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: Миронов Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _______________

Дата: 28.10.24

Москва, 2024

Постановка задачи

Вариант 1.

Пользователь вводит строки произвольной длины, которые принимаются родительским процессом. Родительский процесс отправляет их первому дочернему процессу, если длина строки меньше 10, или второму – если больше. Дочерние процессы принимают строки и записывают их в собственные выходные файлы, удаляя из строк все гласные.

Общий метод и алгоритм решения

Использованные системные вызовы:

- int channel[2];
  pipe(channel); – создает два канала связи.
- const pid_t child = fork(); – создает дочерний процесс.
- pid_t pid = getpid(); – получает номер текущего процесса.
- dup2(STDIN_FILENO, channel[STDIN_FILENO]); – перенаправляет стандартный ввод на дескриптор родительского канала связи.
- int32_t status = execv(path, args); – заменяет код завершения дочернего процесса.
- wait(&child_status); – родительский процесс ждет завершения дочернего процесса.

Решение:

1. Обрабатываю путь переданный через аргументы командной строки.
2. Считываю строку
3. С помощью функций написанных выше связываю родительский процесс с дочерним.
4. В дочернем процессе получаю строку, переданную от родительского процесса и удаляю из неё гласные.
5. Записываю полученную строку в файл.

Код программы

Server.c

```c
#include <stdint.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>

static char CLIENT1_PROGRAM_NAME[] = "client1";
static char CLIENT2_PROGRAM_NAME[] = "client2";

int main(int argc, char **argv) {
    if (argc == 2) {
        char msg[1024];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]);
        write(STDERR_FILENO, msg, len);
        exit(EXIT_SUCCESS);
    }
    char progpath[1024];
    {
        // NOTE: Read full program path, including its name
        ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
        if (len == -1) {
            const char msg[] = "error: failed to read full program path\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        // NOTE: Trim the path to first slash from the end
        while (progpath[len] != '/')
            --len;
        progpath[len] = '\0';
    }

    char buf[4096];
    ssize_t bytes;
    int flag = 0;

    while (bytes = read(STDIN_FILENO, buf, sizeof(buf))) {
        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        } else if (buf[0] == '\n') {
            break;
        }
        if (bytes < 10) {
            flag = 1;
        } else {
            flag = 2;
        }
        buf[bytes - 1] = '\0';

        // NOTE: Open pipe
        int channel[2];
        if (pipe(channel) == -1) {
            const char msg[] = "error: failed to create pipe\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }

        // NOTE: Spawn a new process
        const pid_t child = fork();

        switch (child) {
            case -1: { // NOTE: Kernel fails to create another process
                const char msg[] = "error: failed to spawn new process\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            } break;

            case 0: { // NOTE: We're a child, child doesn't know its pid after fork
                pid_t pid = getpid(); // NOTE: Get child PID

                // NOTE: Connect parent stdin to child stdin
                dup2(STDIN_FILENO, channel[STDIN_FILENO]);
                close(channel[STDOUT_FILENO]);

                {
                    char msg[64];
                    const int32_t length = snprintf(msg, sizeof(msg),
                        "%d: I'm a child\n", pid);
                    write(STDOUT_FILENO, msg, length);
                }

                {
                    char path[1024];
                    if (flag == 1) {
                        snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
CLIENT1_PROGRAM_NAME);
                        char *const args[] = {CLIENT1_PROGRAM_NAME, argv[1], buf, NULL};
                        int32_t status = execv(path, args);
                        if (status == -1) {
                            const char msg[] = "error: failed to exec into new
exectuable image\n";
                            write(STDERR_FILENO, msg, sizeof(msg));
                            exit(EXIT_FAILURE);
                        }
                    } else {
                        snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
CLIENT2_PROGRAM_NAME);
                        char *const args[] = {CLIENT2_PROGRAM_NAME, argv[2], buf, NULL};
                        int32_t status = execv(path, args);
                        if (status == -1) {
                            const char msg[] = "error: failed to exec into new
exectuable image\n";
                            write(STDERR_FILENO, msg, sizeof(msg));
                            exit(EXIT_FAILURE);
                        }
                    }
                }
            } break;

            default: { // NOTE: We're a parent, parent knows PID of child after fork
                pid_t pid = getpid(); // NOTE: Get parent PID

                {
                    char msg[64];
                    const int32_t length = snprintf(msg, sizeof(msg),
                        "%d: I'm a parent, my child has PID %d\n", pid, child);
                    write(STDOUT_FILENO, msg, length);
                }

                // NOTE: `wait` blocks the parent until child exits
                int child_status;
                wait(&child_status);

                if (child_status != EXIT_SUCCESS) {
                    const char msg[] = "error: child exited with error\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(child_status);
                }
            } break;
        }
    }
}
```

