|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**ФАКУЛЬТЕТ** «Информатика и системы управления»

**КАФЕДРА «**Теоретическая информатика и компьютерные технологии»

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**ПО КУРСУ:**

***«ОПЕРАЦИОННЫЕ СИСТЕМЫ»***

|  |  |
| --- | --- |
| Студент *ИУ9-41Б* | *Ионов Т. Р.* |
| Преподаватель | *Брагин А.В.* |

Москва, 2021 г.

**Содержание**

1.Постановка задачи.

2.Практическая реализация.

2.1 Reactos.

2.2 NetBSD.

3.Результаты.

4.Вывод.

5.Используемая литература.

**Постановка задачи**

Разработать загружаемый модуль ядра (драйвер), работающий с виртуальной памятью и выводящий отладочную информацию.

**Практическая реализация**

**2.1. ReactOS**

1. Изучение общей структуры загружаемого модуля ядра в системе ReactOS в официальной документации. Изучение библиотек, позволяющим получить доступ к виртуальной памяти
2. Модификация исходных файлов операционной системы, путём добавления в папку drivers файла с исходным кодом драйвера, а также служебных файлов, необходимых для компиляции
3. Пересборка iso-образа операционной системы и его установка на виртуальную машину Oracle VirtualBox.
4. Запуск полученного драйвера с помощью встроенной утилиты sc start.

**2.2. NetBSD**

1. Изучение общей структуры загружаемого модуля ядра в системе NetBSD в официальной документации. Изучение библиотек, позволяющим получить доступ к списку процессов в системе
2. Добавление исходного кода драйвера в папку /usr/src/sys/dev и makefile в папку /usr/src/sys/modules/lab4.
3. Сборка драйвера с помощью утилиты make и его установка с помощью встроенной утилиты modload.

**Исходный код**

1) Reactos

/\*

\* PROJECT: ReactOS lab4

\* LICENSE: GPL - See COPYING in the top level directory

\* FILE: drivers/lab4.c

\* PURPOSE: Processes Output Driver

\* PROGRAMMERS: Ionov Timur IU9-41B

\*/

/\* INCLUDES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <ntddk.h>

#ifndef NDEBUG

#define NDEBUG

#endif

#include <debug.h>

#include <ntifs.h>

#include <ndk/ntndk.h>

#include <windef.h>

/\* FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DRIVER\_UNLOAD DriverUnload;

VOID

NTAPI

DriverUnload(IN PDRIVER\_OBJECT DriverObject)

{

PDEVICE\_OBJECT DeviceObject;

/\* Get DO \*/

DeviceObject = DriverObject->DeviceObject;

/\* Delete the object \*/

DPRINT1("------------------DRIVER-UNLOADED--------------------\n");

IoDeleteDevice(DeviceObject);

}

NTSTATUS

NTAPI

Virtual()

{

DPRINT1("--------Ionov Timru lab4--------");

NTSTATUS Status;

PVOID Pages = **NULL**;

PHARDWARE\_PTE PTE\_BASE = (PHARDWARE\_PTE)0xc0000000;

PHARDWARE\_PTE pte;

SIZE\_T sizeReserve = 1024 \* 4 \* 10; // bytes for 32-bit version

SIZE\_T sizeCommit = 1024 \* 4 \* 5; // bytes for 32-bit version

**int** i;

/\* Reserving 10 pages \*/

Status = ZwAllocateVirtualMemory(NtCurrentProcess(), &Pages, 0, &sizeReserve, MEM\_RESERVE, PAGE\_READWRITE);

**if** (!NT\_SUCCESS(Status)){

DPRINT1("ERROR: Failed to reserve memory\n");

**return** Status;

}

DPRINT1("10 PAGES RESERVED");

/\* Committing first 5 pages \*/

Status = ZwAllocateVirtualMemory(NtCurrentProcess(), &Pages, 0 , &sizeCommit, MEM\_COMMIT, PAGE\_READWRITE);

**if**(!NT\_SUCCESS(Status)) {

DPRINT1("ERROR: Failed to commit memory\n");

**return** Status;

}

/\* Assigning values to memory allocated \*/

**for**(i = 0; i < 5; i++){

\*((PCHAR)Pages + 0x1000 \* i) = i + 1;

