**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»

РАБОТА С ВИРТУАЛЬНОЙ ПАМЯТЬЮ

Студент группы ИУ9-42Б Нащекин Н. Д.

Преподаватель: Брагин А. В.

*Москва, 2024*

# 1 Содержание

3 - Цель

4 - Постановка задачи

5 - Практическая реализация

11 - Результаты

13 - Выводы

14 - Список литературы

# 2 Цель

Разработать загружаемый по требованию модуль ядра (драйвер) для операционных систем ReactOS / Windows и NetBSD, работающий с виртуальной памятью.

# 3 Постановка задачи

ЧАСТЬ 1. РАБОТА С ПАМЯТЬЮ В РЕЖИМЕ ЯДРА В ОС СЕМЕЙСТВА WINDOWS NT

Используя созданный в лабораторной работе № 3 минимальный драйвер, совместимый с операционными системами Windows NT / ReactOS, реализовать следующие действия с виртуальной памятью:
1. Зарезервировать 10 страниц виртуальной памяти, используя ZwAllocateVirtualMemory(NtCurrentProcess(), MEM_RESERVE, . . . )
2. Обеспечить 5 первых страниц из выделенных 10-ти физическими страницами памяти, используя ZwAllocateVirtualMemory(NtCurrentProcess(), MEM_COMMIT, . . . ).
3. Вывести физические адреса и значения PTE для этих 5 страниц в шестнадцатеричном формате.
4. Освободить выделенную память.

ЧАСТЬ 2. РАБОТА С ПАМЯТЬЮ В РЕЖИМЕ ЯДРА В NETBSD

Используя созданный в лабораторной работе № 3 драйвер, реализовать следующие действия с виртуальной памятью:
1. Зарезервировать 10 страниц виртуальной памяти используя функцию ядра uvm_km_alloc().
2. страницами Обеспечить 5 первых страниц из выделенных 10-ти физическими памяти используя функции ядра uvm_pglistalloc() и pmap_kenter_pa().
3. Вывести физические адреса и значения PTE для этих 5 страниц в шестнадцатеричном формате.
4. Освободить выделенную память используя функции ядра pmap_kremove(), pmap_update(), uvm_pglistfree(), uvm_km_free().

# 4 Практическая реализация

ЧАСТЬ 1

Аналогично лабораторной работе 3, в скачанном ранее каталоге с исходным кодом ReactOS[1] в папке reactos/drivers я создал папку lab4_driver. В drivers/CMakeLists.txt добавил строку

```
add_subdirectory(lab4_driver)
```

для того, чтобы мой драйвер участвовал в сборке системы. В папке drivers/lab4_driver были созданы три файла: CMakeLists.txt, lab4_driver.c и lab4_driver.rc. Содержимое этих файлов было создано на основе моего драйвера из прошлых лабораторных. Содержимое CMakeLists.txt:

```
add_library(lab4_driver MODULE lab4_driver.c lab4_driver.rc)
set_module_type(lab4_driver kernelmodedriver)
add_importlibs(lab4_driver ntoskrnl)
add_cd_file(TARGET lab4_driver DESTINATION reactos/system32/drivers FOR all)
```

В главном файле с помощью ZwAllocateVirtualMemory[2] выделяется 10 страниц виртуальной памяти. Затем для 5 из 10 страниц с помощью этой же функции выделяются 5 страниц физической памяти. С помощью битовых операций в цикле для каждой страницы вычисляется значение PTE и выводятся некоторые его поля: Valid, Dirty, LargePage, Accessed, reserved и физический адрес. Также для проверки выводится физический адрес, вычисленный не с помощью PTE, а полученный с использованием MmGetPhysicalAddress. После вывода зарезервированная память освобождается.

lab4_driver.c:[3]

```
#include <ntddk.h>
#include <debug.h>
#include <ntifs.h>
#include <ndk/exfuncs.h>
#include <ndk/ketypes.h>
#include <ntstrsafe.h>

