Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина «Конструирование программ»

**ОТЧЕТ**

к лабораторной работе №7

на тему:

**«Интеграция ассемблерных прерываний в проекты на C++.»**

БГУИР 6-05-0612-02 85

|  |
| --- |
| Выполнил студент группы 353503  МАРТИНОВИЧ Андрей Александрович |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 1 Индивидуальное задание

**Задание 1. Вариант 10.** На стороне Assembler:

Показать все текущие выполняющиеся процессы с их идентификаторами, именами, статусами и потреблением ресурсов. Предоставлять возможность приостанавливать, возобновлять и завершать процессы. При выборе определенного процесса, пользователь должен увидеть детали о нем, такие как время запуска, путь к файлу, используемые ресурсы. Позволять фильтровать процессы по имени, ресурсам, а также сортировать их по различным критериям(1 любой критерий).

На стороне C++: Логика управления. Вызов ассемблерных функций. Меню и интерфейс. Используемые прерывания: int 10h, int 16h, int 21h.

# 2 Выполнение работы

Программа реализована с использованием ассемблера nasm, языка С++ для ОС Linux.

Листинг кода 1 – Внешние функции в С++

extern "C" const char \*get\_starttime(const char \*pid);

extern "C" const char \*get\_procname(const char \*pid);

extern "C" const char \*get\_procstate(const char \*pid);

extern "C" const char \*get\_cmdline(const char \*pid);

extern "C" const char \*get\_info(const char \*pid);

extern "C" void suspend\_proc(int pid);

extern "C" void resume\_proc(int pid);

extern "C" void complete\_proc(int pid);

extern "C" const char \*get\_btime();

Листинг кода 2 – Пример вызова assembler-функции из С++ кода

void print\_process\_info(const std::string& pid, const char\* btime) {

const char\* stime = get\_starttime(pid.c\_str());

std::string p\_time = get\_time\_format(stime, btime);

std::string p\_name = get\_procname(pid.c\_str());

std::string p\_state = get\_procstate(pid.c\_str());

std::string p\_path = get\_cmdline(pid.c\_str());

if (!p\_name.empty()) {

std::cout << std::left << std::setw(10) << pid << std::setw(15) << p\_time << std::setw(25)

<< p\_name << std::setw(7) << p\_state << std::setw(60) << p\_path << "\n";

}

}

Листинг кода 3 – Пример реализованной функции

%macro print\_string 1

push rax

push rdi

push rsi

push rdx

push rcx

mov rax, 1 ; системный вызов sys\_write

mov rdi, 1 ; файловый дескриптор 1 (stdout)

mov rsi, %1 ; указатель на строку

mov rdx, 1024

syscall

pop rcx

pop rdx

pop rsi

pop rdi

pop rax

%endmacro

section .data

prefix db '/proc/', 0

suffix db '/stat', 0

buffer times 1024 db 0 ; буфер для чтения содержимого файла

msg db 'a', 0Ah, 0Dh, 0

starttime times 20 db 0

section .bss

filename resb 256 ; буфер для хранения starttime

section .text

global get\_procstate

get\_procstate:

mov rbx, rdi

; Формирование строки пути к файлу

lea rsi, [rel filename] ; указатель на буфер для конечного пути

lea rdi, [rel prefix] ; указатель на префикс "/proc/"

call strcpy

mov rdi, rbx ; указатель на строку с номером (PID)

call strcpy

lea rdi, [rel suffix] ; указатель на суффикс "/stat"

call strcpy

; Открыть файл

mov rax, 2 ; системный вызов sys\_open

lea rdi, [rel filename] ; имя файла (с относительным адресом)

mov rsi, 0 ; режим чтения

syscall

mov rbx, rax ; сохранить файловый дескриптор

; Читать содержимое файла

mov rax, 0 ; системный вызов sys\_read

mov rdi, rbx ; файловый дескриптор

lea rsi, [rel buffer] ; буфер для чтения (с относительным адресом)

mov rdx, 1024 ; количество байт для чтения

syscall

lea rdi, [rel buffer] ; указатель на начало буфера (с относительным адресом)

mov ecx, 2 ; количество пробелов для поиска

next\_space:

cmp byte [rdi], 0 ; проверить конец строки

je end\_of\_string

cmp byte [rdi], ' ' ; найти пробел

je found\_space

inc rdi ; перейти к следующему символу

jmp next\_space

found\_space:

inc rdi

loop next\_space

; Копировать значение starttime в буфер

lea rsi, [rel starttime] ; буфер для хранения starttime (с относительным адресом)

mov rcx, 20 ; максимальная длина поля

copy\_field:

cmp byte [rdi], ' ' ; проверить конец поля

je end\_of\_copy

mov al, [rdi]

mov [rsi], al

inc rdi

inc rsi

loop copy\_field

end\_of\_copy:

mov byte [rsi], 0 ; добавить завершающий нулевой байт

; Закрыть файл

mov rax, 3 ; системный вызов sys\_close

mov rdi, rbx ; файловый дескриптор

syscall

lea rax, [rel starttime]

ret

end\_of\_string:

jmp end\_of\_copy

strcpy:

.copy\_next:

mov al, [rdi]

mov [rsi], al

test al, al

je .done

inc rsi

inc rdi

jmp .copy\_next

.done:

Ret

Объектные файлы assembler-функций подключаются в файле CMake.

Листинг кода 4 – Подключение объектных файлов

cmake\_minimum\_required(VERSION 3.27)

project(untitled)

set(CMAKE\_CXX\_STANDARD 17)

add\_executable(

untitled

main.cpp

get\_time.o

get\_name.o

get\_state.o

get\_btime.o

get\_cmdline.o

get\_info.o

suspend\_proc.o

resume\_proc.o

complete\_proc.o

)

# Вывод

В ходе выполнения лабораторной работы были получены навыки работы с системными вызовами ОС Linux с использованием языка assembler; интеграции кода на языках С++ и языке assembler.