}

/\* Printing informating \*/

**for**(i = 0; i < 5; i++) {

pte = ((ULONG)Pages >> 12) + PTE\_BASE + i;

DPRINT1("Page №%d\n\

Physical address: %d\n\

Valid: %d\n\

WriteThrough: %d\n\

CacheDisable: %d\n\

Accessed: %d\n\

Dirty: %d\n\

LargePage: %d\n\

Global: %d\n\

CopyOnWrite: %d\n\

Prototype: %d\n\

PageFrameNumber: 0x%x\n\

\n", i + 1, pte->PageFrameNumber \* 0x1000, pte->Valid,

pte->WriteThrough, pte->CacheDisable, pte->Accessed,

pte->Dirty, pte->LargePage, pte->Global,

pte->CopyOnWrite, pte->Prototype, pte->PageFrameNumber);

}

/\* Releasing memory \*/

ZwFreeVirtualMemory(NtCurrentProcess(), &Pages, 0, MEM\_RELEASE);

DPRINT1("MEMROY IS RELEASED");

**return** Status;

}

NTSTATUS

NTAPI

DriverEntry(IN PDRIVER\_OBJECT DriverObject,

IN PUNICODE\_STRING RegistryPath)

{

// PDEVICE\_EXTENSION DeviceExtension;

PDEVICE\_OBJECT DeviceObject;

UNICODE\_STRING DeviceName = RTL\_CONSTANT\_STRING(L"\\Device\\Virtual");

NTSTATUS Status;

UNREFERENCED\_PARAMETER(RegistryPath);

/\* Create the device \*/

Status = IoCreateDevice(DriverObject,

0,

&DeviceName,

FILE\_DEVICE\_UNKNOWN,

0,

**FALSE**,

&DeviceObject);

**if** (!NT\_SUCCESS(Status)) **return** Status;

DriverObject->DriverUnload = DriverUnload;

/\* Page the entire driver \*/

MmPageEntireDriver(DriverEntry);

/\* Allocating virtual memory \*/

Virtual();

**return** STATUS\_SUCCESS;

}

/\* EOF \*/

2) NetBSD:

#include <sys/module.h>

#include <sys/cdefs.h>

#include <sys/param.h>

#include <sys/sysctl.h>

#include <uvm/uvm.h>

MODULE(MODULE\_CLASS\_MISC, virtual, NULL);

#define PAGESIZE 0x1000

extern paddr\_t avail\_end;

vaddr\_t va;

struct pglist plist;

static int virtual\_modcmd(modcmd\_t cmd, void\* arg) {

va = uvm\_km\_alloc(kernel\_map, 10 \* PAGESIZE, 0, UVM\_KMF\_VAONLY);

if(va == 0) {

printf("Can't allocate virtual memory ;( \n");

return 1;

}

int er = uvm\_pglistalloc(5 \* PAGESIZE, 0, avail\_end, 0, 0, &plist, 5, 0);

if(er)

printf("Can't create pages ;(, err %d\n", er);

} else {

printf("Success\n");

}

struct vm\_page \*page = TAILQ\_FIRST(&plist);

for(int i = 0; page; i++) {

pd\_entry\_t \*ppte;

ppte = L2\_BASE + pl2\_i(va + PAGESIZE \* i);

paddr\_t pa = VM\_PAGE\_TO\_PHYS(page);

printf("Page №%d\n", i + 1);

printf("Phys addr page = 0x%lx\n", pa);

printf("Valid: %d\n", ((\*ppte & PG\_V) ? 1 : 0));

printf("WriteThrough: %d\n", ((\*ppte & PG\_WT) ? 1 : 0));

printf("NonCachable: %d\n", ((\*ppte & PG\_N) ? 1 : 0));

printf("Used: %d\n", ((\*ppte & PG\_U) ? 1 : 0));

printf("Modified: %d\n", ((\*ppte & PG\_M) ? 1 : 0));

printf("\n");

page = TAILQ\_NEXT(page, pageq.queue);

}

uvm\_pglistfree(&plist);

