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-210Б-23

Студент: Жданович Е.Т.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.12.24

Москва, 2024

Постановка задачи

Вариант 11.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. Родительский процесс создает два дочерних процесса. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода. 11 вариант) Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «_».

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void) – используется для создания дочернего процесса.
- int pipe(int fd) создает канал для однонаправленной связи между процессами. fd[0] используется для чтения из канала, а fd[1] — для записи в него.
- ssize_t write(int fd, const void buf, size_t count) записывает данные из буфера buf в файл, связанный с файловым дескриптором fd, в количестве байтов, указанном в count.
- ssize_t read(int fd, void buf, size_t count) читает данные из файла или канала, связанного с файловым дескриптором fd, в буфер buf в количестве байтов, указанном в count.
- int execv(const char path, char const argv[]) заменяет текущий процесс новым процессом, запускающим указанную программу.
- int32_t open(const char* file, int oflag, ...); – открывает файл и возвращает файловый дескриптор.
- int close int fd) — закрывает файл.
- int dup2(int oldfd, int newfd) дублирует файловый дескриптор oldfd, заменяя им дескриптор newfd. Перенаправление стандартного ввода дочернего процесса на канал.
- int wait(int status) приостанавливает выполнение родительского процесса до завершения дочернего процесса.

Код программы

Parent.c

```
    #include <stdint.h>

    #include <stdlib.h>

    #include <sys/wait.h>

    #include <unistd.h>

    #include <string.h>

    static char CHILD1_PROGRAM_NAME[] = "./child1";

    static char CHILD2_PROGRAM_NAME[] = "./child2";

    int main(int argc, char **argv) {

     if (argc != 1) { char msg[] = "usage: ./{filename}\n";
write(STDOUT_FILENO, msg, strlen(msg)); exit(EXIT_SUCCESS); }

    char progpath[1024];
{
    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);
    }

    while (progpath[len] != '/')
        --len;

    progpath[len + 1] = '\0';
}



int pipe1[2], pipe2[2], pipe3[2];
if (pipe(pipe1) == -1 || pipe(pipe2) == -1 || pipe(pipe3) == -1) {
    const char msg[] = "error: failed to create pipe\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}



const pid_t child1 = fork();
```

```
switch (child1) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } break;

    case 0: {

        dup2(pipe1[STDIN_FILENO], STDIN_FILENO);
        dup2(pipe2[STDOUT_FILENO], STDOUT_FILENO);

        close(pipe1[STDOUT_FILENO]);
        close(pipe2[STDIN_FILENO]);
        close(pipe3[STDIN_FILENO]);
        close(pipe3[STDOUT_FILENO]);



        {
            char *const args[] = {CHILD1_PROGRAM_NAME, NULL};
            int32_t status = execv(CHILD1_PROGRAM_NAME, 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;
}


const pid_t child2 = fork();
switch (child2) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } break;

    case 0: {

        dup2(pipe2[STDIN_FILENO], STDIN_FILENO);
        dup2(pipe3[STDOUT_FILENO], STDOUT_FILENO);
```

```
        close(pipe1[STDIN_FILENO]);
        close(pipe1[STDOUT_FILENO]);

        close(pipe2[STDOUT_FILENO]);
        close(pipe3[STDIN_FILENO]);



        {
            char *const args[] = {CHILD2_PROGRAM_NAME, NULL};
            int32_t status = execv(CHILD2_PROGRAM_NAME, 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;
}



close(pipe1[0]);
close(pipe2[0]);
close(pipe3[1]);

ssize_t bytes;
char buf[1024];

char msg_of_hint[] = "Enter your string or (Enter / CTRL + D) for stop: \n";
int len_of_msg_of_hint = strlen(msg_of_hint);
write(STDOUT_FILENO, msg_of_hint, len_of_msg_of_hint);
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;
    }
    buf[bytes] = '\0';



    write(pipe1[1], buf, strlen(buf));



    char result[1024];
```

```
    ssize_t bytes_read = read(pipe3[0], result, sizeof(result) - 1);
    if (bytes_read > 0) {
        result[bytes_read] = '\0';
        char msg[] = "Processed result: ";
        write(STDOUT_FILENO, msg, strlen(msg));
        write(STDOUT_FILENO, result, bytes_read - 1);
        write(STDOUT_FILENO, "\n\n", 2);
        write(STDOUT_FILENO, msg_of_hint, len_of_msg_of_hint);
    }
}


close(pipe1[1]);
close(pipe3[0]);
close(pipe2[1]);


wait(NULL);
wait(NULL);

return 0;


    }
```

