Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

К ЗАЩИТЕ ДОПУСТИТЬ
_____ Поденок Л.П.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

УТИЛИТА СБОРА ИНФОРМАЦИИ О СИСТЕМЕ

БГУИР КП 1-40 02 01 104 ПЗ

Студент: гр. 150501 Гаращук Н.В.

Руководитель: Поденок Л.П.

Минск 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
_____Б.В. Никульшин
(подпись)
« » _____2023г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Гаращуку Никите Васильевичу.*

1. Тема проекта *Утилита сбора информации о системе.*
2. Срок сдачи студентом законченного проекта *17 мая 2023 г.*
3. Исходные данные к проекту *Язык программирования – C, библиотека GTK3.0.*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Результаты. Заключение Список используемых источников
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. Блок-схема функции `kernel_info();`
2. Блок-схема функции `proc_info()`.
6. Консультант по проекту *Поденок Л. П.*
7. Дата выдачи задания *24 февраля 2023 г.*
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
разделы 1 к 1 марта 2023 г. – 20 %;
разделы 2, 3 к 1 апреля 2023 г. – 30 %;
разделы 4, 5 к 1 мая 2023 г. – 30 %;
оформление пояснительной записки и графического материала к 15 мая 2023 г. 20 %
Защита курсового проекта с 29 мая 2023 по 09 июня 2023г.

РУКОВОДИТЕЛЬ _____ *Л. П. Поденок*

(подпись)

Задание принял к исполнению_____ *Н. В. Гаращук*

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Язык программирования C — универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней. Большинство программистов предпочитают использовать язык C для серьезных разработок потому, что их привлекают такие особенности языка, как свобода выражения мыслей, мобильность и чрезвычайная доступность.

Язык C — это универсальный язык программирования, для которого характерны экономичность выражения, современный поток управления и структуры данных, богатый набор операторов. Язык C не является ни языком "очень высокого уровня", ни "большим" языком, и не предназначается для некоторой специальной области применения, но отсутствие ограничений и общность языка делают его более удобным и эффективным для многих задач, чем языки, предположительно более мощные.

Язык C, первоначально предназначавшийся для написания операционной системы «UNIX» на ЭВМ DEC PDP-11, был разработан и реализован на этой системе Деннисом Ричи. Операционная система, компилятор с языка C и, по существу, все прикладные программы системы «UNIX» написаны на C. Язык C, однако, не связан с какими-либо определенными аппаратными средствами или системами, и на нем легко писать программы, которые можно пропускать без изменений на любой ЭВМ, имеющей C-компилятор.

Язык C также используется при составлении программ для микроконтроллеров.

Язык C оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования как C++, C#, Java, PHP и др.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ существующих аналогов

Тема курсового проекта была выбрана в первую очередь для углубления знаний по языку C и в области проектирования утилит, поэтому моей целью не является разработать конкурентоспособный продукт.

Тем не менее, чтобы создать корректно работающее приложение, нужно иметь представление о существующих аналогах, об их недостатках, преимуществах и реализованных внутри функциях.

1.1.1 Утилита `vmstat`

Наиболее эффективным средством для оценки необходимого объема ресурсов является команда `vmstat`, которая предоставляет информацию о загруженности процессора, интенсивности операций дискового ввода-вывода и использовании оперативной памяти. Пример работы этой утилиты показан ниже.

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 1884628 103000 3099216    0    0   166    79  420 1110 11  3 83  4  0
```

1.1.2 Утилита `lshw`

Утилита `lshw` используется для получения информации об аппаратном обеспечении компьютера. Данная утилита позволяет выводить информацию об отдельных классах устройств, использовать различное форматирование вывода, а также удалять из вывода информацию, позволяющую идентифицировать устройства (серийные номера и подобные идентификаторы). Пример работы этой утилиты показан ниже.

