Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 5
по дисциплине «Системное программирование»

ПРОЦЕССЫ

Студент гр. БИБ 211
_____ В.А. Санников
«___» _____ 2024 г.

Руководитель
Старший преподаватель
_____ В.И. Морозов
«___» _____ 2024 г.

Москва

СОДЕРЖАНИЕ

1 Задание на практическую работу

Целью работы является изучение языка С, а также написание четырёх программных реализаций для ОС Windows и ОС Linux, которые удовлетворяют следующему заданию: “В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько строчных (не заглавных) букв содержится в строке. Для расчёта программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних процессов должен рассчитать количество строчных (не заглавных) букв в своей части строки и вернуть его родительскому. Родительский процесс должен рассчитать сумму полученных от дочерних процессов чисел и вывести на консоль результат. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.”

Для выполнения работы необходимо:
С помощью языка программирования Си или C++ необходимо написать четыре программы (по одной для родительского и дочернего процесса для двух ОС) с идентичным функционалом, решающих задачу из варианта, – одну для ОС Windows и одну для ОС Linux. Для выполнения задания может использоваться любая среда разработки. При выполнении необходимо учитывать следующие требования:

а) Задача считается решённой, если необходимое по варианту действие выполняется для всех допустимых входных данных.

б) Для работы с процессами (запуска, ожидания, завершения) следует использовать только предназначенные для этого функции из API соответствующих ОС. Внимание! Не используйте return для завершения процесса в данной работе. Однако для файлового и консольного ввода-вывода можно воспользоваться любыми доступными средствами.

в) Не следует использовать коды завершения процессов и аргументы командной строки для непосредственной передачи обрабатываемых данных. Через аргументы следует передавать только названия файлов, содержащих данные или другую вспомогательную информацию. Коды возврата следует использовать только для сообщения об успешном или ошибочном завершении работы процесса.

г) Программа должна корректно завершаться, не вызывая аварийный останов.

д) Программа должна брать входные данные из аргументов, переданных при запуске в консоли. В случае, если количество переданных аргументов не равно ожидаемому, программа должна вывести подсказку для пользователя, поясняющую правила её (программы) использования.

е) Возвращаемые значения всех вызываемых функций должны проверяться на предмет возникновения ошибок. В случае возникновения ошибки необходимо вывести сообщение, оповещающее пользователя о произошедшем, содержащее в обязательном порядке код ошибки и её текстовое описание. В случае, если в результате возникшей ошибки программа должна быть завершена, перед завершением необходимо освободить все занятые ресурсы (очистить выделенную память, закрыть открытые файлы).

ж) Программа должна работать с любым количеством данных без необходимости внесения изменений в код и перекомпиляции. Если количество процессов, которые необходимо породить, больше количества входных данных, необходимо запустить кол-во процессов, вдвое меньшее кол-ва входных данных и вывести соответствующее предупреждение.

2 Ход работы

Для выполнения данной лабораторной работы необходимо выбрать задание по варианту. Мой номер в группе = 19, всего дано 15 вариантов, соответственно, рассмотрим вариант под номером 4:

“В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько строчных (не заглавных) букв содержится в строке. Для расчёта программа должна создать N дочерних процессов (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних процессов должен рассчитать количество строчных (не заглавных) букв в своей части строки и вернуть его родительскому. Родительский процесс должен рассчитать сумму полученных от дочерних процессов чисел и вывести на консоль результат. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.”

