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: Амелина А.Е.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 11.12.24

Москва, 2024

Постановка задачи

Вариант 7.

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 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); – создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- int dup2(int __fd, int __fd2); – копирует FD в FD2, закрыв FD2 если это требуется.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
- pid_t waitpid(pid_t pid, int *status, int options) - ожидание завершения дочернего процесса.
- int execl(const char *path, const char *arg, ..., nullptr); - заменяет текущий образ процесса на новый образ.

Для выполнения данной лабораторной работы я изучила указанные выше системные вызовы, а также пример выполнения подобного задания.

Программа parent.cpp запрашивает у пользователя имя файла с помощью write и считывает с помощью read. Создается канал с помощью функции Create_pipe, которая принимает указатель на массив для дескрипторов чтения и записи.

Создается дочерний процесс с помощью fork(). Он открывает файл в режиме для чтения с помощью open и перенаправляет стандартный ввод STDIN_FILENO на него, а стандартный вывод STDOUT_FILENO на канал. С помощью execl текущий процесс перенаправляется на другой, и идет выполнение программы ./child.

Программа child.cpp читает данные из стандартного ввода STDIN_FILENO с помощью read, данные сохраняются в буфер input. Данные в буфере разделяются на отдельные строки по символу «\n». Далее, пока не достигнут конец ввода, для каждой строки создается объект std::istringstream, который позволяет обрабатывать строку как поток данных. Из строки извлекаются числа с помощью оператора >>, и вычисляется их сумма. В буфер output записываются суммы чисел с помощью snprintf. Результат выводится в стандартный вывод из буфера с помощью write.

Родительский процесс читает данные из канала в цикле с помощью read, пока он не станет пустым. На экран выводятся прочитанные данные. Далее ожидается завершение дочернего процесса.

Код программы

parent.cpp

```cpp
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <cstdlib>
#include <cstring>
#include <cstdio>

void Create_pipe(int *fd);

int main() {
    char file[256];
    ssize_t read_bytes;
    int pipe1[2];

    write(STDOUT_FILENO, "Enter the file name: ", 21);
    read_bytes = read(STDIN_FILENO, file, sizeof(file) - 1);

    if (read_bytes > 0) {
        file[read_bytes - 1] = '\0';
    } else {
        perror("Problems with the name of file");
        exit(EXIT_FAILURE);
    }

    Create_pipe(pipe1);

    pid_t pid = fork();
    if (pid == -1) {
        perror("Error with fork\n");
        exit(EXIT_FAILURE);

    }
    else if (pid == 0) {
        close(pipe1[0]);

        int file_fd = open(file, O_RDONLY);
        if (file_fd == -1) {
            perror("Problems with opening file\n");
            exit(EXIT_FAILURE);
        }

        dup2(file_fd, STDIN_FILENO);
        dup2(pipe1[1], STDOUT_FILENO);

        close(file_fd);
        close(pipe1[1]);

        execl("./child", "./child", nullptr);
```

```
        perror("Error with execl\n");
        exit(EXIT_FAILURE);
    }
    else {
        close(pipe1[1]);

        char buffer[256];
        while ((read_bytes = read(pipe1[0], buffer, sizeof(buffer) - 1)) > 0) {
            buffer[read_bytes] = '\0';
            write(STDOUT_FILENO, buffer, read_bytes);
        }

        close(pipe1[0]);

        int status;
        waitpid(pid, &status, 0);
    }

    return 0;
}

void Create_pipe(int* fd) {
    if (pipe(fd) == -1) {
        perror("Error with pipe\n");
        exit(EXIT_FAILURE);
    }
}
```

child.cpp

```
#include <unistd.h>
#include <sstream>
#include <string>
#include <cstring>
#include <iostream>

int main() {

    char input[256];
    ssize_t read_bytes;

    while ((read_bytes = read(STDIN_FILENO, input, sizeof(input) - 1)) > 0) {
        input[read_bytes] = '\0';

        char* line = strtok(input, "\n");

        while (line) {
            std::istringstream stream(line);
            float number, sum = 0;

            while (stream >> number) {
                sum += number;
            }

            char output[256];
            int output_len = snprintf(output, sizeof(output), "Sum of elements:
%.2f\n", sum);
            write(STDOUT_FILENO, output, output_len);

            line = strtok(nullptr, "\n");
        }
    }
    return 0;
}
```

Протокол работы программы

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab1$ ./parent

Enter the file name: nums.txt

Sum of elements: 4.56

Sum of elements: 36.61

Sum of elements: 6.66

Sum of elements: 60.12

Sum of elements: 170.99

Sum of elements: 26.10

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab1$ cat nums.txt

3.14 8.52 -7.1

0.01 32.1 3.3 1.2

1.11 2.22 3.33

83.32 -23.2

48.3 99.2 1.1 22.39

1.2 3.4 5.6 7.8 9.1 -1.0

anegamelina@LAPTOP-0ED9K3JN:/mnt/c/Users/Anega/CLionProjects/osi_labs/lab1$ strace ./parent

execve("./parent", ["./parent"], 0x7ffd9c1e1670 /* 26 vars */) = 0

brk(NULL) = 0x558cadfeb000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc96eb6de0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8c54395000

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=19779, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 19779, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8c54390000

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\3\0>\0\1\0\0\0P\237\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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\302\211\332Pq\2439\235\350\223\322\257\201\326\243\f"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, 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, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8c54167000

mprotect(0x7f8c5418f000, 2023424, PROT_NONE) = 0

mmap(0x7f8c5418f000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f8c5418f000

mmap(0x7f8c54324000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f8c54324000

mmap(0x7f8c5437d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f8c5437d000

mmap(0x7f8c54383000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f8c54383000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8c54164000

arch_prctl(ARCH_SET_FS, 0x7f8c54164740) = 0

set_tid_address(0x7f8c54164a10) = 23497

set_robust_list(0x7f8c54164a20, 24) = 0

rseq(0x7f8c541650e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f8c5437d000, 16384, PROT_READ) = 0

mprotect(0x558c826ea000, 4096, PROT_READ) = 0

mprotect(0x7f8c543cf000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f8c54390000, 19779) = 0

write(1, "Enter the file name: ", 21Enter the file name:) = 21

read(0, nums.txt

"nums.txt\n", 255) = 9

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

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f8c54164a10) = 23536

close(4) **= 0**

read(3, "Sum of elements: 4.56\nSum of ele"..., 255) = 137

write(1, "Sum of elements: 4.56\nSum of ele"..., 137Sum of elements: 4.56

Sum of elements: 36.61

Sum of elements: 6.66

Sum of elements: 60.12

Sum of elements: 170.99

Sum of elements: 26.10

) = 137

read(3, "", 255) = 0

close(3) = 0

wait4(23536, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 23536

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=23536, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе написания данной лабораторной работы я научилась работать с системными вызовами в СИ, создавать программы, состоящие из нескольких процессов, и передавать данные между процессами по каналам. Во время отладки программы я познакомилась с утилитой strace, она оказалась достаточно удобной для получения информации о работе многопоточных программ. Лабораторная работа была довольно интересна, так как я раньше не создавала программы на СИ, которые запускают несколько процессов параллельно.