```
$ lshw -class processor
 *-cpu
      продукт: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
      производитель: Intel Corp.
      сведения о шине: cpu@0
      версия: 6.158.9
      размер: 3227MHz
      capacity: 3500MHz
      разрядность: 64 bits
```

1.2 Постановка задачи

Проанализировав аналоги можно понять, что программа “Утилита сбора информации о системе” должна иметь удобный пользовательский интерфейс. Утилита должны выполнять следующие команды:

- Вывод информации о ОС и ее версии;
- Вывод информации об объем памяти:
- Вывод информация о жестких дисках;
- Вывод информация о подключенных USB-устройствах;
- Вывод полной информации о процессоре;
- Вывод о работающих процессах;
- Вывод информации об ядре.

В программе должны быть использованы как системные утилиты, так и чтение различных системных файлов. В качестве языка программирования выбран C, по причине быстрой производительности, требуемой для обработки большого количества объектов и наличия опыта в использования данного языка.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость программного продукта в будущем путем добавления новых блоков.

2.1 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса предназначен для взаимодействия пользователя с приложением, путем представления и визуализации данных.

Для разрабатываемого проекта создается простейшей пользовательский интерфейс с помощью библиотеки `GTK3.0`. Данный интерфейс представляет собой набор кнопок, с которыми может взаимодействовать пользователь.

2.2 Модуль получения информации о системе

Модуль получения информации о системе предназначен для получения общей информации о всей системе. В общую информацию могут входить такая информация, как название операционной системы, информация о процессоре и количество ядер, тип машины(ее разрядность), `id` пользователя, `id` группы и информация об оперативной памяти.

2.3 Модуль получения информации об ядре

Данный модуль предоставляет пользователю подробную информацию об ядре. В эту информацию входят имя узла, выпуск ядра, его версия и информация об установленных пакетах в ядре.

2.4 Модуль получения информации о жестких дисках

Модуль получения информации о жестких дисках предоставляет информацию о блочных устройствах в системе. Она показывает список устройств и их свойства, такие как размер, тип файловой системы и точки монтирования.

2.5 Модуль получения информации о процессоре

Данный модуль предоставляет пользователю подробную информацию о процессоре и его характеристиках, а именно количестве ядер процессора, его тактовой частоте, кэше, размере адресов и других параметрах.

2.6 Модуль получения информации о подключенных устройствах

Модуль получения информации о подключенных устройствах предоставляет информацию о всех подключенных к компьютеру устройствах USB. Информация содержит следующее: VID (Vendor ID), PID (Product ID), информация о производителе, модели и серийном номере устройства. Кроме того, можно получить информацию о скорости передачи данных, типе устройства и его интерфейсе.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

3.1 Структура `cpu_info`

Данная структура содержит информацию о процессоре. Она включает в себя следующие поля:

- `char cpu_product[128]` — название процессора;
- `char *cpu_vendor[128]` — производитель;
- `char *cpu_family[128]` — семья процессора;
- `char *cpu_cores[128]` — количество ядер;
- `char *cpu_microcode[128]` — микрокод процессора;
- `char *cpu_cache[128]` — кеш процессора;
- `char *cpu_bogomips[128]` — скорость исполнения;
- `char *cpu_addres_size[128]` - размеры адресов.

3.2 Структура `modul`

Данная структура содержит информацию о модуле ядра. Она включает в себя следующие поля:

- `char name[128]` — название пакета;
- `char size [128]` — размер пакета;
- `char used_by[128]` — название других пакетов которые используют этот пакет.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Функция `get_system_info()`

1) начало.
2) объявление переменных `utsname uname_data`, `sysinfo sysinfo_data`.
3) получение общесистемную статистики: вызов функции `sysinfo(&sysinfo_data)`.
4) если функция `sysinfo` вернула значение -1, переход на пункт 9, иначе на пункт 5.
5) получение информации о системе: вызов функции `uname(&uname_data)`.
6) если функция `uname` вернула значение -1, переход на пункт 9, иначе на пункт 7.
7) вывод структуры `uname_data`.
8) вывод структуры `sysinfo_data`.
9) конец.

4.2 Функция `get_modules_info()`

1) начало.
2) объявление переменных: `FILE *fp`, `char line[256]`, `struct modul ker_modul`.
3) открытие файла `fp` с названием «`/proc/modules`».
4) если файл `fp` не удалось открыть, переход на пункт 9, иначе на пункт 5.
5) чтение из файла `fp` в `line`.
6) заполнение структуры `ker_modul` из `line`.
7) вывод структуры `ker_modul`.
8) если конец файла `fp` переход на пункт 9, иначе на пункт 5.
9) конец.

4.3 Схема алгоритма `kernel_info()`

Схема алгоритма `kernel_info()` показана в приложении А.

4.4 Схема алгоритма `proc_info()`

Схема алгоритма `proc_info()` показана в приложении Б.

5 РЕЗУЛЬТАТ РАБОТЫ

На рисунке 5.1 представлено окно которое появляется при запуске программы.

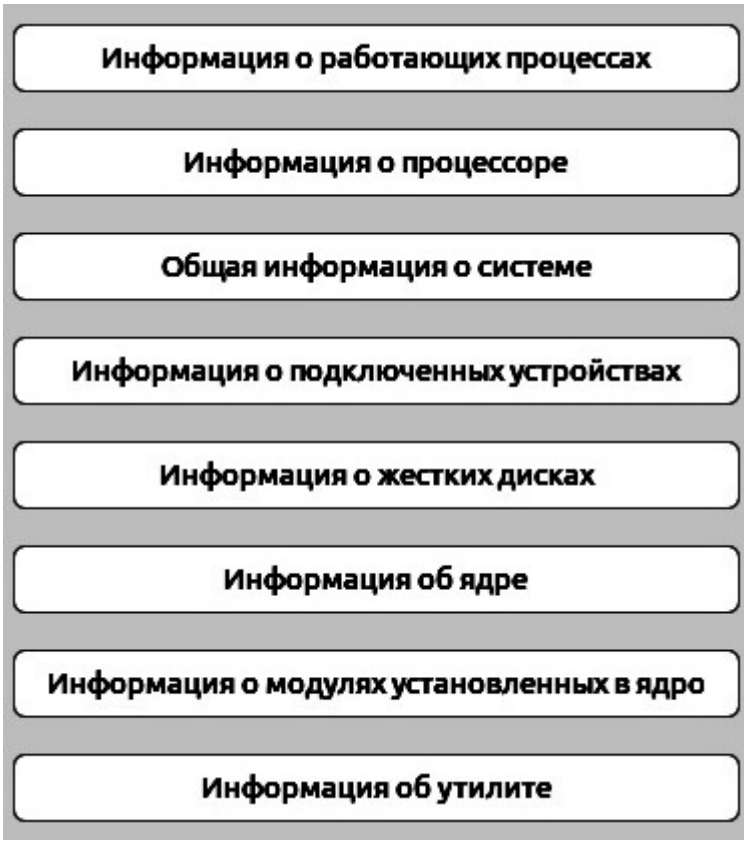


Рисунок 5.1 — окно программы «Утилита сбора информации»

При нажатии кнопки в консоли выводится выводится соответствующая информация. Ниже представлен примеры.

Пример нажатия кнопки «Общая информация о системе»:

```
$ ./main

Operating system: Linux 5.19.0-41-generic
CPU: Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
CPU temperature: 44,5 C
Machine type: x86_64
User ID: 1000
Group ID: 1000
Effective user ID: 1000
Effective group ID: 1000
Number of processors: 4
Total RAM: 7840 MB
```

```
    Free RAM: 268 MB
```

Пример нажатия кнопки «Информация о подключенных устройствах»:

```
    Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
    Bus 001 Device 003: ID 5986:06b0 Acer, Inc EasyCamera
    Bus 001 Device 002: ID 1ea7:0064 SHARKOON Technologies GmbH 2.4GHz
Wireless rechargeable vertical mouse [More&Better]
    Bus 001 Device 004: ID 8087:0a2b Intel Corp. Bluetooth wireless interface
    Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Пример нажатия кнопки «Информация об ядре»:

```
    Operating System: Linux
    Node Name: NikitoskaLinux
    Kernel Release: 5.19.0-41-generic
    Kernel Version: #42~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue
Apr 18 17:40:00 UTC 2
    Machine Hardware Name: x86_64
```

Пример нажатия кнопки «Информация о модулях установленных в ядро»:

```
    Загруженные модули:
    ntfs3                                282624          -
    rfcomm                                86016          -
    ccm                                   20480          -
    cmac                                  16384          -
    algif_hash                            16384          -
    algif_skcipher                        16384          -
    bnep                                  28672          -
    binfmt_misc                           24576          -
    snd_hda_codec_hdmi                    86016          -
    snd_hda_codec_realtek                163840          -
    intel_rapl_msr                        20480          -
    i915                                3125248          -
    snd_hda_intel                         53248          -
    intel_tcc_cooling                     16384          -
    x86_pkg_temp_thermal                  20480          -
    intel_powerclamp                      24576          -
    drm_buddy                             20480 i915,
    kvm_intel                            434176          -
    rc_core                               65536 cec,
    aesni_intel                          376832          -
    input_leds                            16384          -
    crypto_simd                           16384 aesni_intel,
    libarc4                               16384 mac80211,
    videobuf2_vmalloc                     20480 uvcvideo,
```

```
videobuf2_memops                       20480 videobuf2_vmalloc,
snd_seq_midi                           20480          -
snd_seq_midi_event                     16384 snd_seq_midi,
snd_rawmidi                            45056 snd_seq_midi,
rc_core                                65536 cec,
aesni_intel                           376832          -
input_leds                             16384          -
```

Пример нажатия кнопки «Информация о жестких дисках»:

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0         7:0    0     4K  1 loop /snap/bare/5
loop1         7:1    0  63,3M  1 loop /snap/core20/1852
loop2         7:2    0  63,3M  1 loop /snap/core20/1879
loop3         7:3    0    73M  1 loop /snap/core22/607
loop4         7:4    0    73M  1 loop /snap/core22/617
loop5         7:5    0 241,9M  1 loop /snap/firefox/2559
loop6         7:6    0 241,9M  1 loop /snap/firefox/2579
loop7         7:7    0 349,7M  1 loop /snap/gnome-3-38-2004/137
loop8         7:8    0 349,7M  1 loop /snap/gnome-3-38-2004/140
loop10        7:10   0 460,6M  1 loop /snap/gnome-42-2204/99
loop11        7:11   0  91,7M  1 loop /snap/gtk-common-themes/1535
loop12        7:12   0  45,9M  1 loop /snap/snap-store/582
loop13        7:13   0  45,9M  1 loop /snap/snap-store/638
loop14        7:14   0  53,2M  1 loop /snap/snapd/18933
loop15        7:15   0  53,2M  1 loop /snap/snapd/19122
loop16        7:16   0   304K  1 loop /snap/snapd-desktop-integration/49
loop17        7:17   0   428K  1 loop /snap/snapd-desktop-integration/57
loop18        7:18   0 460,6M  1 loop /snap/gnome-42-2204/102
sda           8:0    0 931,5G  0 disk
├─sda1        8:1    0  97,7G  0 part
├─sda2        8:2    0  19,5G  0 part /media/nikitagara/Новый том
├─sda3        8:3    0  93,1G  0 part /var/snap/firefox/common/host-hunspell
│                                    /
└─sda4        8:4    0  18,6G  0 part /home
nvme0n1     259:0    0 232,9G  0 disk
├─nvme0n1p1 259:1    0   100M  0 part /boot/efi
├─nvme0n1p2 259:2    0    16M  0 part
├─nvme0n1p3 259:3    0 232,3G  0 part
└─nvme0n1p4 259:4    0   521M  0 part
```

При реализации программы была использованная библиотека `GTK3.0`. Для корректной работы программы необходимо установить эту библиотеку. Также в проекте используются следующий системные утилиты: `lsblk` и `lsusb`.

Программа запускается с помощью `makefile`:

```
CC = gcc
CFLAGS = -W -Wall -Wno-unused-parameter -Wno-unused-variable -std=c11 -pedantic
.PHONY : clean
```

```
all: main

main: ../src/util.c makefile
    $(CC) $(CFLAGS) `pkg-config --cflags gtk+-3.0` -o main ../src/util.c `pkg-config --libs gtk+-3.0`