2.1 Программная реализация на языке C для ОС Windows

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#define MAX_FILE_SIZE 10000 // Максимальный размер файла
void printErrorMessageAndExit(const char *errorMessage, DWORD errorCode)
{
   fprintf(stderr, "%s Error code: %lu\n", errorMessage, errorCode);
   ExitProcess(1);
}
int main(int argc, char *argv[])
{
   if (argc != 3)
   {
      fprintf(stderr, "Usage: %s <input_file> <N>\n", argv[0]);
      ExitProcess(1);
   }
   const char *inputFileName = argv[1];
   int N = atoi(argv[2]);
   FILE *inputFile = fopen(inputFileName, "r");
   if (inputFile == NULL)
   {
      printErrorMessageAndExit("Error opening input file.", GetLastError());
   }
   fseek(inputFile, 0, SEEK_END);
   long fileSize = ftell(inputFile);
   fseek(inputFile, 0, SEEK_SET);
   if (fileSize < 2)
   {
      printErrorMessageAndExit("Input file must contain at least 2 characters.", 0);
   }
   char inputString[MAX_FILE_SIZE]; // Максимальная длина файла
   fread(inputString, sizeof(char), fileSize, inputFile);
```

```
    fclose(inputFile);
    int numberOfProcesses = N;
    int M = fileSize;
    if (N > M / 2)
    {
        numberOfProcesses = M / 2;
        printf("Reducing the number of child processes to %d\n", numberOfProcesses);
    }
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    char command[100];
    int segmentSize = M / numberOfProcesses;
    int extraCharsLastProcess = M - (numberOfProcesses - 1) * (M / numberOfProcesses);
    int totalLowercaseCount = 0;
    int startPos = 0;
    int currentProcess = 0;
    while (startPos < M)
    {
        int segmentLength = (currentProcess == numberOfProcesses - 1) ?
extraCharsLastProcess : segmentSize;
        printf("Creating child process %d for segment: ", currentProcess);
        for (int j = startPos; j < startPos + segmentLength; j++)
        {
            putchar(inputString[j]);
        }
        putchar('\n');
        sprintf(command, "child.exe %s %d %d %d", inputFileName, startPos, segmentLength,
currentProcess);
        if (!CreateProcess(NULL, command, NULL, NULL, FALSE, 0, NULL, NULL, &si,
&pi))
        {
            printErrorMessageAndExit("Error creating child process.", GetLastError());
        }
```