```

```
8  static DRIVER_UNLOAD DriverUnload;
9  static VOID NTAPI
10 DriverUnload(IN PDRIVER_OBJECT DriverObject)
11 {
12     IoDeleteDevice(DriverObject->DeviceObject);
13         DPRINT1("NASHCHEKIN NIKITA'S Driver unloaded\n");
14 }
15
16 NTSTATUS NTAPI DriverEntry(IN PDRIVER_OBJECT DriverObject,
17             IN PUNICODE_STRING RegistryPath) {
18     /* For non-used parameter */
19     UNREFERENCED_PARAMETER(RegistryPath);
20
21     /* Setup the Driver Object */
22     DriverObject->DriverUnload = DriverUnload;
23
24         ///Основная функция
25     DPRINT1("NASHCHEKIN NIKITA!!! From another driver\n\n");
26
27     PVOID base_adress = 0;
28     long unsigned int region_size = PAGE_SIZE * 10;
29
30     int status = ZwAllocateVirtualMemory(NtCurrentProcess(), &base_adress, 0, &region_size, MEM_RESERVE,
31
32     if (!NT_SUCCESS(status)) {
33         DPRINT1("Ошибка при выделении памяти");
34         goto end;
35     }
36
37     long unsigned int reserved_region_size = PAGE_SIZE * 5;
38
39     status = ZwAllocateVirtualMemory(NtCurrentProcess(), &base_adress, 0, &reserved_region_size, MEM_COMM
40
41     if (!NT_SUCCESS(status)) {
42         DPRINT1("Ошибка при резервировании памяти");
43         goto end;
44     }
45
46     for (int i = 0; i < 5; i++) {
47         DPRINT1("Страница %d:\n", i+1);
48
49         PVOID page_virtual_address = (PVOID)((ULONG_PTR)base_adress + i * PAGE_SIZE);
50
51
52         PMDL mdl = IoAllocateMdl(page_virtual_address, PAGE_SIZE, FALSE, FALSE, NULL);
53         if (mdl == NULL) {
54             DPRINT1("Ошибка при создании MDL\n");
55             goto end;
56         }
57
```

```
58          MmProbeAndLockPages(mdl, KernelMode, IoReadAccess);
59          PHYSICAL_ADDRESS physical_address = MmGetPhysicalAddress(page_virtual_address);
60          MmUnlockPages(mdl);
61          IoFreeMdl(mdl);
62
63        PHARDWARE_PTE_X86 pte = (PHARDWARE_PTE_X86)((ULONG)0xC0000000 + (((ULONG)page_vir
64
65
66          if (pte != NULL) {
67              DPRINT1("Виртуальный адрес: %#x\n", page_virtual_address);
68              DPRINT1("Физический адрес: %#x\n", physical_address);
69
70              DPRINT1("PTE (Valid): %s\n", (pte->Valid != 0 ? "Да" : "Нет"));
71              DPRINT1("PTE (Dirty): %s\n", (pte->Dirty != 0 ? "Да" : "Нет"));
72              DPRINT1("PTE (LargePage): %s\n", (pte->LargePage != 0 ? "Да" : "Нет"));
73              DPRINT1("PTE (Accessed): %s\n", (pte->Accessed != 0 ? "Да" : "Нет"));
74              DPRINT1("PTE (Reserved): %s\n", (pte->reserved != 0 ? "Да" : "Нет"));
75              DPRINT1("Физический адрес из PTE (для проверки): %#x\n\n", (pte->PageFrameNumber)<<12);
76          } else {
77              DPRINT1("PTE не получен\n\n");
78          }
79      }
80
81      for (int i = 5; i < 10; i++) {
82          DPRINT1("Страница %d:\n", i+1);
83
84          PVOID page_virtual_address = (PVOID)((ULONG_PTR)base_adress + i * PAGE_SIZE);
85
86          PHARDWARE_PTE_X86 pte = (PHARDWARE_PTE_X86)((ULONG)0xC0000000 + (((ULONG)page_vi
87
88          DPRINT1("Виртуальный адрес: %#x\n", page_virtual_address);
89          DPRINT1("Физический адрес: %#x\n", 000000);
90          DPRINT1("PTE (Valid): %s\n", (pte->Valid != 0 ? "Да" : "Нет"));
91          DPRINT1("PTE (Dirty): %s\n", (pte->Dirty != 0 ? "Да" : "Нет"));
92          DPRINT1("PTE (LargePage): %s\n", (pte->LargePage != 0 ? "Да" : "Нет"));
93          DPRINT1("PTE (Accessed): %s\n", (pte->Accessed != 0 ? "Да" : "Нет"));
94          DPRINT1("PTE (Reserved): %s\n", (pte->reserved != 0 ? "Да" : "Нет"));
95          DPRINT1("Физический адрес из PTE (для проверки): %#x\n\n", (pte->PageFrameNumber)<<12);
96      }
97
98      status = ZwFreeVirtualMemory(NtCurrentProcess(), &base_adress, &reserved_region_size, MEM_DECOMMIT
99
100     if (!NT_SUCCESS(status)) {
101         DPRINT1("Ошибка при освобождении физической памяти\n");
102         goto end;
103     }
104
105     status = ZwFreeVirtualMemory(NtCurrentProcess(), &base_adress, &region_size, MEM_RELEASE);
106
107     if (!NT_SUCCESS(status)) {
```

```
108         DPRINT1("Ошибка при освобождении виртуальной памяти\n");
109         goto end;
110     }
111 
112     DPRINT1("Память освобождена!\n\n");
113 
114     DPRINT1("%d\n", base_adress);
115     DPRINT1("%d\n", region_size);
116     DPRINT1("%d\n", reserved_region_size);
117     DPRINT1("!!!");
118 
119     end:
120     /* Return success. */
121     return STATUS_SUCCESS;
122 }
123 
```