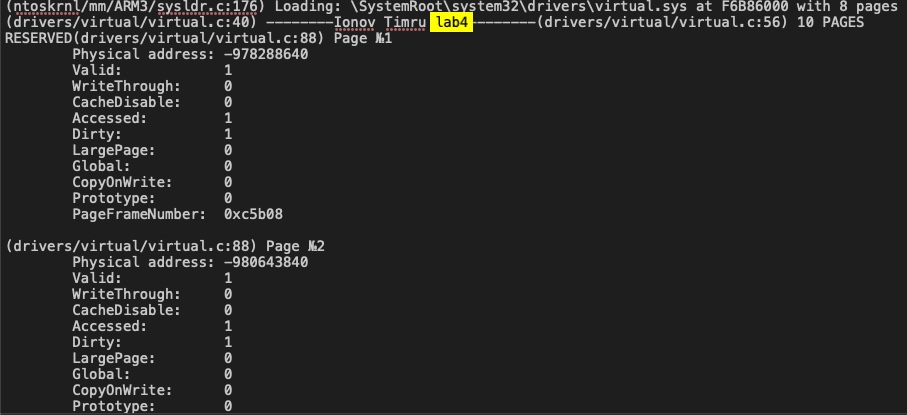
uvm\_km\_free(kernel\_map, va, 10 \* PAGESIZE, UVM\_KMF\_VAONLY);

return 0;

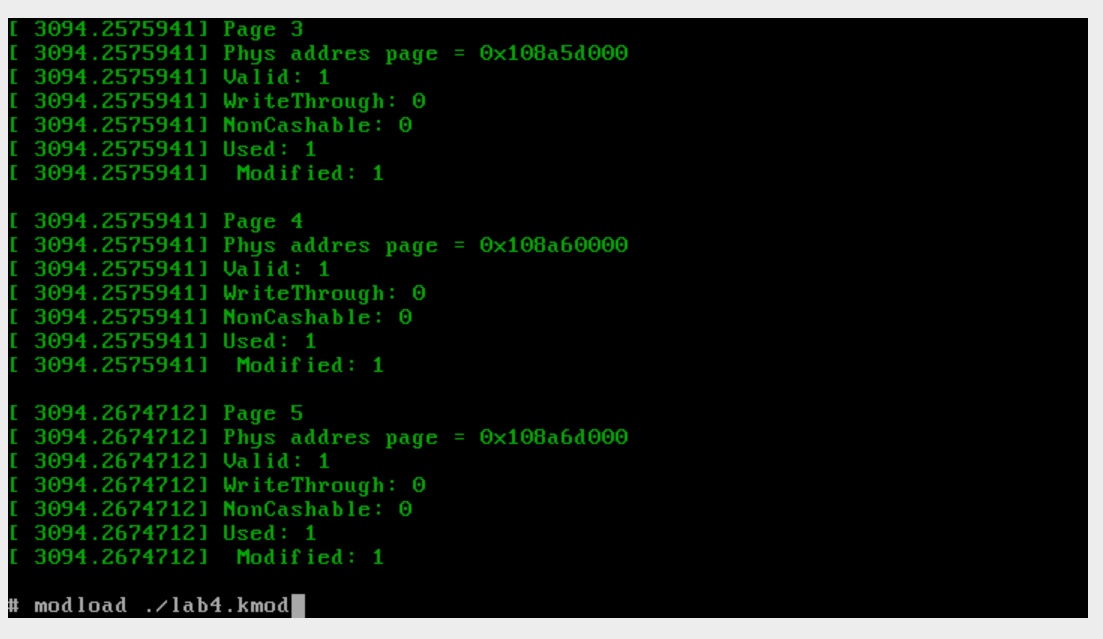
}

**Результат**

1) ReactOs:



2) NetBSD:

****

**Вывод**

В ходе выполнения были изучены библиотеки, позволяющие создавать страницы виртуальной памяти. Написание драйвера для ReactOS показалось задачей сложнее за счет более сложной архитектуры. Драйвер для NetBSD оказалось писать намного легче и не вызвало каких-либо трудностей. Хотя, работа с Vi доставила некоторые неудобства, так как я привык к более user-friendly IDE.

**Используемая литература**

1) Официальный вики операционной системы ReactOS - reactos.org/wiki.

2) Официальная документация к системе NetBSD - www.netbsd.org/docs.

3) Введение в сборку загружаемых модулей для NetBSD - http://www.home.unix-ag.org/bmeurer/NetBSD/howto-lkm.html.