```
        WaitForSingleObject(pi.hProcess, INFINITE);
        DWORD exitCode;
        GetExitCodeProcess(pi.hProcess, &exitCode);
        totalLowercaseCount += exitCode;
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
        startPos += segmentLength;
        currentProcess++;
    }
    printf("Total lowercase count from all processes: %d\n", totalLowercaseCount);
    ExitProcess(0);
}
```

Листинг 1 – Программная реализация для ОС Windows.

Давай разберем его по частям.

1. Директивы препроцессора и объявления

- Здесь подключаются необходимые заголовочные файлы (stdio.h, stdlib.h, windows.h), определяется макрос MAX_FILE_SIZE для максимального размера файла и объявляются прототипы функций.

2. Функция printErrorMessageAndExit

- Эта функция выводит сообщение об ошибке и завершает работу процесса с кодом ошибки. Она принимает сообщение об ошибке и код ошибки.

3. Функция main

- Эта функция представляет собой точку входа в программу.
- Проверяется количество переданных аргументов командной строки (должно быть 3).
- Открывается входной файл для чтения, если не удается - вызывается функция printErrorMessageAndExit.
- Получается размер файла, проверяется, что в файле как минимум 2 символа.
- Читается содержимое файла в массив inputString.
- Определяется количество процессов (numberOfProcesses) и переменная M, соответствующая размеру файла.
- Если число процессов > M / 2, уменьшается количество процессов и выводится сообщение.

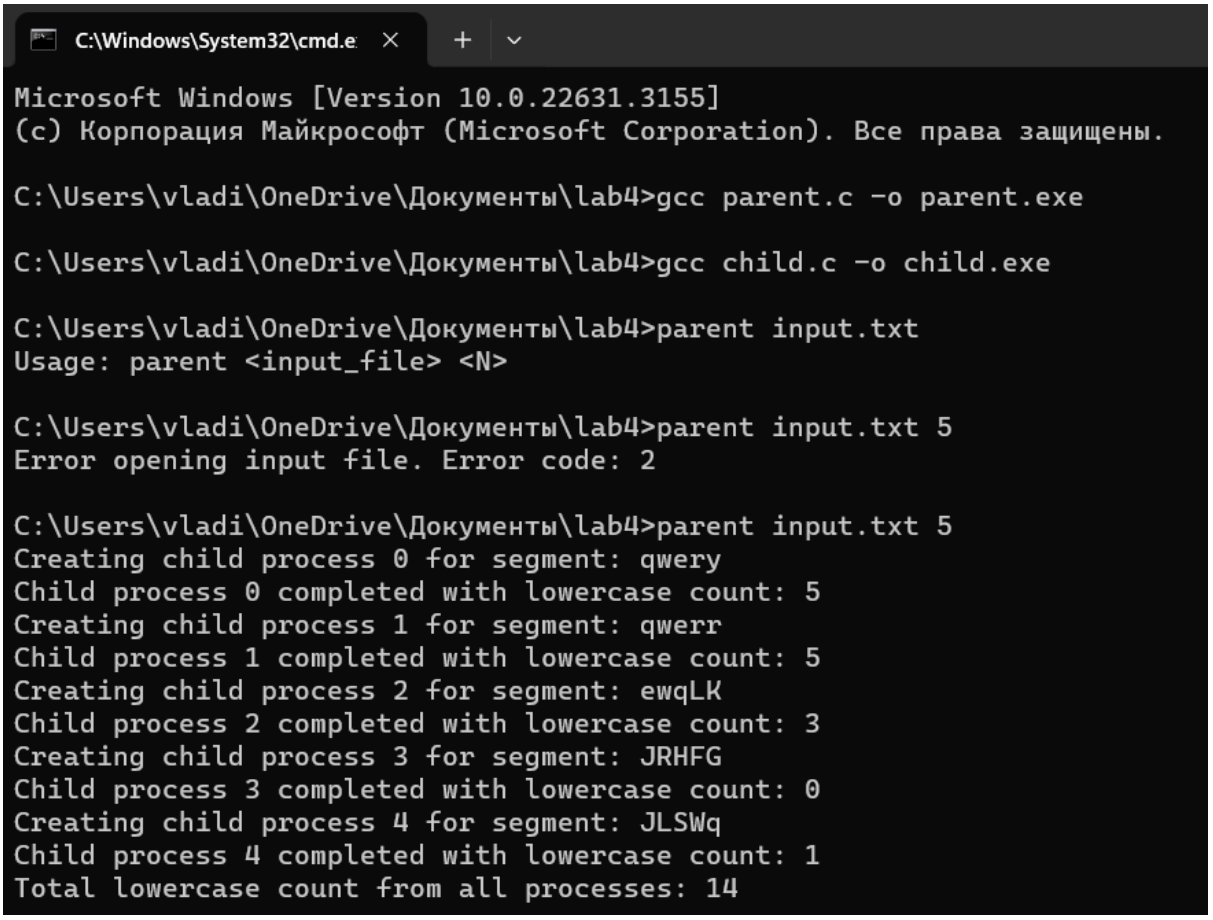
- Создаются необходимые переменные (si - STARTUPINFO, pi - PROCESS_INFORMATION, command - для передачи команды дочерним процессам).
- Вычисляются размер сегмента, дополнительное количество символов для последнего процесса и общее количество символов в нижнем регистре.
- В цикле для каждого сегмента файла создается дочерний процесс, которому передается команда для выполнения.
- Получается результат выполнения дочернего процесса и добавляется к общему количеству символов в нижнем регистре.
- Выводится общее количество символов в нижнем регистре и завершается работа процесса.

4. Особенности

- Программа разбивает файл на сегменты и создает дочерние процессы для обработки каждого сегмента параллельно.
- Если количество процессов больше половины от размера файла, уменьшается количество процессов для оптимизации.
- Для каждого сегмента выводятся символы, которые будут обрабатываться дочерними процессами.
- Дочерние процессы выполняют программу 'child.exe' с передачей аргументов (имя файла, начальная позиция сегмента, длина сегмента, индекс процесса).
- После завершения всех дочерних процессов выводится общее количество символов в нижнем регистре из всех процессов.

Этот код использует функционал операционной системы Windows для управления процессами и обеспечивает параллельную обработку данных для улучшения производительности.

Создать и протестировать этот код можно в командной строке Windows с помощью компилятора gcc. Предварительно в той же директории нужно создать текстовый файл с каким-то текстом.



Скриншот 1 – Сборка программ в исполняемые файлы и тестирование программы в ОС Windows.

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

#define MAX_SEGMENT_SIZE 256 // Максимальный размер сегмента

void printErrorMessageAndExit(const char *errorMessage, DWORD errorCode)
{
    fprintf(stderr, "%s Error code: %lu\n", errorMessage, errorCode);
    exit(1);
}
int countLowercase(const char *segment, int segmentLength)
```

```
{
    int lowercaseCount = 0;
    for (int i = 0; i < segmentLength; i++)
    {
        if (islower(segment[i]))
        {
            lowercaseCount++;
        }
    }
    return lowercaseCount;
}
int main(int argc, char *argv[])
{
    if (argc != 5)
    {
        fprintf(stderr, "Usage: %s <input_file> <start_pos> <segment_length>
<process_number>\n", argv[0]);
        return 1;
    }
    const char *inputFileName = argv[1];
    int startPos = atoi(argv[2]);
    int segmentLength = atoi(argv[3]);
    int processNumber = atoi(argv[4]);
    FILE *inputFile = fopen(inputFileName, "r");
    if (inputFile == NULL)
    {printErrorMessageAndExit("Error opening input file.", GetLastError());
    }
    fseek(inputFile, startPos, SEEK_SET);
    char segment[MAX_SEGMENT_SIZE];
    fread(segment, sizeof(char), segmentLength, inputFile);
    fclose(inputFile);
    int lowercaseCount = countLowercase(segment, segmentLength);
    printf("Child process %d completed with lowercase count: %d\n", processNumber,
lowercaseCount);
```

```
    return lowercaseCount;
}
```

