

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: Болдинова В.В.
Преподаватель: Бахарев В.Д.
Оценка: _____
Дата: 01.01.25

Москва, 2024

Постановка задачи

Вариант 2.

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

Общий метод и алгоритм решения

Использованные системные вызовы:

- `write(int fd, const void *buf, size_t count)` - записывает данные в файл или в стандартный поток вывода, используя файловый дескриптор;
- `open(const char *pathname, int flags, mode_t mode)` - открывает файл с заданными правами доступа (для записи в файл);
- `close(int fd)` - закрывает файловый дескриптор, который ранее был открыт;
- `read(int fd, void *buf, size_t count)` - читает данные из файла или стандартного ввода;
- `strtok(char *str, const char *delim)` - разбивает строку на токены (части), используя разделители;
- `strcspn(const char *str1, const char *str2)` - вычисляет индекс первого символа из `str1`, который есть в `str2`;
- `strtof(const char *nptr, char **endptr)` - преобразует строку в число с плавающей точкой (float);
- `CreatePipe(LPHANDLE hReadPipe, LPHANDLE hWritePipe, LPSECURITY_ATTRIBUTES lpPipeAttributes, DWORD nSize)` - создает пару связанных дескрипторов для межпроцессного общения (pipe);
- `WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)` - записывает данные в файл или в pipe;
- `ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped)` - читает данные из файла или из pipe;
- `CreateProcess(LPCTSTR lpApplicationName, LPTSTR lpCommandLine, LPSECURITY_ATTRIBUTES lpProcessAttributes, LPSECURITY_ATTRIBUTES lpThreadAttributes, BOOL bInheritHandles, DWORD dwCreationFlags, LPVOID lpEnvironment, LPCTSTR lpCurrentDirectory, LPSTARTUPINFO lpStartupInfo, LPPROCESS_INFORMATION lpProcessInformation)` - создает новый процесс с указанными параметрами, включая стандартные потоки (вход, выход, ошибки);
- `WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds)` - ожидает завершения процесса или другого объекта синхронизации;
- `CloseHandle(HANDLE hObject)` - закрывает дескриптор объекта, такого как файл или процесс;
- `GetStdHandle(DWORD nStdHandle)` - получает дескриптор для стандартного потока ввода/вывода/ошибок;
- `SetLastError(DWORD dwErrCode)` - устанавливает код ошибки, который может быть получен через `GetLastError()`;

- `snprintf(char *str, size_t size, const char *format, ...)` - форматирует строку с заданными параметрами и записывает её в буфер.

В данной лабораторной работе я написала программу, состоящую из двух процессов: родительского и дочернего, которые взаимодействуют друг с другом с помощью канала (pipe). Родительский процесс запрашивает ввод чисел и передает их дочернему процессу для обработки. Дочерний процесс читает данные из канала, вычисляет сумму введенных чисел каждой новой строки, пока не встретит end, и записывает результаты в указанный файл. Программа включает в себя обработку ошибок, таких как отсутствие аргументов командной строки и сбои при создании процессов и открытии файлов.

Код программы

Parent.c:

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define SIZE_BUF 100
#define SIZE_MSG 100
#define SIZE_CMDLINE 256

void HandleError(const char *message) {
    DWORD errorCode = GetLastError();
    char errorBuffer[SIZE_MSG];
    snprintf(errorBuffer, SIZE_MSG, "%s (error code: %lu)\n", message,
    errorCode);
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), errorBuffer,
    strlen(errorBuffer), NULL, NULL);
    exit(EXIT_FAILURE);
}

int main(int argc, char *argv[]) {
    HANDLE pipeRead, pipeWrite;
    PROCESS_INFORMATION pi;
    STARTUPINFO si;
    char buffer[SIZE_BUF];

    if (argc < 2) {
        const char error_msg[] = "You must specify a file name as an
    argument.\n";
        WriteFile(GetStdHandle(STD_ERROR_HANDLE), error_msg,
    sizeof(error_msg) - 1, NULL, NULL);
        exit(EXIT_FAILURE);
    }

    SECURITY_ATTRIBUTES sa = {sizeof(SECURITY_ATTRIBUTES), NULL, TRUE};

    if (!CreatePipe(&pipeRead, &pipeWrite, &sa, 0)) {
        HandleError("Failed to create pipe");
    }

    ZeroMemory(&si, sizeof(STARTUPINFO));
    si.cb = sizeof(STARTUPINFO);
    si.hStdInput = pipeRead;
```

