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 (ріре). Родительский процесс запрашивает ввод чисел и передает их дочернему процессу для обработки. Дочерний процесс читает данные из канала, вычиеляет 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 = strtof(token, &endptr); // Use strtof for float (Используем strtof 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

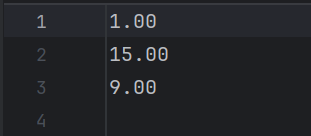
PS D:\si\OSI\Lab1\L1> .\parent.exe output.txt

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



**Strace**:

**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.