Министерство цифрового развития, связи и
массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

**Отчёт**
по лабораторной работе №2
по дисциплине «**Операционные системы**»

Выполнил:
студент гр. ИС-142
«__» декабря 2023 г. _____ /Наумов А.А./

Проверил:
ассистент
«__» декабря 2023 г. _____ /Третьяков Г.Н./

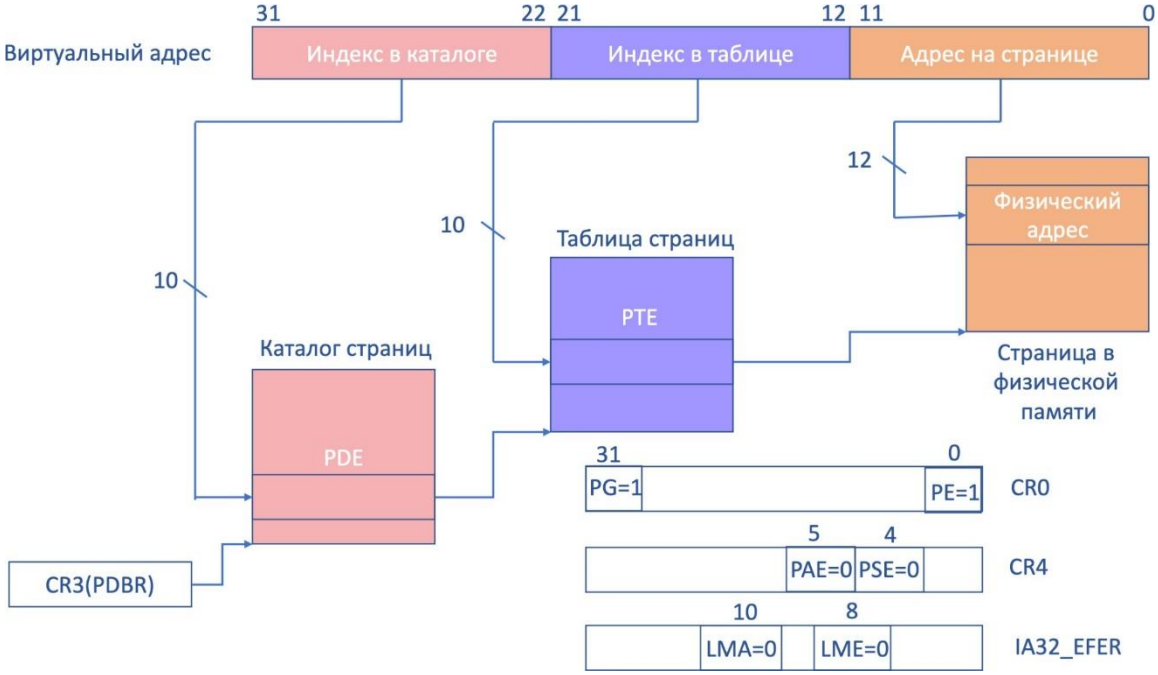
Оценка « _____»

Новосибирск 2023

## ВЫПОЛНЕНИЕ РАБОТЫ

Целью работы является создание программы для демонстрации отображениявиртуальной памяти в архитектуре Intel x86_64.

Обычный режим трансляции адресов на диаграмме:




**Нужно:** программа на языке Си для вывода в терминал отображения виртуальнойпамяти для некоторого размера оперативной памяти (в байтах/килобайтах/мегабайтах).

Код представляет собой пример реализации простой системы управления памятью (MMU) в операционной системе. Давайте разберём его функциональность:

**1. Структуры `page_directory_entry_t` и `page_table_entry_t`:**

Эти структуры представляют записи в каталоге страниц и таблице страниц соответственно. Они используются для управления виртуальной памятью в операционной системе. В каталоге страниц хранятся ссылки на таблицы страниц, а таблицы страниц содержат информацию о физических адресах памяти, соответствующих виртуальным адресам.

**2. Глобальные переменные `p_dir` и `p_tab`:**

`p_dir` — это переменная, представляющая каталог страниц, а `p_tab` — массив, представляющий таблицу страниц. Они выровнены по границе 4096 байт (4 КБ), что является обычным размером страницы памяти во многих системах.

