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: 24.10.24

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**

**Использованные системные вызовы:**

**pipe(int pipefd[2]);** — создает неименованный канал для межпроцессного взаимодействия, возвращает два файловых дескриптора для чтения и записи.

**fork();** — создает дочерний процесс, который является копией родительского, возвращает PID дочернего процесса в родительском и 0 в дочернем процессе.

**execlp(const char \*file, const char \*arg, ...);** — замещает текущий процесс программой, указанной в аргументе file.

**dup2(int oldfd, int newfd);** — дублирует файловый дескриптор oldfd в newfd, перенаправляя потоки.

**read(int fd, void \*buf, size\_t count);** — считывает до count байт данных из файлового дескриптора fd в буфер buf.

**write(int fd, const void \*buf, size\_t count);** — записывает до count байт данных из буфера buf в файловый дескриптор fd.

**wait(int \*status);** — заставляет родительский процесс ожидать завершения дочернего, возвращая его статус завершения.

**open(const char \*pathname, int flags, mode\_t mode);** — открывает файл по пути pathname с указанными флагами и правами доступа.

**close(int fd);** — закрывает файловый дескриптор fd, освобождая его ресурсы.

**snprintf(char \*str, size\_t size, const char \*format, ...);** — записывает отформатированную строку в буфер str размером size.

**Реализация: идея и подход**

Идея реализации программы заключается в создании двух процессов, где родительский процесс отвечает за ввод данных и их передачу, а дочерний процесс выполняет обработку и вывод результатов. Такой подход позволяет продемонстрировать основные концепции работы с процессами и межпроцессным взаимодействием в Unix-подобных операционных системах.

**Код программы**

**Файл parent.c**

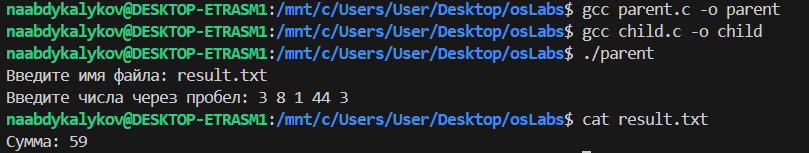
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <string.h>  #include <fcntl.h>  #include <sys/wait.h>  #define BUFFER\_SIZE 2048  int main() {      int pipe1[2];      pid\_t pid;      char buffer[BUFFER\_SIZE];      char filename[100];      ssize\_t bytes\_read, bytes\_written;      // Создание pipe1      if (pipe(pipe1) == -1) {          perror("pipe");          exit(EXIT\_FAILURE);      }      // Сообщение пользователю для ввода имени файла      const char \*msg = "Введите имя файла: ";      bytes\_written = write(STDOUT\_FILENO, msg, strlen(msg));      if (bytes\_written == -1) {          perror("write");          exit(EXIT\_FAILURE);      }      // Чтение имени файла с консоли      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) {          // Дочерний процесс          close(pipe1[1]);  // Закрываем запись в pipe для дочернего процесса          // Перенаправление стандартного ввода на pipe          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          // Сообщение для ввода чисел          const char \*msg2 = "Введите три числа через пробел: ";          write(STDOUT\_FILENO, msg2, strlen(msg2));          // Чтение чисел с консоли          bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer));          if (bytes\_read == -1) {              perror("read");              exit(EXIT\_FAILURE);          }          // Отправляем данные в pipe          write(pipe1[1], buffer, bytes\_read);          close(pipe1[1]);  // Закрываем запись после отправки данных          // Ожидание завершения дочернего процесса          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') {          // Пропускаем пробелы          while (isspace(str[i])) {              i++;          }          // Считываем число          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';  // Завершаем строку      int array[BUFFER\_SIZE];      int size = 0;      int sum = 1;      // Конвертируем строку в массив чисел      convertStringToIntArray(buffer, array, &size);      // Вычисляем сумму      for (int i = 0; i < size; ++i) {          sum \*= array[i];      }      // Записываем результат в файл      char output[128];      int len = snprintf(output, sizeof(output), "Сумма: %d\n", sum);      write(file, output, len);      close(file);      return 0;  } |

**Протокол работы программы**

**Работа программы и её вывод:**



**Strace:**

execve("./parent", ["./parent"], 0x7ffff1de61e0 /\* 15 vars \*/) = 0

brk(NULL) = 0x7ffff2f0a000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd8f0310000

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=16626, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 16626, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fd8f035a000

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\20t\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|0755, st\_size=1922136, ...}, 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, 1970000, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fd8f0120000

mmap(0x7fd8f0146000, 1396736, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x26000) = 0x7fd8f0146000

mmap(0x7fd8f029b000, 339968, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x17b000) = 0x7fd8f029b000

mmap(0x7fd8f02ee000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1ce000) = 0x7fd8f02ee000

mmap(0x7fd8f02f4000, 53072, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fd8f02f4000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd8f0110000

arch\_prctl(ARCH\_SET\_FS, 0x7fd8f0110740) = 0

set\_tid\_address(0x7fd8f0110a10) = 870

set\_robust\_list(0x7fd8f0110a20, 24) = 0

rseq(0x7fd8f0111060, 0x20, 0, 0x53053053) = -1 ENOSYS (Function not implemented)

mprotect(0x7fd8f02ee000, 16384, PROT\_READ) = 0

mprotect(0x7fd8f0362000, 4096, PROT\_READ) = 0

mprotect(0x7fd8f0350000, 8192, PROT\_READ) = 0

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

munmap(0x7fd8f035a000, 16626) = 0

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

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 34) = 34

read(0, "result.txt\n", 99) = 11

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fd8f0110a10) = 871

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

wait4(-1, NULL, 0, NULL) = 871

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=871, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

exit\_group(0) = ?

+++ exited with 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.