clean:
      rm -f main
```

Код программы показан в приложении В.

ЗАКЛЮЧЕНИЕ

В результате работы над данным курсовым проектом было разработано работоспособное приложение со своим набором функций. Данный курсовой проект был разработан в соответствии с поставленными задачами, весь функционал был реализован в полном объеме. Для создания программного продукта была подробно исследована технология создания утилиты сбора информации. В ходе разработки были углублены знания языка программирования C, а также приобретён опыт использования библиотеки GTK3.0.

В проекте были использованы системные утилиты и чтение различных системных файлов. Язык программирования C был выбран из-за его быстрой производительности и наличия опыта в его использовании. В результате, данная программа обеспечит пользователям удобный и быстрый доступ к информации о состоянии и характеристиках компьютерной системы.

В дальнейшем планируется усовершенствование текущего функционала приложения, путем добавления новых функций и модулей, а также добавления возможности работы под разными системами.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

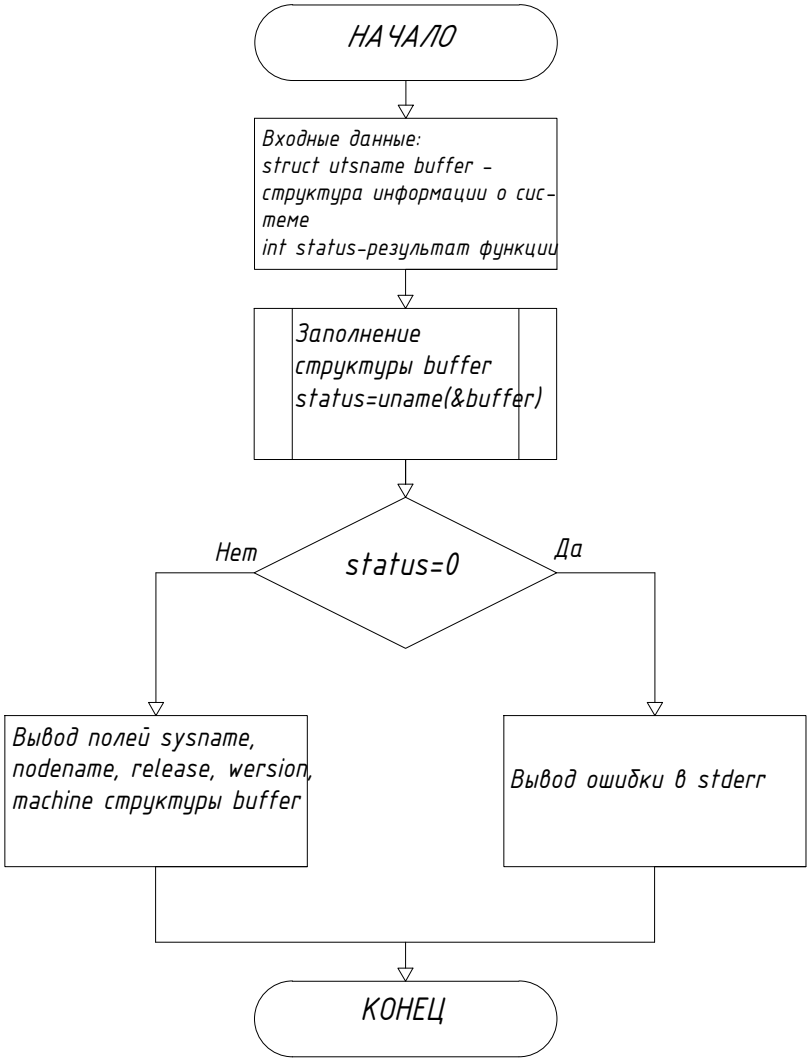
[1] Роберт Лав. Linux. Системное программирование. 2-е изд. – Спб.: Питер, 2014 – 448с.

[2] Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]. – Минск, БГУИР 2019.

ПРИЛОЖЕНИЕ А

(обязательно)

Блок-схема функции `kernel_info()`



ГУИР.400201.104 ПД.1

Изм	Лист	N° докум.	Подп.	Дата
Разраб.		Гаращук		
Пров.		Поденок		
Т.контр.				
Н.контр.				
Утв.				

Схема алгоритма функции kernel_info()
информация о ядре

Лит.	Масса	Масштаб

Лист	Листов

ЭВМ, гр. 150501

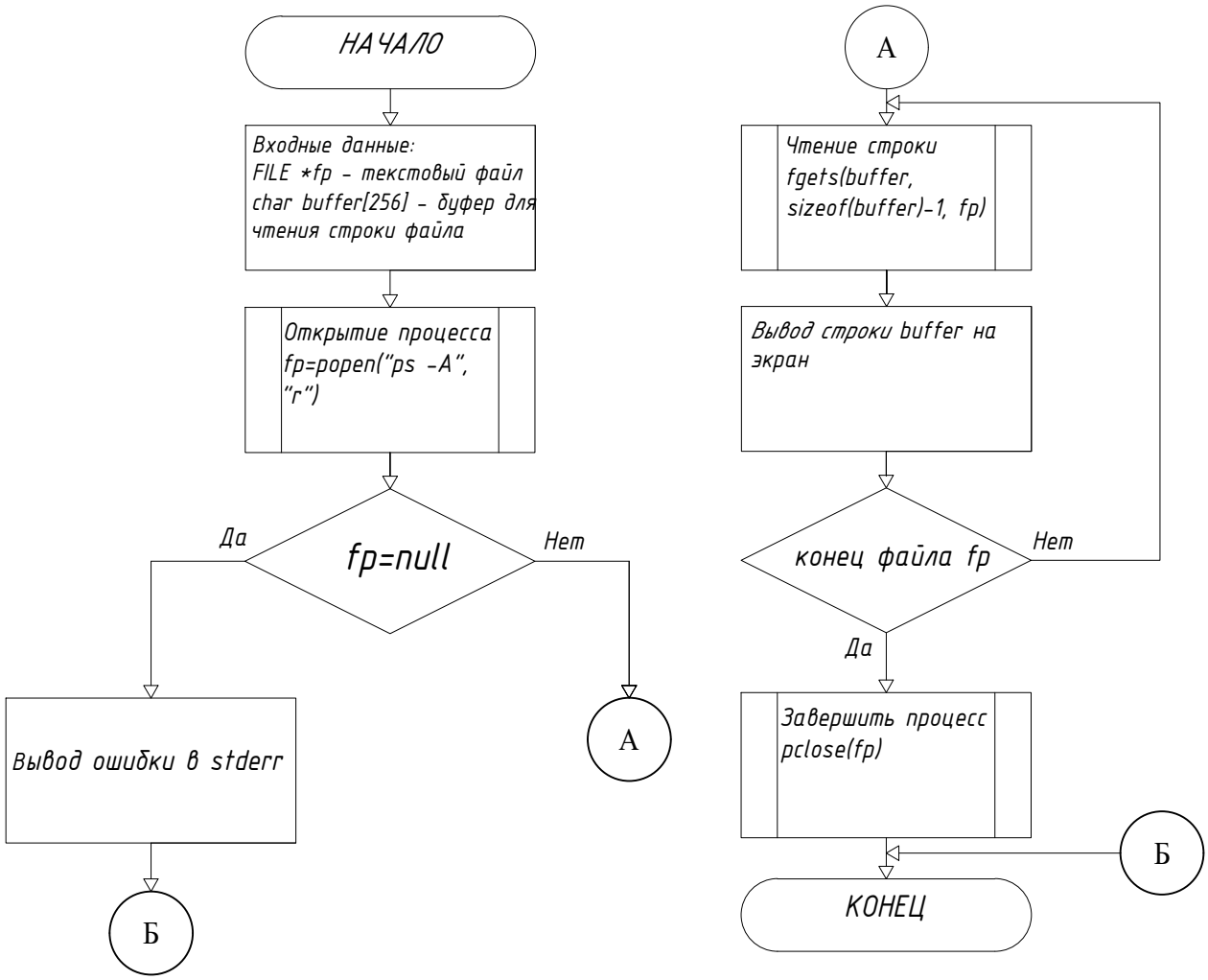
Копировал

Формат А4

ПРИЛОЖЕНИЕ Б

(обязательно)

Блок-схема функции `proc_info()`



ГУИР.400201.104 ПД.2

Изм	Лист	N° докум.	Подп.	Дата
Разраб.		Гаращук		
Пров.		Поденок		
Т.контр.				
Н.контр.				
Утв.				

Схема алгоритма proc_info() вывод информации о процессах

Лит.	Масса	Масштаб

Лист	Листов

ЭВМ, гр. 150501

Копировал

Формат А4

ПРИЛОЖЕНИЕ В

(обязательно)

Листинг кода