**3. Функция `mmu_init()`:**

Эта функция инициализирует каталог страниц (`p_dir`) и таблицу страниц (`p_tab`). Она задает флаги для каждой записи в каталоге и таблице

страниц, а также устанавливает физические адреса для каждой страницы в таблице.

**4. Функции `mmu_get_dir()` и `mmu_get_tab()`:**

Эти функции предоставляют доступ к каталогу страниц и таблице страниц соответственно. Они возвращают указатели на соответствующие структуры.

**5. Функция `mmu_print()`:**

Эта функция выводит информацию о каталоге страниц и таблице страниц. Она печатает флаги для каталога страниц и адреса каждой записи в таблице страниц.

**6. Функция `main()`:**

В функции `main` вызываются `mmu_init()` для инициализации системы управления памятью и `mmu_print()` для вывода информации о ней.

В целом, этот код является учебным примером того, как работает система виртуальной памяти на низком уровне в операционных системах. Он демонстрирует использование каталога страниц и таблицы страниц для отображения виртуальных адресов на физические адреса. Это ключевая часть механизма виртуальной памяти, позволяющего операционной системе эффективно управлять доступной памятью.

Цель:

Разработать программу на языке C, которая моделирует базовую структуру и функциональность системы управления виртуальной памятью (Memory Management Unit - MMU) в операционной системе.

**Реализация и Работа Кода: Симулятор Управления Виртуальной Памятью (MMU)**

Обзор:

Программа представляет собой симулятор MMU (Memory Management Unit), который используется в операционных системах для управления виртуальной памятью. Основная задача симулятора - моделировать работу каталога страниц и таблицы страниц, используемых для преобразования виртуальных адресов в физические.

Структуры данных:

**1. Структура `page_directory_entry_t` (PDE):**

- Эта структура представляет собой запись в каталоге страниц. Она содержит флаги и адрес таблицы страниц.
- Флаги включают в себя присутствие страницы в памяти, доступ для

чтения/записи, уровень доступа (пользовательский/супервизор), и другие настройки.

**2. Структура `page_table_entry_t` (PTE):**

- Эта структура представляет запись в таблице страниц. Она содержит флаги и физический адрес страницы.
- Флаги аналогичны PDE и включают в себя присутствие страницы, доступ для чтения/записи, и другие параметры.

**Глобальные переменные:**

**1. `p_dir`:**

- Глобальная переменная, представляющая каталог страниц.

**2. `p_tab`:**

- Глобальный массив, представляющий таблицу страниц.

**Инициализация и Работа:**

**1. Функция `mmu_init()`:**

- Инициализирует `p_dir` и `p_tab` с использованием заданных значений и флагов.
- Задаёт физический адрес для каждой страницы в `p_tab`.

**2. Функция `mmu_print()`:**

- Выводит информацию о настройках и адресах в `p_dir` и `p_tab`.
- Отображает флаги каждой записи и адреса таблицы страниц.

**Вывод во время работы программы:**

**1. Информация о каталоге страниц (`p_dir`):**

- Показывает состояние флагов каталога страниц, например, присутствие страницы, доступ на чтение и запись.
- Выводит физический адрес каталога страниц.

**2. Информация о таблице страниц (`p_tab`):**

- Выводит флаги для каждой записи в таблице страниц.
- Показывает физический адрес каждой страницы в таблице.

**Ключевые моменты работы кода:**

**1. Моделирование преобразования адресов:**

- Программа моделирует, как виртуальные адреса могли бы быть отображены на физические адреса в операционной системе через механизмы каталога и таблицы страниц.

**2. Флаги управления доступом:**

- Флаги в PDE и PTE определяют, как операционная система

управляет доступом к памяти, включая права на чтение/запись и уровень доступа.

**3. Физические и виртуальные адреса:**

- Программа демонстрирует соответствие между виртуальными и физическими адресами, хотя не управляет реальной памятью.

**Заключение:**

Эта программа - учебный инструмент, предназначенный для демонстрации основных принципов работы системы управления виртуальной памятью. Она не взаимодействует с реальной памятью компьютера и не выполняет реальное отображение адресов, как это происходит в операционных системах.