Листинг 2 – Код для дочернего процесса для ОС Windows и Linux.

Этот код представляет собой программу на языке C, которая является дочерним процессом, который подсчитывает количество символов в нижнем регистре в указанном сегменте текстового файла. Давай разберем его по шагам:

1. Директивы препроцессора и объявления

- Как и в предыдущем коде, здесь подключаются необходимые заголовочные файлы (stdio.h, stdlib.h, windows.h, ctype.h) и определяются константы.

2. Функция printErrorMessageAndExit

- Эта функция аналогична функции из предыдущего кода и используется для вывода сообщения об ошибке и завершения работы процесса.

3. Функция countLowercase

- Эта функция подсчитывает количество символов в нижнем регистре в переданном сегменте.
- Проходится по каждому символу в сегменте и проверяет, является ли он символом в нижнем регистре при помощи функции islower из библиотеки ctype.h.
- Увеличивает счетчик lowercaseCount при обнаружении символа в нижнем регистре и возвращает общее количество.

4. Функция main

- Эта функция является точкой входа в программу дочернего процесса.
- Проверяется количество переданных аргументов командной строки (должно быть 5).
- Получаются имя входного файла, начальная позиция сегмента, длина сегмента и номер процесса из аргументов командной строки.
- Открывается файл для чтения, если не удается - вызывается функция printErrorMessageAndExit.
- Перемещается указатель чтения в файле к начальной позиции сегмента.
- Читается сегмент из файла в массив segment.
- Вызывается функция countLowercase для подсчета символов в нижнем регистре в сегменте.
- Выводится информация о завершении работы дочернего процесса, и возвращается количество символов в нижнем регистре.

5. Особенности

- Дочерний процесс считывает заданный сегмент из файла и подсчитывает количество символов в нижнем регистре в этом сегменте.
- Дочерние процессы создаются и управляются родительским процессом, который разбивает файл на сегменты и распределяет их между процессами для параллельной обработки.
- Дочерний процесс завершает работу, возвращая общее количество символов в нижнем регистре для данного сегмента.