```
//util.h
#define _GNU_SOURCE
#include <gtk/gtk.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <locale.h>
#include <string.h>
#include <sys/utsname.h>
#include <sys/sysinfo.h>

struct modul{           //пакет ядра
   char name[256];      //название пакета
   char size[256];      //размер пакет
   char used_by[256];   //пакеты которые используются данный
};

struct cpu_info{            //инфморация ядра
    char cpu_product[128];      //продукт
    char cpu_vendor[128];       //производитель
    char cpu_family[128];       //семья процессора
    char cpu_cores[128];        //количество ядер
    char cpu_microcode[128];    //микрокод процуессора
    char cpu_cache[128];        //кеш
    char cpu_bogomips[128];     //скорость исполнения
    char cpu_addres_size[128];  //размеры адресов
};

void get_proc_info();   //информация о процессах
void cpu_temp();        //температура процессора
void get_system_info(); //информация о системе
void get_cpu_info();    //информация о процессоре
void get_modules_info();//установленные пакеты я ядре
void kernel_info();     //информация о ядре
void get_usb_info();    //информация о подключенныхх устройствах
void get_loop_info();   //информация о дисках
void get_information(); //информация об утилите

//основное оконное приложение
void activate(GtkApplication *app, gpointer user_data);
```

```
//util.c
#include "util.h"

int main(int argc, char **argv)
{
    GtkApplication *app;
    int status;

    // инициализация приложения
    app = gtk_application_new(NULL, G_APPLICATION_FLAGS_NONE);
    g_signal_connect(app, "activate", G_CALLBACK(activate), NULL);

    // Запуск приложениz
    status = g_application_run(G_APPLICATION(app), argc, argv);
    g_object_unref(app);

    return status;
}

//получить информацию о процессоре
int fill_cpu(struct cpu_info *cpu){
    char buffer[1024];
    FILE *fp = fopen("/proc/cpuinfo", "r");
    if (fp == NULL) {
        perror("Не удалось открыть файл /proc/cpuinfo\n");
        return 1;
    }

    while (fgets(buffer, sizeof(buffer), fp)) {     //модель
        if (strncmp(buffer, "model name", 10) == 0) {
            strcpy(cpu->cpu_product, buffer);
        }

        //производитель
        else if (strncmp(buffer, "vendor_id", 9) == 0) {
            strcpy(cpu->cpu_vendor, buffer);
        }

        //количествово ядер
        else if (strncmp(buffer, "cpu cores", 9) == 0) {
            strcpy(cpu->cpu_cores, buffer);
        }

        //семья процессора
        else if (strncmp(buffer, "cpu family", 9) == 0) {
            strcpy(cpu->cpu_family, buffer);
        }
        else if (strncmp(buffer, "microcode", 9) == 0) {    //микрокод
            strcpy(cpu->cpu_microcode, buffer);
        }