Client1.c

```
#include <stdint.h>
#include <stdbool.h>

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

int main(int argc, char **argv) {
    char buf[4096];
    const char *vowels = "aeiouAEIOU";

    pid_t pid = getpid();

    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_APPEND, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int i = 0;
    int ind = 0;
    while (argv[2][i] != '\0') {
```

```c
        if (!strchr(vowels, argv[2][i])) {
            buf[ind] = argv[2][i];
            ind++;
        }
        i++;
    }
    buf[ind] = '\0';
    int32_t len = ind + 1;

    int32_t written = write(file, buf, len);
    if (written != len) {
        const char msg[] = "error: failed to write to file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    close(file);
}
```

Client2.c

```c
#include <stdint.h>
#include <stdbool.h>

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

int main(int argc, char **argv) {
    char buf[4096];
    const char *vowels = "aeiouAEIOU";

    pid_t pid = getpid();

    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_APPEND, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int i = 0;
    int ind = 0;
    while (argv[2][i] != '\0') {
        if (!strchr(vowels, argv[2][i])) {
            buf[ind] = argv[2][i];
            ind++;
        }
        i++;
    }
    buf[ind] = '\0';
    int32_t len = ind + 1;

    int32_t written = write(file, buf, len);
    if (written != len) {
        const char msg[] = "error: failed to write to file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    close(file);
}
```