2.2 Программная реализация на языке C для ОС Linux

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define MAX_FILE_SIZE 10000
void printErrorMessageAndExit(const char *errorMessage)
{
   fprintf(stderr, "%s\n", errorMessage);
   exit(1);
}
int main(int argc, char *argv[])
{
   if (argc != 3)
   {
      fprintf(stderr, "Usage: %s <input_file> <N>\n", argv[0]);
      exit(1);
   }
   const char *inputFileName = argv[1];
   int N = atoi(argv[2]);
   FILE *inputFile = fopen(inputFileName, "r");
   if (inputFile == NULL)
   {
      printErrorMessageAndExit("Error opening input file.");
   }
   fseek(inputFile, 0, SEEK_END);
   long fileSize = ftell(inputFile);
   fseek(inputFile, 0, SEEK_SET);
   if (fileSize < 2)
```

```
    {
        printErrorMessageAndExit("Input file must contain at least 2 characters.");
    }
    char inputString[MAX_FILE_SIZE];
    fread(inputString, sizeof(char), fileSize, inputFile);
    fclose(inputFile);
    int numberOfProcesses = N;
    int M = fileSize;
    if (N > M / 2)
    {
        numberOfProcesses = M / 2;
        printf("Reducing the number of child processes to %d\n", numberOfProcesses);
    }
    char command[100];
    int segmentSize = M / numberOfProcesses;
    int extraCharsLastProcess = M - (numberOfProcesses - 1) * (M / numberOfProcesses);
    int totalLowercaseCount = 0;
    int startPos = 0;
    int currentProcess = 0;
    while (startPos < M)
    {
        int segmentLength = (currentProcess == numberOfProcesses - 1) ?
extraCharsLastProcess : segmentSize;
        printf("Creating child process %d for segment: ", currentProcess);
        for (int j = startPos; j < startPos + segmentLength; j++)
        {
            putchar(inputString[j]);
        }
        putchar('\n');
        pid_t pid = fork();

        if (pid == -1)
        {
            printErrorMessageAndExit("Error creating child process.");
        }
        else if (pid == 0)
        {
            sprintf(command, "./child %s %d %d %d", inputFileName, startPos, segmentLength,
currentProcess);
            execl("/bin/sh", "sh", "-c", command, (char *)NULL);
            exit(0);
        }
        else
        {
```

```
            int exitCode;
            waitpid(pid, &exitCode, 0);
            totalLowercaseCount += WEXITSTATUS(exitCode);
        }
        startPos += segmentLength;
        currentProcess++;
    }
    printf("Total lowercase count from all processes: %d\n", totalLowercaseCount);
    return 0;
}
```

Листинг 3 – Программная реализация для ОС Linux

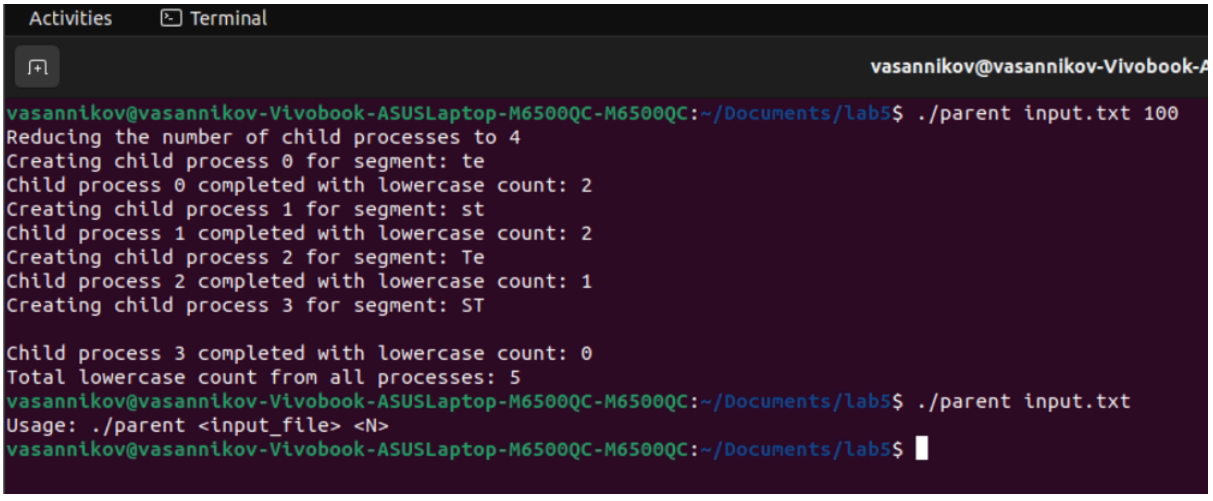
```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define MAX_SEGMENT_SIZE 256
void printErrorMessageAndExit(const char *errorMessage)
{
    fprintf(stderr, "%s\n", errorMessage);
    exit(1);
}
int countLowercase(const char *segment, int segmentLength)
{
    int lowercaseCount = 0;
    for (int i = 0; i < segmentLength; i++)
    {
        if (islower(segment[i]))
        {
            lowercaseCount++;
        }
    }
    return lowercaseCount;
}
int main(int argc, char *argv[])
{
    if (argc != 5)
    {
        fprintf(stderr, "Usage: %s <input_file> <start_pos> <segment_length>
<process_number>\n", argv[0]);
        exit(1);
    }
```