lab4_driver.rc:

```
#define REACTOS_VERSION_DLL
#define REACTOS_STR_FILE_DESCRIPTION  "my another driver"
#define REACTOS_STR_INTERNAL_NAME     "lab4_driver"
#define REACTOS_STR_ORIGINAL_FILENAME "lab4_driver.sys"
#include <reactos/version.rc>
```

Далее в среде сборки был пересобран образ, а затем переустановлена система. В работающей системе были выполнены команды:

```
sc create lab4driver binPath=C:\ReactOS\system32\drivers\lab4_driver.sys type= kernel
sc start lab4driver
```

В логи выводится информация о всех 10 страницах виртуальной памяти.

ЧАСТЬ 2

Как и в предыдущей лабораторной, сначала был создан файл /usr/src/sys/dev/lab4.c и добавлена реализация драйвера:

```
#include <sys/module.h>
#include <sys/kernel.h>
#include <sys/proc.h>
#include <sys/types.h>
#include <sys/systm.h>
```

```
6  #include <uvm/uvm.h>
7  #include <sys/param.h>
8
9  MODULE(MODULE_CLASS_MISC, lab4, NULL);
10
11 static int lab4_modcmd(modcmd_t cmd, void* arg){
12         printf("NASHCHEKIN NIKITA Frrom another NetBSD driver!!!\n\n");
13
14         vaddr_t base_addres = uvm_km_alloc(kernel_map, 10 * 4096, 0,
15                                 UVM_KMF_VAONLY | UVM_KMF_WAITVA);
16         paddr_t physical_page_addres;
17         struct pglist plist;
18
19         if (uvm_pglistalloc(5 * 4096, 0, -1, 0, -1, &plist, 5, 0) != 0) {
20                 printf("Allocation of phys page failed\n");
21                 return 1;
22         }
23
24         struct vm_page* pg = TAILQ_FIRST(&plist);
25
26         for (int i = 0; i < 5; i++) {
27
28                 physical_page_addres = VM_PAGE_TO_PHYS(pg);
29
30                 pmap_kenter_pa(base_addres + i*4096, physical_page_addres,
31                         VM_PROT_READ | VM_PROT_WRITE, 0);
32
33                 pg = TAILQ_NEXT(pg, pageq.queue);
34                 printf("PHYS ADDR: %#lx\n", physical_page_addres);
35
36         }
37
38
39         for (int i = 0; i < 10; i++) {
40
41                 printf("\nPage %d:\n", i+1);
42                 printf("Virtual addres: %#lx\n", base_addres + i*4096);
43
44                 pt_entry_t* pte;
45                 pte = vtopte(base_addres + i*4096);
46
47                 printf("Valid: %s\n", ((*pte)&PG_V) ? "true" : "false");
48                 printf("Used: %s\n", ((*pte)&PG_U) ? "true" : "false");
49                 printf("Modified: %s\n", ((*pte)&PG_M) ? "true" : "false");
50                 printf("Physical addres: %#lx\n", (*pte)&PG_FRAME);
51         }
52
53         pmap_kremove(base_addres, 10 * 4096);
54         pmap_update(pmap_kernel());
55         uvm_pglistfree(&plist);
```

```
56          uvm_km_free(kernel_map, base_addres, 10 * 4096,
57                                  UVM_KMF_VAONLY | UVM_KMF_WAITVA);
58          return 0;
59  }
```

