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-211Б-23

Студент: Тропн М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.10.24

Москва, 2024

Постановка задачи

Вариант 7.

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

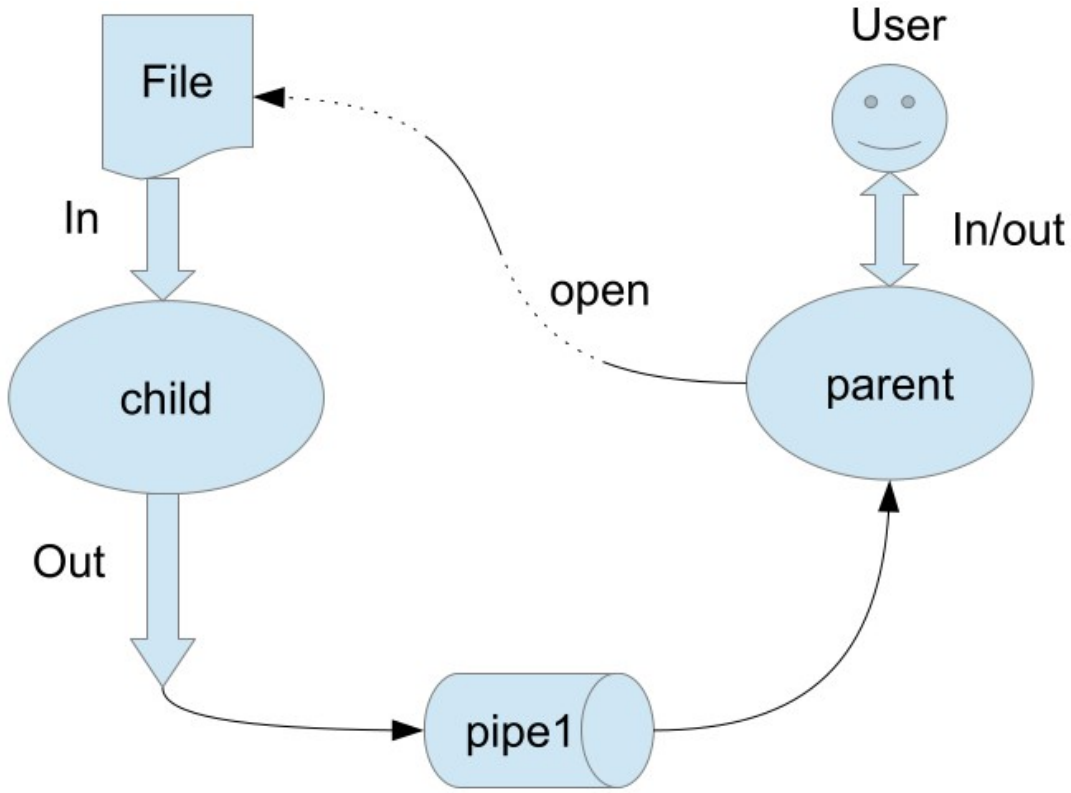
Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды 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.

7 вариант) В файле записаны команды вида: «число число число<endline>». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- int pipe(int *fd); – 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
- void exit(int status) - завершения выполнения процесса и возвращение статуса
- int execv(const char *filename, char *const argv[]) - замена образа памяти процесса
- pid_t getpid(void) — получение ID процесса
- ssize_t read(int __fd, void* __buf, size_t __nbytes) — чтение из fd в буфер
- ssize_t write(int __fd, const void* __buf, size_t __n) — запись байтов в буфер



Главная программа создает дочерный процесс. Он открывает файл на чтение, и переопределяет файловые дескрипторы (STDIN_FILENO становится открытым файлом). Выход (STDOUT_FILENO же становится левым кончиком pipe[STDOUT_FILENO]). Далее запускаем дочерный процесс с аргументами. Клиент же читает из STDIN_FILENO строку в буфер, выполняет необходимо действие (сумма чисел) и пишет результат в STDOUT_FILENO.