```

si.hStdOutput = GetStdHandle(STD_OUTPUT_HANDLE);
si.hStdError = GetStdHandle(STD_ERROR_HANDLE);
si.dwFlags |= STARTF_USESTDHANDLES;

ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));

char cmdLine[SIZE_CMDLINE];
size_t len = strlen(argv[1]);
if (len >= sizeof(cmdLine) - 11) {
    const char error_msg[] = "Argument is too long\n";
    WriteFile(GetStdHandle(STD_ERROR_HANDLE), error_msg,
sizeof(error_msg) - 1, NULL, NULL);
    CloseHandle(pipeRead);
    CloseHandle(pipeWrite);
    exit(EXIT_FAILURE);
}

strcpy(cmdLine, "child.exe ");
strcat(cmdLine, argv[1]);

if (!CreateProcess(NULL, cmdLine, NULL, NULL, TRUE, 0, NULL, NULL, &si,
&pi)) {
    HandleError("Failed to create process");
    CloseHandle(pipeRead);
    CloseHandle(pipeWrite);
}

CloseHandle(pipeRead);

while (1) {
    const char prompt[] = "Enter numbers (or 'end' to finish): ";
    if (!WriteFile(GetStdHandle(STD_OUTPUT_HANDLE), prompt,
sizeof(prompt) - 1, NULL, NULL)) {
        HandleError("Failed to write prompt");
    }

    DWORD bytesRead;
    if (!ReadFile(GetStdHandle(STD_INPUT_HANDLE), buffer, sizeof(buffer)
- 1, &bytesRead, NULL)) {
        HandleError("Failed to read input");
    }

    buffer[bytesRead] = '\\0';
    buffer[strcspn(buffer, "\\r\\n")] = '\\0';

    if (strcmp(buffer, "end") == 0) {
        DWORD written;
        if (!WriteFile(pipeWrite, "end\\n", 4, &written, NULL)) {
            HandleError("Failed to write 'end' to pipe");
        }
        CloseHandle(pipeWrite);
        break;
    }

    DWORD written;
    if (!WriteFile(pipeWrite, buffer, strlen(buffer), &written, NULL) ||
!WriteFile(pipeWrite, "\\n", 1, &written, NULL)) {
        HandleError("Failed to write to pipe");
    }
}

WaitForSingleObject(pi.hProcess, INFINITE);
CloseHandle(pi.hProcess);

```

```

        CloseHandle(pi.hThread);

        return 0;
    }

```

Child.c:

```

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

#define SIZE_BUF 4096
#define SIZE_MSG 128

//Функция для обработки ошибок и завершения программы
void HandleError(const char *message) {
    const char error_msg[] = "Error: "; // "Ошибка: "
    write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
    write(STDERR_FILENO, message, strlen(message));
    write(STDERR_FILENO, "\n", 1);
    exit(EXIT_FAILURE);
}

//Функция для записи суммы в файл
void writeSumToFile(const char *filename, float sum) {
    int fd = open(filename, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd == -1) {
        HandleError("opening the file"); // "открытие файла"
    }

    char sum_str[20];
    int len = snprintf(sum_str, sizeof(sum_str), "%.2f\n", sum); // Output
    with 2 decimal places (Вывод с точностью до 2 знаков)
    if (len < 0 || write(fd, sum_str, len) != len) {
        close(fd);
        HandleError("writing to the file"); // "запись в файл"
    }

    if (close(fd) == -1) {
        HandleError("closing the file"); // "закрытие файла"
    }
}

int main(int argc, char *argv[]) {
    char buffer[SIZE_BUF];

    // Check if a filename is provided as an argument (Проверяем, передано ли
    имя файла как аргумент)
    if (argc < 2) {
        const char error_msg[] = "You must specify a file name as an
        argument.\n"; // "Вы должны указать имя файла как аргумент"
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        exit(EXIT_FAILURE);
    }

    char *filename = argv[1];

    while (1) {

```

```

// Читаем ввод от родительского процесса
ssize_t bytesRead = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
if (bytesRead == -1) {
    HandleError("reading input"); // "чтение входных данных"
}

if (bytesRead == 0) { // (EOF) Обнаружен конец ввода
    break;
}

buffer[bytesRead] = '\0';
buffer[strcspn(buffer, "\n")] = 0; // Удаляем символ новой строки

if (strlen(buffer) == 0) {
    continue; // Skip empty input Пропускаем пустой ввод
}

if (strcmp(buffer, "end") == 0) {
    break;
}

char *token;
float sum = 0.0f;
char *endptr;