Драйвер резервирует 10 страниц вирутальной памяти с помощью uvm_km_alloc()[4], выделяет 5 страниц в физической памяти с помощью uvm_pglistalloc()[5] и привязывает к 5 первым страницам виртуальной памяти по одной соответствующей странице физической памяти с помощью функции pmap_kenter_pa()[6]. Затем вычисляется PTE с помощью vtopte()[7] и выводятся его поля. После вывода зарезервированная память освобождается[8].
Далее был создан файл /usr/src/sys/modules/lab4/Makefile со следующим содержимым:

```
1   .include "../Makefile.inc"
2   KMOD=lab4
3   .PATH: ${S}/dev
4   SRCS=lab4.c
5   .include <bsd.kmodule.mk>
```

Затем была выполнена команда make и modload ./lab4.kmod
В логи была выведена информациях обо всех страницах.

# 5 Результаты

Были реализованы драйверы на операционных системах ReactOS и NetBSD, резервирующие 10 страниц в виртуальной и 5 в физической памяти. Пять первых страниц виртуальной памяти обеспечиваются пятью страницами в физической, а остальные остаются "как есть". Затем для всех страниц в виртуальной памяти вычисляется PTE (Page Table Entry) и выводятся некоторые его поля, включая физический адрес.

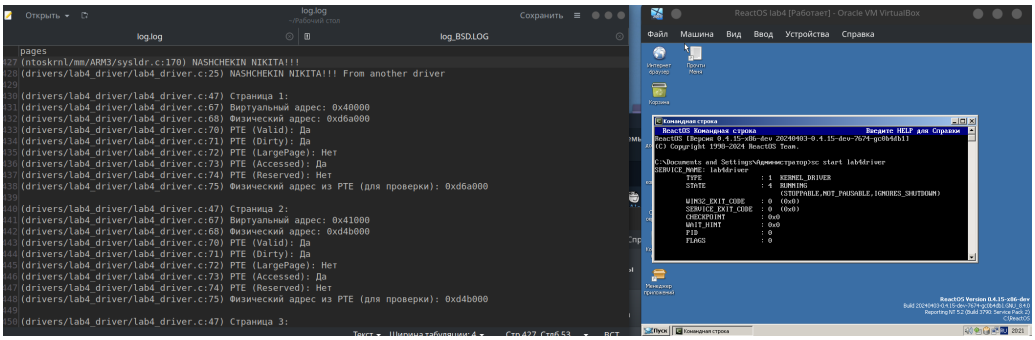


Рис. 1 — Скриншот 1

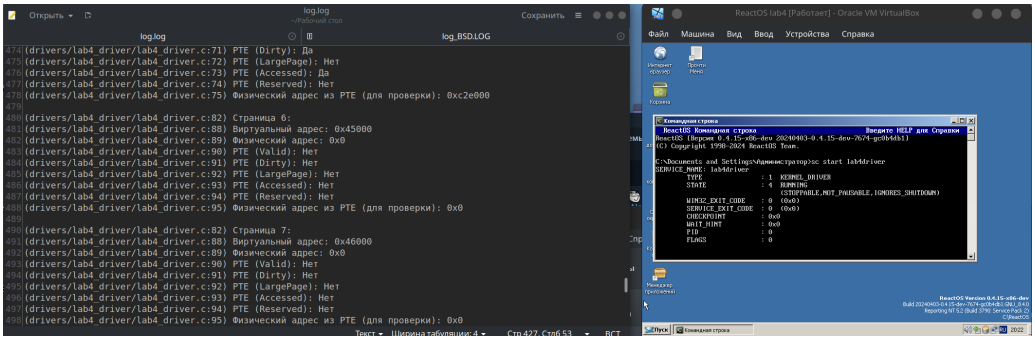


Рис. 2 — Скриншот 2

