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-213Б-23

Студент: Османова В.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 06.10.24

Москва, 2024

Постановка задачи

Вариант 6.

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 число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork() - создание дочернего процесса
- int execve(const char *filename, char *const argv[], char *const envp[]) (и другие вариации exec) - замена образа памяти процесса
- pid_t waitpid(pid_t pid, int *status, int options) - Ожидание завершения дочернего процесса
- void exit(int status) - завершения выполнения процесса и возвращение статуса
- int pipe(int pipefd[2]) - создание неименованного канала для передачи данных между процессами
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) - открытие\создание файла
- int close(int fd) - закрыть файл

Программа считывает из стандартного ввода имя файла, открывает его функцией open и создаёт канал с помощью функции pipe. Далее порождается дочерний процесс, у которого с помощью функции dup2 стандартный ввод перенаправлен на открытый ранее файл, а стандартный вывод – на конец для записи созданного канала. Образ дочернего процесса заменяется на child.out с помощью функции execv.

Родительский процесс считывает значения типа int из канала, пока дочерний процесс его не закроет, каждое значение выводится в стандартный вывод с помощью функции print_int, после закрытия канала родительский процесс ждёт завершения дочернего и завершается.

Дочерний процесс считывает данные из стандартного ввода (перенаправленного на файл) с помощью функции read_line_of_int, которая считывает целые числа, разделённые пробелом, пока не достигнет конца строки, после чего записывает сумму чисел в переменную адрес, которой передаётся в качестве аргумента, и возвращает 1. Если строка была пустой возвращается 0. Результат работы read_line_of_int выводится в стандартный вывод (перенаправленный на канал), пока функция не вернёт 0. После этого канал закрывается и процесс завершается.

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 = "fail 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 = "fail to open file\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

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

    __pid_t pid = fork();
    if (pid == -1) {
        char* msg = "fail to fork\n";
        write(STDOUT_FILENO, msg, strlen(msg));
        exit(-1);
    }

    if (pid == 0) {
        if (dup2(file_fd, STDIN_FILENO) == -1) {
            char* msg = "fail to reassign file descriptor\n";
            write(STDOUT_FILENO, msg, strlen(msg));
            exit(-1);
        }
        close(file_fd);
        if (dup2(pipe_fd[1], STDOUT_FILENO) == -1) {
            char* msg = "fail to reassign file descriptor\n";
            write(STDOUT_FILENO, msg, strlen(msg));
            exit(-1);
        }
        close(pipe_fd[1]);
        close(pipe_fd[0]);
        char *arg[] = {"./child.out", NULL};
        if (execv("./child.out", arg) == -1) {
            char* msg = "fail to replace process image\n";
            write(STDOUT_FILENO, msg, strlen(msg));
            exit(-1);
        }
    } else {
        close(pipe_fd[1]);
        int x;
        while (read(pipe_fd[0], &x, sizeof(int)) > 0) {
            print_int(x);
        }
        waitpid(pid, 0 , 0);
    }
    return 0;
}
```

child.c

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

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

int read_line_of_int(int* res) {
    char c = 0;
```

```
    int i = 0, k = 0, sign = 1, r;
    int flag = 1;
    int buf = 0;
    int sum = 0;
    while (c != '\n') {
        while(1) {
            r = read(STDIN_FILENO, &c, sizeof(char));
            if (r < 1) {
                return r;
            }
            if ((c == '\n') || (c == ' ')) {
                break;
            }
            if (flag && (c == '-')) {
                sign = -1;
            } else if (!is_num(c)) {
                return -1;
            } else {
                buf = buf * 10 + c - '0';
                i++;
                if (flag) {
                    k++;
                    flag = 0;
                }
            }
        }
        sum = sum + buf * sign;
        buf = 0;
        flag = 1;
        sign = 1;
    }
    if (k == 0) {
        return 0;
    }
    *res = sum;
    return 1;
}

int main() {
    int res = 0;
    while (read_line_of_int(&res) > 0) {
        write(STDOUT_FILENO, &res, sizeof(int));
    }
    close(STDOUT_FILENO);
    return 0;
}
```

