x56987420e000
 brk(0x56987422f000)                     = 0x56987422f000
 read(0, "a", 1)                         = 1
 read(0, ".", 1)                         = 1
 read(0, "t", 1)                         = 1
 read(0, "x", 1)                         = 1
 read(0, "t", 1)                         = 1
 read(0, "\n", 1)                        = 1
 openat(AT_FDCWD, "a.txt", O_RDONLY)     = 3
```

pipe2([4, 5], 0) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 8075 attached

<unfinished ...>

[pid 8075] set_robust_list(0x72eb2f28fa20, 24) = 0

[pid 8075] dup2(3, 0) = 0

[pid 8075] close(3) = 0

[pid 8075] dup2(5, 1 <unfinished ...>

[pid 8074] <... clone resumed>, child_tidptr=0x72eb2f28fa10) = 8075

[pid 8075] <... dup2 resumed>) = 1

[pid 8075] close(5) = 0

[pid 8075] close(4) = 0

[pid 8075] execve("./child.out", ["./child.out"], 0x7ffc909533e8 /* 63 vars */ <unfinished ...>

[pid 8074] close(5) = 0

[pid 8074] read(4, <unfinished ...>

[pid 8075] <... execve resumed>) = 0

[pid 8075] brk(NULL) = 0x5558b4fc7000

[pid 8075] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffce4d98e00) = -1 EINVAL (Invalid argument)

[pid 8075] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7818946a7000

[pid 8075] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 8075] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 8075] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58791, ...}, AT_EMPTY_PATH) = 0

[pid 8075] mmap(NULL, 58791, PROT_READ, MAP_PRIVATE, 3, 0) = 0x781894698000

[pid 8075] close(3) = 0

[pid 8075] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 8075] 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

[pid 8075] 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

[pid 8075] 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

```
[pid 8075] 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

[pid 8075] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...},
AT_EMPTY_PATH) = 0

[pid 8075] 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

[pid 8075] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|
MAP_DENYWRITE, 3, 0) = 0x781894400000

[pid 8075] mprotect(0x781894428000, 2023424, PROT_NONE) = 0

[pid 8075] mmap(0x781894428000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x781894428000

[pid 8075] mmap(0x7818945bd000, 360448, PROT_READ, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7818945bd000

[pid 8075] mmap(0x781894616000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) =
0x781894616000

[pid 8075] mmap(0x78189461c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x78189461c000

[pid 8075] close(3)                     = 0

[pid 8075] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x781894695000
[pid 8075] arch_prctl(ARCH_SET_FS, 0x781894695740) = 0
[pid 8075] set_tid_address(0x781894695a10) = 8075
[pid 8075] set_robust_list(0x781894695a20, 24) = 0
[pid 8075] rseq(0x7818946960e0, 0x20, 0, 0x53053053) = 0
[pid 8075] mprotect(0x781894616000, 16384, PROT_READ) = 0
[pid 8075] mprotect(0x5558b4be9000, 4096, PROT_READ) = 0
[pid 8075] mprotect(0x7818946e1000, 8192, PROT_READ) = 0

[pid 8075] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
[pid 8075] munmap(0x781894698000, 58791) = 0
[pid 8075] read(0, "4", 1)              = 1
[pid 8075] read(0, " ", 1)              = 1
[pid 8075] write(1, "\4\0\0\0", 4 <unfinished ...>
[pid 8074] <... read resumed>"\4\0\0\0", 4) = 4
[pid 8075] <... write resumed>)         = 4
[pid 8074] write(1, "4\n", 2 <unfinished ...>
[pid 8075] read(0, 4
 <unfinished ...>
[pid 8074] <... write resumed>)         = 2
```

```
[pid 8075] <... read resumed>"6", 1)   = 1
[pid 8074] read(4, <unfinished ...>
[pid 8075] read(0, " ", 1)             = 1
[pid 8075] write(1, "\6\0\0\0", 4 <unfinished ...>
[pid 8074] <... read resumed>"\6\0\0\0", 4) = 4
[pid 8075] <... write resumed>)        = 4
[pid 8074] write(1, "6\n", 2 <unfinished ...>
[pid 8075] read(0, 6
 <unfinished ...>
[pid 8074] <... write resumed>)        = 2
[pid 8075] <... read resumed>"8", 1)   = 1
[pid 8074] read(4, <unfinished ...>
[pid 8075] read(0, " ", 1)             = 1
[pid 8075] write(1, "\10\0\0\0", 4 <unfinished ...>
[pid 8074] <... read resumed>"\10\0\0\0", 4) = 4
[pid 8075] <... write resumed>)        = 4
[pid 8074] write(1, "8\n", 2 <unfinished ...>
[pid 8075] read(0, 8
 <unfinished ...>
[pid 8074] <... write resumed>)        = 2
[pid 8075] <... read resumed>"8", 1)   = 1
[pid 8074] read(4, <unfinished ...>
[pid 8075] read(0, "\n", 1)            = 1
[pid 8075] write(1, "\10\0\0\0", 4)    = 4
[pid 8074] <... read resumed>"\10\0\0\0", 4) = 4
[pid 8074] write(1, "8\n", 2 <unfinished ...>
8
[pid 8075] read(0, <unfinished ...>
[pid 8074] <... write resumed>)        = 2
[pid 8074] read(4, <unfinished ...>
[pid 8075] <... read resumed>"", 1)    = 0
[pid 8075] exit_group(0)               = ?
[pid 8074] <... read resumed>"", 4)    = 0
[pid 8075] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=8075,
si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
close(4)                               = 0
wait4(8075, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 8075
exit_group(0)                          = ?
+++ exited with 0 +++
```

Вывод

Была написана программа на Си, использующая родительский и дочерний процессы, а также переход pipe между ними.