Child1.c

```
    #include <ctype.h>

    #include <unistd.h>

    #include <string.h>

    int main() {

     char input[1024];

    ssize_t bytes_read;

    while ((bytes_read = read(STDIN_FILENO, input, sizeof(input))) > 0) {


    input[bytes_read]                          =                          '\0';

    for     (int     i     =     0;     i     <     bytes_read;     i++)     {
        input[i]                      =                      toupper(input[i]);
    }
```

```
    write(STDOUT_FILENO,                input,                bytes_read);
}
return                                                                  0;


    }
```

Child2.c

```
    #include <unistd.h>

    #include <string.h>

    int main() {

     char input[1024];

     ssize_t bytes_read;

     while ((bytes_read = read(STDIN_FILENO, input, sizeof(input))) > 0) {


    input[bytes_read] = '\0';


    for (int i = 0; i < bytes_read; i++) {


        if          (input[i]          ==          '          ')          {
            input[i]                          =                          '_';
        }
    }
    write(STDOUT_FILENO,                input,                bytes_read);
}
return                                                                  0;


    }
```

Протокол работы программы

Тестирование:

```
Enter your string or (Enter / CTRL + D) for stop: ijon kj lk
Processed result: IJON_KJ_LK
```

```
Enter your string or (Enter / CTRL + D) for stop: knm kl kjl
Processed result: KNM_KL_KJL


Enter your string or (Enter / CTRL + D) for stop: Hello you
Processed result: HELLO_YOU
```

Strace:

lizka@LizaAlisa:~/ЛАБЫ_ОС/Лаба1$ strace -f ./parent

execve("./parent", ["./parent"], 0x7fffe34dbaf8 /* 29 vars */) = 0 brk(NULL) = 0x55e49d2bc000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f7e3bf2d000

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=19163, ...}) = 0

mmap(NULL, 19163, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f7e3bf28000 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"..., 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) = 0x7f7e3bd16000

mmap(0x7f7e3bd3e000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f7e3bd3e000

mmap(0x7f7e3bec6000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f7e3bec6000

mmap(0x7f7e3bf15000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f7e3bf15000

mmap(0x7f7e3bf1b000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f7e3bf1b000 close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f7e3bd13000

arch_prctl(ARCH_SET_FS, 0x7f7e3bd13740) = 0

set_tid_address(0x7f7e3bd13a10) = 575

set_robust_list(0x7f7e3bd13a20, 24) = 0

rseq(0x7f7e3bd14060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f7e3bf15000, 16384, PROT_READ) = 0

mprotect(0x55e479cfe000, 4096, PROT_READ) = 0

mprotect(0x7f7e3bf65000, 8192, PROT_READ) = 0

*prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,

rlim_max=RLIM64_INFINITY}) = 0 munmap(0x7f7e3bf28000, 19163) = 0 pipe2([3, 4], 0) = 0 pipe2([5, 6], 0) = 0

clone(child_stack=NULL,

flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace:

Process 576 attached , child_tidptr=0x7f7e3bd13a10) = 576

[pid 576] set_robust_list(0x7f7e3bd13a20, 24 <unfinished ...>

[pid 575] close(3 <unfinished ...> [pid 576] <... set_robust_list resumed>) = 0

[pid 575] <... close resumed>) = 0

[pid 576] close(4 <unfinished ...>

[pid 575] close(6 <unfinished ...> [pid 576] <... close resumed>) = 0

[pid 575] <... close resumed>) = 0

[pid 576] close(5 <unfinished ...>

[pid 575] read(0, <unfinished ...>

[pid 576] <... close resumed>) = 0

[pid 576] dup2(3, 0) = 0

[pid 576] dup2(6, 1) = 1

[pid 576] close(3) = 0

[pid 576] close(6) = 0

[pid 576] execve("./out/child1", ["child1"], 0x7ffe4841ed68 /* 29 vars */) = -1 ENOENT (No such file or directory)

[pid 576] exit_group(1) = ?

[pid 576] +++ exited with 1 +++ <... read resumed>0x7ffe4841eb20, 256) = ? ERESTARTSYS (To be restarted if SA_RESTART is set) --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=576, si_uid=1000, si_status=1, si_utime=0, si_stime=0} --- read(0,

Вывод

Во время выполнения лабораторной работы я разработала программу, которая использует несколько процессов для обработки строк, вводимых пользователем. Основная сложность возникла из-за не закрытых каналов (pipes), что приводило к зависанию процессов: дочерние процессы не завершались, поскольку продолжали ждать ввода. Я исправила это, убедившись, что все ненужные дескрипторы закрыты после их использования. В будущем хотелось бы уделить больше времени отладке и тестированию процессов, чтобы избежать подобных проблем. В целом, работа была полезной и помогла мне лучше понять