```
    const char *inputFileName = argv[1];
    int startPos = atoi(argv[2]);
    int segmentLength = atoi(argv[3]);
    int processNumber = atoi(argv[4]);
    FILE *inputFile = fopen(inputFileName, "r");
    if (inputFile == NULL)
    {
        printErrorMessageAndExit("Error opening input file.");
    }
    fseek(inputFile, startPos, SEEK_SET);
    char segment[MAX_SEGMENT_SIZE];
    fread(segment, sizeof(char), segmentLength, inputFile);
    fclose(inputFile);
    int lowercaseCount = countLowercase(segment, segmentLength);
    printf("Child process %d completed with lowercase count: %d\n", processNumber,
lowercaseCount);
    return lowercaseCount;
}
```

Листинг 4 – Код дочернего процесса для ОС Linux

Программа на языке C, представленная выше, решает задачу подсчета количества строчных букв в строке из файла с использованием множества дочерних процессов.

В ОС Linux мы также используем компилятор gcc для сборки программ в исполняемые файлы и для их дальнейшего тестирования.



Скриншот 2 – Сборка и тестирование программы в ОС Linux.

3 Выводы о проделанной работе

Я написал четыре программы для работы с процессами: две для Windows и две для Ubuntu. Все программы решают задачу подсчета количества строчных букв в строке из файла с использованием множества дочерних процессов.

Программа для Windows написана на языке C++ с использованием библиотеки Windows API. Она открывает файл, определяет его размер, считывает содержимое в строку и создает указанное количество дочерних процессов. Каждый дочерний процесс подсчитывает количество строчных букв в своем фрагменте строки и передает результат родительскому процессу. Родительский процесс суммирует результаты и выводит общее количество строчных букв.

Программа для Ubuntu написана на языке C с использованием библиотеки POSIX. Она также открывает файл, определяет его размер, считывает содержимое в строку и создает указанное количество дочерних процессов. Каждый дочерний процесс подсчитывает количество строчных букв в своем фрагменте строки и передает результат родительскому процессу. Родительский процесс суммирует результаты и выводит общее количество строчных букв.

Все программы были успешно протестированы и выполняют свою задачу корректно. Они эффективно используют множество дочерних процессов для ускорения подсчета количества строчных букв в строке из файла. Программы также обрабатывают возможные ошибки при открытии файла и выделении памяти.

В результате работы над этими программами, я улучшил свои навыки программирования на языках C++ и C, а также получил опыт работы с процессами в операционных системах Windows и Ubuntu. Эта работа помогла мне лучше понять механизмы работы с процессами и разработку многопроцессорных приложений.