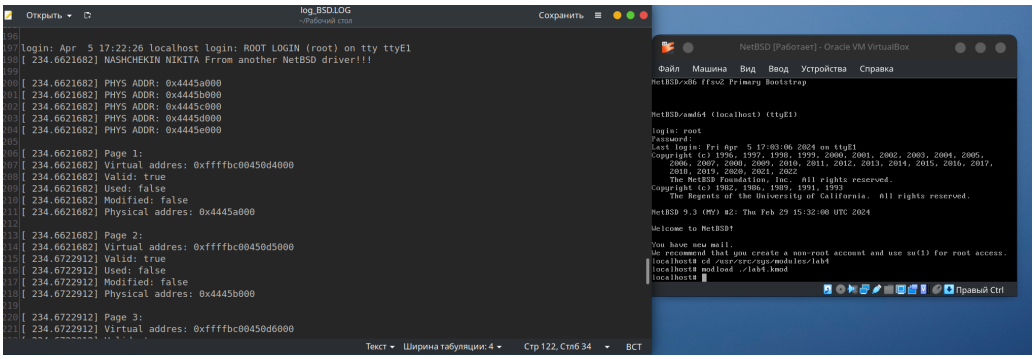


Рис. 3 — Скриншот 3

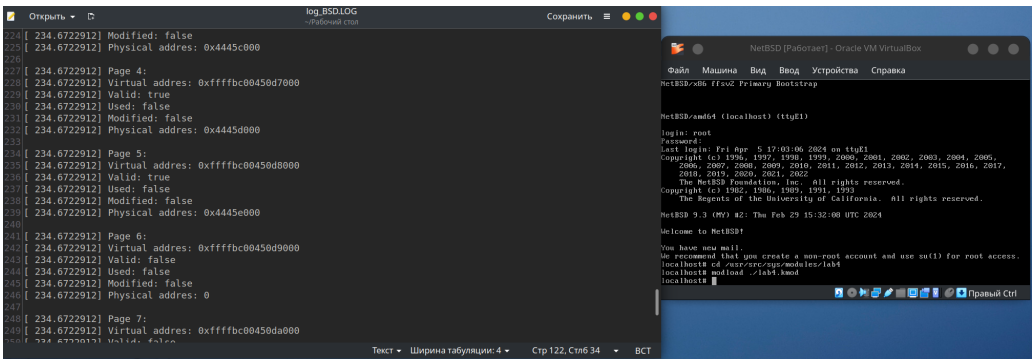


Рис. 4 — Скриншот 4

# 6 Выводы

Выполнив данную лабораторную работу, я разработал два драйвера для ReactOS и NetBSD. Выполнение работы на ReactOS оказалось проще, чем на NetBSD, поскольку здесь функция ZwAllocateVirtualMemory() отвечает за выделение виртуальной памяти, выделение физической памяти и ее привязку к виртуальным страницам, в то время как на NetBSD за это отвечают сразу три разных функции. Я научился азам работы с виртуальной памятью, её выделению, привязке к физической памяти и её освобождению.

# 7 Список литературы

[1] - https://reactos.org/wiki/Building_ReactOS
[2] - https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-zwallocatevirtualmemory
[3] - https://cpp.hotexamples.com/examples/-/-/ZwAllocateVirtualMemory/cpp-zwallocatevirtualmemory-function-examples.html
[4] - https://cpp.hotexamples.com/examples/-/-/uvm_km_alloc/cpp-uvm_km_alloc-function-examples.html
[5] - https://cpp.hotexamples.com/examples/-/-/uvm_pglistalloc/cpp-uvm_pglistalloc-function-examples.html
[6] - https://man.netbsd.org/pmap.9
[7] - https://netbsd.org/docs/kernel/porting_netbsd_arm_soc.html
[8] - https://edgebsd.org/index.php/manual/9/uvm_pglistfree