// Parse and process input tokens (Разбираем и обрабатываем токены
// ввода)
token = strtok(buffer, " ");
while (token != NULL) {
    errno = 0;
    float num = strtod(token, &endptr); // Use strtod for float
    (Используем strtod для работы с float)

    if (errno != 0 || *endptr != '\0') {
        const char error_msg[] = "Invalid number in input.
        Skipping.\n"; // "Недопустимое число во входных данных. Пропуск."
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
    } else {
        sum += num;
    }

    token = strtok(NULL, " ");
}

// Write the computed sum to the file (Записываем вычисленную сумму в
// файл)
writeSumToFile(filename, sum);
}

return 0;
}

```

Протокол работы программы

Тестирование:

PS D:\si\OSI\Lab1\L1> gcc -o child child.c

PS D:\si\OSI\Lab1\L1> gcc -o parent parent.c -lws2_32

PS D:\si\OSI\Lab1\L1> .\parent.exe output.txt

Enter numbers (or 'end' to finish): 1.0

Enter numbers (or 'end' to finish): 1.5 1.5 12.0

Enter numbers (or 'end' to finish): 1.0 1.2 1.3 5.5

Enter numbers (or 'end' to finish): end

| | |
|---|-------|
| 1 | 1.00 |
| 2 | 15.00 |
| 3 | 9.00 |
| 4 | |

Strace:

```
execve("/mnt/d/si/OSI/Lab1/L1/parent", ["/mnt/d/si/OSI/Lab1/L1/parent",  
"/mnt/d/si/OSI/Lab1/L1/output.txt"], 0x7ffc0aca5258 /* 26 vars */) = 0  
brk(NULL) = 0x564afeb5b000  
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,  
0) = 0x7ff9324cd000  
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=20115, ...}) = 0  
mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff9324c8000  
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\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"..., 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"..., 784, 64) = 784  
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =  
0x7ff9322b6000  
mmap(0x7ff9322de000, 1605632, PROT_READ|PROT_EXEC,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7ff9322de000  
mmap(0x7ff932466000, 323584, PROT_READ,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7ff932466000  
mmap(0x7ff9324b5000, 24576, PROT_READ|PROT_WRITE,  
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7ff9324b5000  
mmap(0x7ff9324bb000, 52624, PROT_READ|PROT_WRITE,  
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff9324bb000  
close(3) = 0  
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -  
1, 0) = 0x7ff9322b3000  
arch_prctl(ARCH_SET_FS, 0x7ff9322b3740) = 0  
set_tid_address(0x7ff9322b3a10) = 59117  
set_robust_list(0x7ff9322b3a20, 24) = 0  
rseq(0x7ff9322b4060, 0x20, 0, 0x53053053) = 0  
mprotect(0x7ff9324b5000, 16384, PROT_READ) = 0
```

```

mprotect(0x564acdacd000, 4096, PROT_READ) = 0
mprotect(0x7ff932505000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7ff9324c8000, 20115) = 0
pipe2([3, 4], 0) = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7ff9322b3a10) = 59118
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 (\320\270\320\273\320"..., 68) = 68
read(0, "1\n", 100) = 2
write(4, "1", 1) = 1
write(4, "\n", 1) = 1
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 (\320\270\320\273\320"..., 68) = 68
read(0, "1.2\n", 100) = 4
write(4, "1.2", 3) = 3
write(4, "\n", 1) = 1
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 (\320\270\320\273\320"..., 68) = 68
read(0, "1.3 1.3\n", 100) = 8
write(4, "1.3 1.3", 7) = 7
write(4, "\n", 1) = 1
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 (\320\270\320\273\320"..., 68) = 68
read(0, "1/8\n", 100) = 4
write(4, "1/8", 3) = 3
write(4, "\n", 1) = 1
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 (\320\270\320\273\320"..., 68) = 68
read(0, "1.0\n", 100) = 4
write(4, "1.0", 3) = 3
write(4, "\n", 1) = 1
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 (\320\270\320\273\320"..., 68) = 68
read(0, "end\n", 100) = 4
close(4) = 0
wait4(-1, NULL, 0, NULL) = 59118
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=59118, si_uid=0,
si_status=0, si_utime=0, si_stime=1 /* 0.01 s */} ---
exit_group(0) = ?
+++ exited with 0 +++

```

Вывод

В ходе лабораторной работы я узнала о некоторых системных вызовах и научилась их использовать. Впервые в ходе работы мне пришлось создавать каналы, чтобы с их помощью обменивать данные между процессами. Сложность возникла в том, что было слишком много новой информации, и некоторая сложная для понимания, Из-за чего ушло много времени.