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: Абдыкалыков Н. А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 13.02.25

Москва, 2024

**Постановка задачи**

**Вариант 1:**

Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

**Общий метод и алгоритм решения**

Общий метод и алгоритм решения

В данной задаче используется межпроцессное взаимодействие через каналы, чтобы передать числа от родительского процесса в дочерний и получить результат.

Использованные системные вызовы:

1. pipe(int pipefd[2]) — создает неименованный канал для межпроцессного взаимодействия, возвращает два файловых дескриптора для чтения и записи.
2. fork() — создает дочерний процесс, который является копией родительского, возвращает PID дочернего процесса в родительском и 0 в дочернем процессе.
3. \*\*execlp(const char *file, const char arg, ...)* — замещает текущий процесс программой, указанной в аргументе file.
4. dup2(int oldfd, int newfd) — дублирует файловый дескриптор oldfd в newfd, перенаправляя потоки.
5. \**read(int fd, void buf, size\_t count)* — считывает до count байт данных из файлового дескриптора fd в буфер buf.
6. \**write(int fd, const void buf, size\_t count)* — записывает до count байт данных из буфера buf в файловый дескриптор fd.
7. \**wait(int status)* — заставляет родительский процесс ожидать завершения дочернего, возвращая его статус завершения.
8. \**open(const char pathname, int flags, mode\_t mode)* — открывает файл по пути pathname с указанными флагами и правами доступа.
9. close(int fd) — закрывает файловый дескриптор fd, освобождая его ресурсы.
10. \*\*snprintf(char *str, size\_t size, const char format, ...)* — записывает отформатированную строку в буфер str размером size.

**Реализация: идея и подход**

Программа состоит из двух процессов: родительского и дочернего. Родительский процесс запрашивает у пользователя ввод чисел, передает их через канал в дочерний процесс, который вычисляет их сумму и записывает результат в файл.

Родительский процесс создает канал и передает данные (числа) в дочерний. Дочерний процесс получает данные, вычисляет их сумму и записывает результат в файл. После завершения дочерний процесс передает результат обратно родительскому процессу через второй канал. Межпроцессное взаимодействие осуществляется через каналы, а для выполнения задач используется системный вызов fork(), а также read(), write(), и другие функции для работы с файлами и процессами.

**Код программы**

**Файл parent.c**

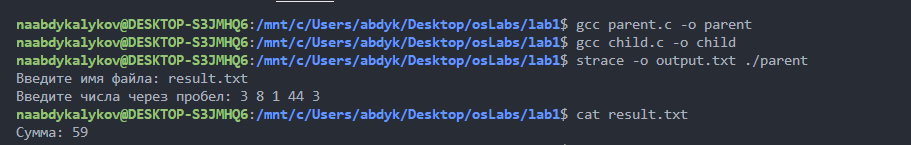
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <fcntl.h>  #include <sys/wait.h>  #define BUFFER\_SIZE 2048  int main() {      int pipe1[2];      pid\_t pid;      char buffer[BUFFER\_SIZE];      char filename[100];      ssize\_t bytes\_read, bytes\_written;      // Создание pipe1      if (pipe(pipe1) == -1) {          perror("pipe");          exit(EXIT\_FAILURE);      }      const char \*msg = "Введите имя файла: ";      bytes\_written = write(STDOUT\_FILENO, msg, strlen(msg));      if (bytes\_written == -1) {          perror("write");          exit(EXIT\_FAILURE);      }      bytes\_read = read(STDIN\_FILENO, filename, sizeof(filename) - 1);      if (bytes\_read == -1) {          perror("read");          exit(EXIT\_FAILURE);      }      // Удаление символа новой строки, если он был введен      if (bytes\_read > 0 && filename[bytes\_read - 1] == '\n') {          filename[bytes\_read - 1] = '\0';      }      // Создание дочернего процесса      pid = fork();      if (pid == -1) {          perror("fork");          exit(EXIT\_FAILURE);      }      if (pid == 0) {          close(pipe1[1]);          dup2(pipe1[0], STDIN\_FILENO);          close(pipe1[0]);          // Запуск дочерней программы          execlp("./child", "child", filename, NULL);          perror("execlp");          exit(EXIT\_FAILURE);      } else {          // Родительский процесс          close(pipe1[0]);  // Закрываем чтение из pipe для родительского процесса          const char \*msg2 = "Введите числа через пробел: ";          write(STDOUT\_FILENO, msg2, strlen(msg2));          bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer));          if (bytes\_read == -1) {              perror("read");              exit(EXIT\_FAILURE);          }          if (write(pipe1[1], buffer, bytes\_read) == -1) {              perror("write to pipe");              exit(EXIT\_FAILURE);          }          close(pipe1[1]);          wait(NULL);      }      return 0;  } |