Протокол работы программы

Тестирование:

```
$ cat > a.txt
1 1 1
0 0
1 -2 -3
20 -14
1 2 3 4 5
$ cat > run.txt
a.txt
$ ./main.out < run.txt
3
0
-4
```

Strace:

```
$ strace -f ./main.out < run.txt
execve("./main.out", ["./main.out"], 0x7ffc7c87f738 /* 56 vars */) = 0
brk(NULL)                               = 0x55708d6f4000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcb76e0310) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9233af000
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=61763, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 61763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff92339f000
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<\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
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2072888, ...}, 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, 2117488, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff923000000
mmap(0x7ff923022000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000)
= 0x7ff923022000
mmap(0x7ff92319a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) =
0x7ff92319a000
mmap(0x7ff9231f2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1f1000) =
0x7ff9231f2000
mmap(0x7ff9231f8000, 53104, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7ff9231f8000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff92339c000
arch_prctl(ARCH_SET_FS, 0x7ff92339c740) = 0
set_tid_address(0x7ff92339ca10)         = 16192
set_robust_list(0x7ff92339ca20, 24)     = 0
rseq(0x7ff92339d060, 0x20, 0, 0x53053053) = 0
mprotect(0x7ff9231f2000, 16384, PROT_READ) = 0
mprotect(0x55708d554000, 4096, PROT_READ) = 0
mprotect(0x7ff9233e4000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7ff92339f000, 61763)           = 0
getrandom("\x71\x8f\x21\x9d\xb6\x39\x94\x68", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55708d6f4000
brk(0x55708d715000)                     = 0x55708d715000
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 16193 attached
, child_tidptr=0x7ff92339ca10) = 16193
[pid 16193] set_robust_list(0x7ff92339ca20, 24 <unfinished ...>
[pid 16192] close(5 <unfinished ...>
[pid 16193] <... set_robust_list resumed>) = 0
[pid 16192] <... close resumed>)        = 0
[pid 16192] read(4,  <unfinished ...>
[pid 16193] dup2(3, 0)                  = 0
[pid 16193] close(3)                    = 0
[pid 16193] dup2(5, 1)                  = 1
[pid 16193] close(5)                    = 0
[pid 16193] close(4)                    = 0
[pid 16193] execve("./child.out", ["./child.out"], 0x7ffcb76e0478 /* 56 vars */) = 0
```

```
[pid 16193] brk(NULL)                   = 0x56144cf20000
[pid 16193] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc3c88c580) = -1 EINVAL (Invalid argument)
[pid 16193] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc5ec6d2000
[pid 16193] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
[pid 16193] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid 16193] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=61763, ...}, AT_EMPTY_PATH) = 0
[pid 16193] mmap(NULL, 61763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc5ec6c2000
[pid 16193] close(3)                    = 0
[pid 16193] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[pid 16193] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P<\2\0\0\0\0\0"..., 832) = 832
[pid 16193] 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\0"..., 784, 64) = 784
[pid 16193] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2072888, ...}, AT_EMPTY_PATH) = 0
[pid 16193] 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\0"..., 784, 64) = 784
[pid 16193] mmap(NULL, 2117488, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc5ec400000
[pid 16193] mmap(0x7fc5ec422000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x22000) = 0x7fc5ec422000
[pid 16193] mmap(0x7fc5ec59a000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) =
0x7fc5ec59a000
[pid 16193] mmap(0x7fc5ec5f2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1f1000) = 0x7fc5ec5f2000