        //размер кеша
        else if (strncmp(buffer, "cache size", 10) == 0) {
            strcpy(cpu->cpu_cache, buffer);
        }
```

```
        //скорость исполнения
        else if (strncmp(buffer, "bogomips", 8) == 0) {
            strcpy(cpu->cpu_bogomips, buffer);
        }

        //размер адресов
        else if (strncmp(buffer, "address sizes", 13) == 0) {
            strcpy(cpu->cpu_addres_size, buffer);
        }
        else if(strncmp(buffer, "\n", 1) == 0)
            break;
    }

    fclose(fp);
    return 0;
}

void get_proc_info()    //информация о процессах
{
    printf("\n\n");
    FILE *fp;
    char path[1035];

    //запуск процесса, который выводит список процессов
    fp = popen("ps -A", "r");
    if (fp == NULL) {
        perror("Ошибка команды \"ps -A\"\n" );
        return;
    }
    printf("PID\tCOMMAND\n");
    while (fgets(path, sizeof(path)-1, fp) != NULL) {
        printf("%s", path);     //вывод процессов
    }
    pclose(fp);
}

void cpu_temp(){    //температура процессора
    //открытие файла с тепературой
    FILE* temp_file = fopen("/sys/class/thermal/thermal_zone0/temp", "r");

    if (temp_file == NULL) {
        perror("Ошибка файла \"/sys/class/thermal/thermal_zone0/temp\"\n");
    return;
    }

    int temp_raw;
    fscanf(temp_file, "%d", &temp_raw);
    fclose(temp_file);

    float temp_c = (float) temp_raw / 1000;    //перевод в цельсии

    printf("CPU temperature: %.1f C\n", temp_c);    //вывод результата
}
void get_system_info()      //информация о системе
{
```

```
    printf("\n\n");
    //Получаем информацию о системе
    struct utsname uname_data;
    if (uname(&uname_data) == -1) {
        perror("uname");
        return;
    }
    struct sysinfo sysinfo_data;
    if (sysinfo(&sysinfo_data) == -1) {
        perror("sysinfo");
        return;
    }

    char buffer[1024];
    FILE* fp;
    fp = popen("cat /proc/cpuinfo | grep 'model name' | uniq", "r");
    fgets(buffer, sizeof(buffer), fp);
    pclose(fp);

    // Выводим информацию на экран
    printf("Operating system: %s %s\n", uname_data.sysname, uname_data.release);
    printf("CPU %s", buffer);
    cpu_temp();
    printf("Machine type: %s\n", uname_data.machine);
    printf("User ID: %d\n", getuid());
    printf("Group ID: %d\n", getgid());
    printf("Effective user ID: %d\n", geteuid());
    printf("Effective group ID: %d\n", getegid());
    printf("Number of processors: %ld\n", sysconf(_SC_NPROCESSORS_ONLN));
    printf("Total RAM: %ld MB\n", sysinfo_data.totalram / 1024 / 1024);
    printf("Free RAM: %ld MB\n", sysinfo_data.freeram / 1024 / 1024);
}

void get_cpu_info(){        //информация о процессоре
    printf("\n\n");
    struct cpu_info cpu;
    if(fill_cpu(&cpu))
        return;

    printf("%s", cpu.cpu_product);
    printf("%s", cpu.cpu_vendor);
    printf("%s", cpu.cpu_family);
    printf("%s", cpu.cpu_cores);
    printf("%s", cpu.cpu_microcode);
    printf("%s", cpu.cpu_cache);
    printf("%s", cpu.cpu_bogomips);
    printf("%s", cpu.cpu_addres_size);
}

void get_modules_info(){        //информация об установленных пакетах
    FILE *fp;
    char line[256];
    struct modul ker_modul;
    // Открываем файл со списком загруженных модулей
    fp = fopen("/proc/modules", "r");
```

```
    if (fp == NULL) {
        perror("Не удалось открыть /proc/modules");
        return;
    }

    printf("Загруженные модули:\n");
    // Считываем построчно файл и выводим информацию о модулях
    while (fgets(line, sizeof(line), fp)) {
         //Разбиваем строку на токены с помощью функции sscanf
        sscanf(line, "%s %s %*s %s", ker_modul.name, ker_modul.size,
               ker_modul.used_by);
        // Выводим отформатированную информацию о модуле
       printf("%-30s %10s %10s\n", ker_modul.name, ker_modul.size,
               ker_modul.used_by);
    }
    fclose(fp);
}

void kernel_info(){     //информация о ядре
    printf("\n\n");
    struct utsname buffer;
    if (uname(&buffer) != 0) {      //заполняем структуру
        perror("Не удалось получить информацию\n");
        return;
    }
    //вывод информации
    printf("Operating System: %s\n", buffer.sysname);
    printf("Node Name: %s\n", buffer.nodename);
    printf("Kernel Release: %s\n", buffer.release);
    printf("Kernel Version: %s\n", buffer.version);
    printf("Machine Hardware Name: %s\n", buffer.machine);
}

void get_usb_info(){        //информация о подключенных устройствах
    printf("\n\n");
    system("lsusb");        //вызов системной утилиты lsusb