Код программы

server.c

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


#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>


static char CLIENT_PROGRAM_NAME[] = "client";


int main(int argc, char** argv) {
  if (argc == 1) {
    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);
```

```
}

// хде я?
char progpath[1024];
{
  // собственно узнаем где
  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);
  }

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

  progpath[len] = '\0';
}

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

// создаем новый процесс
const pid_t child = fork();

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

  case 0: {  // Я - ребенок и не знаю свой PID
             // собственно узнаю
    // pid_t pid = getpid();

    // подключаю родительский ввод к дочернему
    // то есть мой ввод - ввод родителя
    // dup2(STDIN_FILENO, pipe1[STDIN_FILENO]);
    // close(pipe1[STDOUT_FILENO]);
```

```
int file = open(argv[1], O_RDONLY);
if (file == -1) {
  perror("Ошибка при открытии файла");
  exit(EXIT_FAILURE);
}


// stdin_no теперь файл | файл теперь поток входа
if (-1 == dup2(file, STDIN_FILENO)) {
  const char msg[] = "error: to dup file as STDIN_FILENO\n";
  write(STDERR_FILENO, msg, sizeof(msg));
  exit(EXIT_FAILURE);
}
close(file);


// stdout_no теперь правый кончик pipe
if (-1 == dup2(pipe1[STDOUT_FILENO], STDOUT_FILENO)) {
  const char msg[] =
      "error: to dup pipe1[STDOUT_FILENO] as STDOUT_FILENO\n";
  write(STDERR_FILENO, msg, sizeof(msg));
  exit(EXIT_FAILURE);
}
close(pipe1[STDIN_FILENO]);
// close(pipe1[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];
  snprintf(path, sizeof(path) - 1, "%s/%s", progpath,
           CLIENT_PROGRAM_NAME);


  // аргументами для клиента являтся -- сам клиент (его запускатор)б
  // полезные аргументы (файл)б и терминатор, ведь exec требует список с
  // терминирующем нулем
  char* const args[] = {CLIENT_PROGRAM_NAME, argv[1], 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: {  // Я родитель и знаю PID дочерный
  // pid_t pid = getpid();  // Получаем родительский 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);
  // }

  // dup2(pipe1[STDIN_FILENO], STDOUT_FILENO);
  close(pipe1[STDOUT_FILENO]);

  // NOTE: `wait` blocks the parent until child exits
  // блокируем родителя до конца выполнения дочерних процессов
  int child_status;
  wait(&child_status);

  {
    char buffer[4096];
    ssize_t count;

    while ((count = read(pipe1[STDIN_FILENO], buffer, sizeof(buffer)))) {
      if (count < 0) {
        const char msg[] = "error: ferror reading from pipe\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
      } else if (buffer[0] == '\n') {
        break;
      }
    }

    fprintf(stdout, "%s\n", buffer);
  }

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

```
      }
      close(pipe1[STDIN_FILENO]);
    } break;
  }
}
```

client.c

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

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

float summ(const char* src) {
  float sum = 0.0;
  char* endptr;

  while (*src) {
    while (*src && !isdigit(*src) && *src != '.' && *src != '-' &&
           *src != '+') {
      ++src;
    }

    float value = strtod(src, &endptr);
    if (value == HUGE_VAL || value == -HUGE_VAL) {
      return 0.0;
    }

    if (endptr != src) {
      sum += value;
      src = endptr;
    } else {
      ++src;
    }
  }

  return sum;
}

int main(int argc, char** argv) {
  char buf[4096];
```

```
  ssize_t bytes;

  // pid_t pid = getpid();

  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') {
      // проверка на дурака + конец ввода
      break;
    }

    {
      // заменяем лишним пробел окончанием строки и получаем готовую строчечку
      buf[bytes - 1] = '\0';
    }

    {
      char out_buf[1024];
      out_buf[0] = '\0';
      float_t sum = summ(buf);
      sprintf(out_buf, "%f", sum);
      int32_t written = write(STDOUT_FILENO, out_buf, strlen(out_buf));
      if (written == -1) {
        const char msg[] = "error: failed to write to out buffer\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
      }
    }

    const char term = '\0';
    write(STDOUT_FILENO, &term, sizeof(term));
  }
}
```

Протокол работы программы

Тестирование:

```
lemito@lemito:~/Desktop/OSi/lab1$ cat data.base
52.25 48.50 7878.0
lemito@lemito:~/Desktop/OSi/lab1$ ./server data.base
7978.750000
lemito@lemito:~/Desktop/OSi/lab1$ cat data.base
0.0 a b c
lemito@lemito:~/Desktop/OSi/lab1$ ./server data.base
0.000000
```

Strace:

```
execve("./server", ["./server", "data.base"], 0x7fff0b000a50 /* 70 vars */) = 0
brk(NULL)                               = 0x6287b4202000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x75be2c7a1000
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=98075, ...}) = 0
mmap(NULL, 98075, PROT_READ, MAP_PRIVATE, 3, 0) = 0x75be2c789000
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) =
 0x75be2c400000
mmap(0x75be2c428000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x75be2c428000
mmap(0x75be2c5b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x75be2c5b0000
mmap(0x75be2c5ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1fe000) = 0x75be2c5ff000
mmap(0x75be2c605000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x75be2c605000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
 0x75be2c786000
arch_prctl(ARCH_SET_FS, 0x75be2c786740) = 0
set_tid_address(0x75be2c786a10)         = 37577
set_robust_list(0x75be2c786a20, 24)     = 0
rseq(0x75be2c787060, 0x20, 0, 0x53053053) = 0
mprotect(0x75be2c5ff000, 16384, PROT_READ) = 0
mprotect(0x6287b3f1e000, 4096, PROT_READ) = 0
mprotect(0x75be2c7d9000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
 rlim_max=RLIM64_INFINITY}) = 0
munmap(0x75be2c789000, 98075)           = 0
readlink("/proc/self/exe", "/home/lemito/Desktop/OSi/lab1/se"..., 1023) = 36
pipe2([3, 4], 0)                        = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
SIGCHLDstrace: Process 37578 attached
, child_tidptr=0x75be2c786a10) = 37578
[pid 37577] close(4 <unfinished ...>
```

```
[pid 37578] set_robust_list(0x75be2c786a20, 24 <unfinished ...>
[pid 37577] <... close resumed>)        = 0
[pid 37578] <... set_robust_list resumed>) = 0
[pid 37577] wait4(-1,  <unfinished ...>
[pid 37578] openat(AT_FDCWD, "data.base", O_RDONLY) = 5
[pid 37578] dup2(5, 0)                  = 0
[pid 37578] close(5)                    = 0
[pid 37578] dup2(4, 1)                  = 1
[pid 37578] close(3)                    = 0
[pid 37578] execve("/home/lemito/Desktop/OSi/lab1/client", ["client",
 "data.base"], 0x7fff7dcb6880 /* 70 vars */) = 0
[pid 37578] brk(NULL)                   = 0x5c294a4f4000
[pid 37578] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x71f803832000
[pid 37578] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
 directory)
[pid 37578] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid 37578] fstat(3, {st_mode=S_IFREG|0644, st_size=98075, ...}) = 0
[pid 37578] mmap(NULL, 98075, PROT_READ, MAP_PRIVATE, 3, 0) = 0x71f80381a000
[pid 37578] close(3)                    = 0
[pid 37578] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|
O_CLOEXEC) = 3
[pid 37578] 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
[pid 37578] 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 37578] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
[pid 37578] 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 37578] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
 0x71f803600000
[pid 37578] mmap(0x71f803628000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x71f803628000
[pid 37578] mmap(0x71f8037b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x1b0000) = 0x71f8037b0000
[pid 37578] mmap(0x71f8037ff000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x71f8037ff000
[pid 37578] mmap(0x71f803805000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x71f803805000
[pid 37578] close(3)                    = 0
[pid 37578] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
 -1, 0) = 0x71f803817000
[pid 37578] arch_prctl(ARCH_SET_FS, 0x71f803817740) = 0
[pid 37578] set_tid_address(0x71f803817a10) = 37578
[pid 37578] set_robust_list(0x71f803817a20, 24) = 0
[pid 37578] rseq(0x71f803818060, 0x20, 0, 0x53053053) = 0
[pid 37578] mprotect(0x71f8037ff000, 16384, PROT_READ) = 0
[pid 37578] mprotect(0x5c2949cc7000, 4096, PROT_READ) = 0
[pid 37578] mprotect(0x71f80386a000, 8192, PROT_READ) = 0
```

```
[pid 37578] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
 rlim_max=RLIM64_INFINITY}) = 0
[pid 37578] munmap(0x71f80381a000, 98075) = 0
[pid 37578] read(0, "1.25 2.25 3.25 4.25\n", 4096) = 20
[pid 37578] write(1, "11.000000", 9)    = 9
[pid 37578] write(1, "\0", 1)           = 1
[pid 37578] read(0, "", 4096)           = 0
[pid 37578] exit_group(0)               = ?
[pid 37578] +++ exited with 0 +++
<... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 37578
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=37578, si_uid=1000,
 si_status=0, si_utime=0, si_stime=0} ---
read(3, "11.000000\0", 4096)            = 10
read(3, "", 4096)                       = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
getrandom("\x5d\x55\x1a\x3f\x18\x18\xbc\x14", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x6287b4202000
brk(0x6287b4223000)                     = 0x6287b4223000
write(1, "11.000000\n", 1011.000000
)             = 10
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе лабораторной работы я приобрел практические навыки в управлении процессами ОС и обеспечении обмена данных между процессами посредством каналов. Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. Проблем в ходе выполнения не возникло.