[pid 16193] mmap(0x7fc5ec5f8000, 53104, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fc5ec5f8000
[pid 16193] close(3)                    = 0
[pid 16193] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc5ec6bf000
[pid 16193] arch_prctl(ARCH_SET_FS, 0x7fc5ec6bf740) = 0
[pid 16193] set_tid_address(0x7fc5ec6bfa10) = 16193
[pid 16193] set_robust_list(0x7fc5ec6bfa20, 24) = 0
[pid 16193] rseq(0x7fc5ec6c0060, 0x20, 0, 0x53053053) = 0
[pid 16193] mprotect(0x7fc5ec5f2000, 16384, PROT_READ) = 0
[pid 16193] mprotect(0x56144c2ba000, 4096, PROT_READ) = 0
[pid 16193] mprotect(0x7fc5ec707000, 8192, PROT_READ) = 0
[pid 16193] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid 16193] munmap(0x7fc5ec6c2000, 61763) = 0
[pid 16193] read(0, "1", 1)             = 1
[pid 16193] read(0, " ", 1)             = 1
[pid 16193] read(0, "1", 1)             = 1
[pid 16193] read(0, " ", 1)             = 1
[pid 16193] read(0, "1", 1)             = 1
[pid 16193] read(0, "\n", 1)            = 1
[pid 16193] write(1, "\3\0\0\0", 4)     = 4
[pid 16193] read(0, "0", 1)             = 1
[pid 16193] read(0, " ", 1)             = 1
[pid 16193] read(0, "0", 1)             = 1
[pid 16193] read(0, "\n", 1)            = 1
[pid 16193] write(1, "\0\0\0\0", 4)     = 4
[pid 16193] read(0, "1", 1)             = 1
[pid 16193] read(0, " ", 1)             = 1
[pid 16193] read(0, "-", 1)             = 1
[pid 16193] read(0, "2", 1)             = 1
[pid 16193] read(0, " ", 1)             = 1
[pid 16193] read(0, "-", 1)             = 1
[pid 16193] read(0, "3", 1)             = 1
[pid 16193] read(0, "\n", 1)            = 1
[pid 16193] write(1, "\374\377\377\377", 4) = 4
[pid 16193] read(0, "2", 1)             = 1
[pid 16193] read(0, "0", 1)             = 1
[pid 16193] read(0, " ", 1)             = 1
[pid 16193] read(0, "-", 1)             = 1
[pid 16193] read(0, "1", 1)             = 1
[pid 16193] read(0, "4", 1)             = 1
[pid 16193] read(0, "\n", 1)            = 1
[pid 16192] <... read resumed>"\3\0\0\0", 4) = 4
[pid 16193] write(1, "\6\0\0\0", 4)     = 4
```

```
[pid 16193] read(0, "1", 1)          = 1
[pid 16193] read(0, " ", 1)          = 1
[pid 16193] read(0, "2", 1)          = 1
[pid 16193] read(0, " ", 1)          = 1
[pid 16193] read(0, "3", 1)          = 1
[pid 16193] read(0, " ", 1)          = 1
[pid 16193] read(0, "4", 1)          = 1
[pid 16193] read(0, " ", 1)          = 1
[pid 16193] read(0, "5", 1)          = 1
[pid 16193] read(0, "\n", 1)         = 1
[pid 16193] write(1, "\17\0\0\0", 4)   = 4
[pid 16193] read(0, "", 1)           = 0
[pid 16193] close(1)                 = 0
[pid 16193] exit_group(0)            = ?
[pid 16193] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=16193, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
write(1, "3\n", 23
)                 = 2
read(4, "\0\0\0\0", 4)               = 4
write(1, "0\n", 20
)                 = 2
read(4, "\374\377\377\377", 4)       = 4
write(1, "-4\n", 3-4
)                 = 3
read(4, "\6\0\0\0", 4)               = 4
write(1, "6\n", 26
)                 = 2
read(4, "\17\0\0\0", 4)              = 4
write(1, "15\n", 315
)                 = 3
read(4, "", 4)                       = 0
wait4(16193, NULL, 0, NULL)          = 16193
exit_group(0)                        = ?
+++ exited with 0 +++
```

Вывод

Работа над этой программой потребовала глубокого понимания взаимодействия между родительским и дочерним процессами в C. Необходимо было тщательно спроектировать механизм перенаправления ввода-вывода с помощью дескрипторов файлов и трубки, а также реализовать эффективные функции для чтения и обработки целых чисел. Особое внимание пришлось уделить обработке возможных ошибок на каждом этапе выполнения программы, чтобы обеспечить надежную и отказоустойчивую работу. В результате была создана программа, демонстрирующая хорошее понимание концепций межпроцессного взаимодействия и обработки данных в системном программировании на языке C.