**Файл child.c**

|  |
| --- |
| #include <fcntl.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <stdio.h>  #include <ctype.h>  #define BUFFER\_SIZE 2048  void convertStringToIntArray(const char \**str*, int *intArray*[], int \**size*) {      int i = 0, num = 0;      \*size = 0;      while (str[i] != '\n' && str[i] != '\0') {          while (isspace(str[i])) {              i++;          }          if (isdigit(str[i])) {              num = 0;              while (isdigit(str[i])) {                  num = num \* 10 + (str[i] - '0');                  i++;              }              intArray[\*size] = num;              (\*size)++;          }      }  }  int main(int *argc*, char \**argv*[]) {      if (argc != 2) {          const char msg[] = "Usage: <program> <filename>\n";          write(STDERR\_FILENO, msg, sizeof(msg) - 1);          exit(EXIT\_FAILURE);      }      // Открываем файл для записи результата      char \*filename = argv[1];      int file = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);      if (file == -1) {          perror("open");          exit(EXIT\_FAILURE);      }      char buffer[BUFFER\_SIZE];      ssize\_t bytes\_read;      // Чтение данных из родительского процесса через pipe      bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer) - 1);      if (bytes\_read <= 0) {          const char msg[] = "error: failed to read from pipe\n";          write(STDERR\_FILENO, msg, sizeof(msg) - 1);          close(file);          exit(EXIT\_FAILURE);      }      buffer[bytes\_read] = '\0';      int array[BUFFER\_SIZE];      int size = 0;      int sum = 0;      convertStringToIntArray(buffer, array, &size);      for (int i = 0; i < size; ++i) {          sum += array[i];      }      // Запись      char output[128];      int len = snprintf(output, sizeof(output), "Сумма: %d\n", sum);      if (write(file, output, len) == -1) {          perror("write to file");          close(file);          exit(EXIT\_FAILURE);      }      close(file);      return 0;  } |

**Протокол работы программы**

**Работа программы и её вывод:**



**Strace:**

execve("./parent", ["./parent"], 0x7ffdc1c8b330 /\* 28 vars \*/) = 0

brk(NULL) = 0x55fc9c1ab000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb8cbe70000

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=19787, ...}) = 0

mmap(NULL, 19787, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fb8cbe6b000

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) = 0x7fb8cbc59000

mmap(0x7fb8cbc81000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fb8cbc81000

mmap(0x7fb8cbe09000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7fb8cbe09000

mmap(0x7fb8cbe58000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7fb8cbe58000

mmap(0x7fb8cbe5e000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fb8cbe5e000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb8cbc56000

arch\_prctl(ARCH\_SET\_FS, 0x7fb8cbc56740) = 0

set\_tid\_address(0x7fb8cbc56a10) = 1816

set\_robust\_list(0x7fb8cbc56a20, 24) = 0

rseq(0x7fb8cbc57060, 0x20, 0, 0x53053053) = 0

mprotect(0x7fb8cbe58000, 16384, PROT\_READ) = 0

mprotect(0x55fc63c2a000, 4096, PROT\_READ) = 0

mprotect(0x7fb8cbea8000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7fb8cbe6b000, 19787) = 0

**### Создание канала (pipe) ###**

**pipe2([3, 4], 0) = 0**

**close(3) = 0**

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\207\320\270\321\201\320\273\320\260 \321\207\320\265\321\200"..., 51) = 51

read(0, "3 8 1 44 3\n", 2048) = 11

write(4, "3 8 1 44 3\n", 11) = 11

close(4) = 0

**### Создание нового процесса ###**

**wait4(-1, NULL, 0, NULL) = 1833**

**--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=1833, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---**

**exit\_group(0) = ?**

**Вывод**

В ходе выполнения лабораторной работы была разработана программа, состоящая из двух процессов: родительского и дочернего. Родительский процесс отвечает за взаимодействие с пользователем, получая от него имя файла и числа, которые необходимо обработать. Дочерний процесс выполняет вычисления, а именно, подсчитывает сумму переданных чисел и записывает результат в указанный файл.