Протокол работы программы

```
$ ./server file1.txt file2.txt
sometext
520: I'm a parent, my child has PID 521
521: I'm a child
text
520: I'm a parent, my child has PID 522
522: I'm a child
and some more text
520: I'm a parent, my child has PID 523
523: I'm a child
string
520: I'm a parent, my child has PID 524
524: I'm a child

$ cat file1.txt
smtxttxtstrng

$ cat file2.txt
nd sm mr txt

Strace:
$ strace -f ./server
execve("./server", ["./server"], 0x7ffea71fb978 /* 27 vars */) = 0
brk(NULL)                               = 0x55e1a952f000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f77df683000
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=19711, ...}) = 0
mmap(NULL, 19711, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f77df67e000
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"..., 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"..., 784, 64)
= 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f77df46c000
mmap(0x7f77df494000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f77df494000
mmap(0x7f77df61c000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f77df61c000
mmap(0x7f77df66b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7f77df66b000
mmap(0x7f77df671000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f77df671000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f77df469000
arch_prctl(ARCH_SET_FS, 0x7f77df469740) = 0
set_tid_address(0x7f77df469a10)         = 642
set_robust_list(0x7f77df469a20, 24)     = 0
rseq(0x7f77df46a060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f77df66b000, 16384, PROT_READ) = 0
mprotect(0x55e1a948a000, 4096, PROT_READ) = 0
mprotect(0x7f77df6bb000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f77df67e000, 19711)           = 0
readlink("/proc/self/exe", "/mnt/c/Users/begemot/ClionProjec"..., 1023) = 54
read(0, scscasc
"scscasc\n", 4096)              = 8
pipe2([3, 4], 0)                        = 0
```

```
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: 
Process 643 attached
, child_tidptr=0x7f77df469a10) = 643
[pid   643] set_robust_list(0x7f77df469a20, 24 <unfinished ...>
[pid   642] getpid( <unfinished ...>
[pid   643] <... set_robust_list resumed>) = 0
[pid   642] <... getpid resumed>)       = 642
[pid   642] write(1, "642: I'm a parent, my child has "..., 40 <unfinished ...>
642: I'm a parent, my child has PID 643
[pid   643] getpid( <unfinished ...>
[pid   642] <... write resumed>)        = 40
[pid   643] <... getpid resumed>)       = 643
[pid   642] wait4(-1,  <unfinished ...>
[pid   643] dup2(0, 3)                  = 3
[pid   643] close(4)                    = 0
[pid   643] write(1, "643: I'm a child\n", 17643: I'm a child
) = 17
[pid   643] execve("/mnt/c/Users/begemot/ClionProjects/OS-labs/Lab1/client1", 
["client1"], 0x7ffce43f37c8 /* 27 vars */) = 0
[pid   643] brk(NULL)                   = 0x56201d214000
[pid   643] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 
0x7f255ffa4000
[pid   643] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
[pid   643] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4
[pid   643] fstat(4, {st_mode=S_IFREG|0644, st_size=19711, ...}) = 0
[pid   643] mmap(NULL, 19711, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7f255ff9f000
[pid   643] close(4)                    = 0
[pid   643] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4
[pid   643] read(4, 
"\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
[pid   643] 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"..., 784, 64) = 784
[pid   643] fstat(4, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid   643] 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"..., 784, 64) = 784
[pid   643] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) = 
0x7f255fd8d000
[pid   643] mmap(0x7f255fdb5000, 1605632, PROT_READ|PROT_EXEC, 
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x7f255fdb5000
[pid   643] mmap(0x7f255ff3d000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 
4, 0x1b0000) = 0x7f255ff3d000
[pid   643] mmap(0x7f255ff8c000, 24576, PROT_READ|PROT_WRITE, 
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1fe000) = 0x7f255ff8c000
[pid   643] mmap(0x7f255ff92000, 52624, PROT_READ|PROT_WRITE, 
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f255ff92000
[pid   643] close(4)                    = 0
[pid   643] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 
0x7f255fd8a000
[pid   643] arch_prctl(ARCH_SET_FS, 0x7f255fd8a740) = 0
[pid   643] set_tid_address(0x7f255fd8aa10) = 643
[pid   643] set_robust_list(0x7f255fd8aa20, 24) = 0
[pid   643] rseq(0x7f255fd8b060, 0x20, 0, 0x53053053) = 0
[pid   643] mprotect(0x7f255ff8c000, 16384, PROT_READ) = 0
[pid   643] mprotect(0x56201bf47000, 4096, PROT_READ) = 0
[pid   643] mprotect(0x7f255ffdc000, 8192, PROT_READ) = 0
[pid   643] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, 
rlim_max=RLIM64_INFINITY}) = 0
[pid   643] munmap(0x7f255ff9f000, 19711) = 0
[pid   643] getpid()                    = 643
[pid   643] openat(AT_FDCWD, NULL, O_WRONLY|O_CREAT|O_APPEND, 0600) = -1 EFAULT (Bad 
address)
[pid   643] write(2, "error: failed to open requested "..., 38error: failed to open 
requested file
) = 38
[pid   643] exit_group(1)               = ?
[pid   643] +++ exited with 1 +++
<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 1}], 0, NULL) = 643
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=643, si_uid=1000, si_status=1, 
si_utime=0, si_stime=0} ---
write(2, "error: child exited with error\n\0", 32error: child exited with error
) = 32
exit_group(256)                         = ?
+++ exited with 0 +++
```

Вывод

В результате выполнения лабораторной работы удалось познакомиться с системными вызовами (такими как pipe(), fork(), dup2(), execv(), wait()) и реализовать программу записи строк в разные файлы. Проблем при выполнении работы не возникло.