}

void get_loop_info(){       //информация о жестких дисках
    printf("\n\n");
    system("lsblk");        //вызов системной утилиты lsblk
}

void get_information(){     //вывод информации об утилите
    printf("\n\nРазработчик: Гаращук Никита Васильевич\nСтудент группы
            150501.\n");
}

// Функция для создания главного окна приложения
void activate(GtkApplication *app, gpointer user_data)
{
    // Объявление переменных для виджетов окна
    GtkWidget *window;
    GtkWidget *button_proc;
```

```
GtkWidget *button_cpu;
GtkWidget *button_system;
GtkWidget *button_usb;
GtkWidget *button_loop;
GtkWidget *button_info;
GtkWidget *modules_info;
GtkWidget *button_kernel;
GtkWidget *vbox;

// Создание главного окна и задание его параметров
window = gtk_application_window_new(app);
gtk_window_set_title(GTK_WINDOW(window), "Nigava (Сбор информации)");
gtk_container_set_border_width(GTK_CONTAINER(window), 10);

// Создание вертикального контейнера для размещения виджетов
vbox = gtk_box_new(TRUE, 8);
gtk_container_add(GTK_CONTAINER(window), vbox);

// Создание кнопок и подключение к ним функций обработки событий
button_proc = gtk_button_new_with_label("Информация о работающих процессах");
button_system = gtk_button_new_with_label("Общая информация о системе");
button_cpu = gtk_button_new_with_label("Информация о процессоре");
button_usb = gtk_button_new_with_label("Информация о подключенных устройствах");
button_loop = gtk_button_new_with_label("Информация о жестких дисках");
button_info = gtk_button_new_with_label("Информация об утилите");
button_kernel = gtk_button_new_with_label("Информация об ядре");
modules_info = gtk_button_new_with_label("Информация о модулях установленных в
                                          ядро");

g_signal_connect(button_proc, "clicked", G_CALLBACK(get_proc_info),
                 "proc_button");

g_signal_connect(button_cpu, "clicked", G_CALLBACK(get_cpu_info),
                 "cpu_button");

g_signal_connect(button_system, "clicked", G_CALLBACK(get_system_info),
                 "system_button");

g_signal_connect(button_usb, "clicked", G_CALLBACK(get_usb_info),
                 "usb_button");

g_signal_connect(button_loop, "clicked", G_CALLBACK(get_loop_info),
                 "loop_button");

g_signal_connect(button_info, "clicked", G_CALLBACK(get_information),
                 "info_button");
g_signal_connect(button_kernel, "clicked", G_CALLBACK(kernel_info),
                 "kernel_button");

g_signal_connect(modules_info, "clicked", G_CALLBACK(get_modules_info),
                 "modules_button");

// Размещение кнопок в контейнере
gtk_box_pack_start(GTK_BOX(vbox), button_proc, TRUE, TRUE, 5);
gtk_box_pack_start(GTK_BOX(vbox), button_cpu, TRUE, TRUE, 5);
```

```
    gtk_box_pack_start(GTK_BOX(vbox), button_system, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), button_usb, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), button_loop, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), button_kernel, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), modules_info, TRUE, TRUE, 5);
    gtk_box_pack_start(GTK_BOX(vbox), button_info, TRUE, TRUE, 5);

    // Отображение всех виджетов на экране
    gtk_widget_show_all(window);
}
```

ПРИЛОЖЕНИЕ Г

(обязательно)

Ведомость документов

Обозначение	Наименование	Примечание
	Графические документы	
ГУИР.400201.104 ПД.1	Схема алгоритма kernel_info()	А4
	Блок-схема.	
ГУИР.400201.104 ПД.2	Схема алгоритма proc_info()	А4
	Блок-схема.	
	Текстовые документы	
ГУИР КП 1-40 02 01 104 ПЗ	Утилита сбора информации о системе	21 с.
	Пояснительная записка.	
	Листинг программы	7 с.
ГУИР.400201.104-01 12	Программа	Диск

Изм	Лист	№ документа	Подпись	Дата	ГУИР.400201.104 Д1		
Разраб.		Гаращук			Утилита сбора информации о системе	Лит.	Лист	Листов
Пров.		Поденок				У		1
						ЭВМ, гр. 150501		