Каждый флаг в структурах `page_directory_entry_t` (PDE) и `page_table_entry_t` (PTE) имеет своё назначение в контексте управления памятью. Вот подробное описание каждого флага в этих структурах:

**Флаги в `page_directory_entry_t` (PDE):**

**1. p (Present):**

- Определяет, присутствует ли страница в физической памяти.
- `1` означает, что страница находится в памяти; `0` - что страница не загружена или отсутствует.

**2. r_w (Read/Write):**

- Указывает, разрешена ли запись в страницу.
- `1` разрешает чтение и запись; `0` разрешает только чтение.

**3. u_s (User/Supervisor):**

- Определяет уровень доступа к странице.
- `1` разрешает доступ пользовательским процессам; `0` ограничивает доступ только супервизорскими процессами (ядром).

**4. w_t (Write-Through):**

- Указывает режим кэширования при записи.
- `1` для режима write-through; `0` для режима write-back.

**5. c_d (Cache-Disabled):**

- Управляет кэшированием этой страницы.
- `1` отключает кэширование; `0` разрешает кэширование.

**6. a (Accessed):**

- Флаг доступа, автоматически устанавливается процессором при доступе к странице.
- `1` указывает, что страница была использована; `0` - не использована.

**7. z (Zero):**

- Зарезервированное поле, обычно должно быть установлено в 0.

**8. p_s (Page Size):**

- Определяет размер страницы.

- `1` указывает на использование страниц большего размера (например, 4 МБ); `0` указывает на стандартный размер (обычно 4 КБ).

**9. i (Ignored):**

- Игнорируемое поле, не используется процессором.

**10. ava (Available):**

- Свободное для использования поле, может использоваться операционной системой для своих целей.

**11. p_table_addr (Page Table Address):**

- Содержит физический адрес таблицы страниц. Это 20-битное значение, которое сочетается с высшими битами физического адреса.

**Флаги в `page_table_entry_t` (PTE):**

**1. p (Present), r_w (Read/Write), u_s (User/Supervisor), w_t (Write-Through), c_d (Cache-Disabled), a (Accessed):**

- Аналогичны флагам в PDE.

**2. d (Dirty):**

- Указывает, была ли страница модифицирована (записана).

- `1` означает, что страница была изменена; `0` - не изменена.

**3. z (Zero), g (Global):**

- `z` - зарезервированное поле, должно быть установлено в 0.

- `g` (Global) указывает, что запись не должна быть удалена из TLB при смене таблицы страниц.

**4. ava (Available):**

- Свободное для использования поле, аналогичное PDE.

**5. p_phys_addr (Physical Address):**

- Содержит физический адрес страницы. Это 20-битное значение, которое сочетается с высшими битами физического адреса.

Эти флаги и поля используются для управления доступом к памяти, её кэшированием, и обеспечением защиты памяти в системах, использующих виртуальную память.

Вывод работы программы:
Выполнение программы выводит информацию о каталоге страниц и таблице страниц, которые используются в управлении виртуальной памятью. Разберем, что показывает каждая часть вывода:

**1. Информация о каталоге страниц:**

- `present: 1` — страница присутствует в памяти.

- `read_write: 1` — страница доступна для чтения и записи.

- `user_supervisor: 1` — страница доступна для пользовательского уровня доступа.

- `write_through: 1` — используется политика записи через кэш.

- `cache_disabled: 0` — кэширование разрешено.

- `accessed: 0` — страница пока не была использована.
- `zero: 1` — зарезервированное поле, всегда должно быть 0.
- `page_size: 0` — используются страницы стандартного размера (обычно 4 КБ).
- `ignored: 0` — игнорируемое поле.
- `available: 0` — доступное для использования пользователем поле.

**2. Адреса каталога и таблицы страниц:**

- `page_dir_phys_addr` — физический адрес каталога страниц в памяти.
- `page_table_addr` — физический адрес таблицы страниц в памяти.

**3. Информация о таблице страниц:**

Выводится информация для каждой записи в таблице страниц. Каждая строка представляет одну запись и содержит следующую информацию:

- Адрес записи в таблице страниц (`Table address`).
- Физический адрес соответствующей страницы памяти (`point to`). Этот адрес представлен в виде смещения (например, `0x0`, `0x1`, `0x2` и т.д.), который на самом деле представляет физический адрес страницы после сдвига на 12 бит влево (из-за формата хранения адреса в записи).

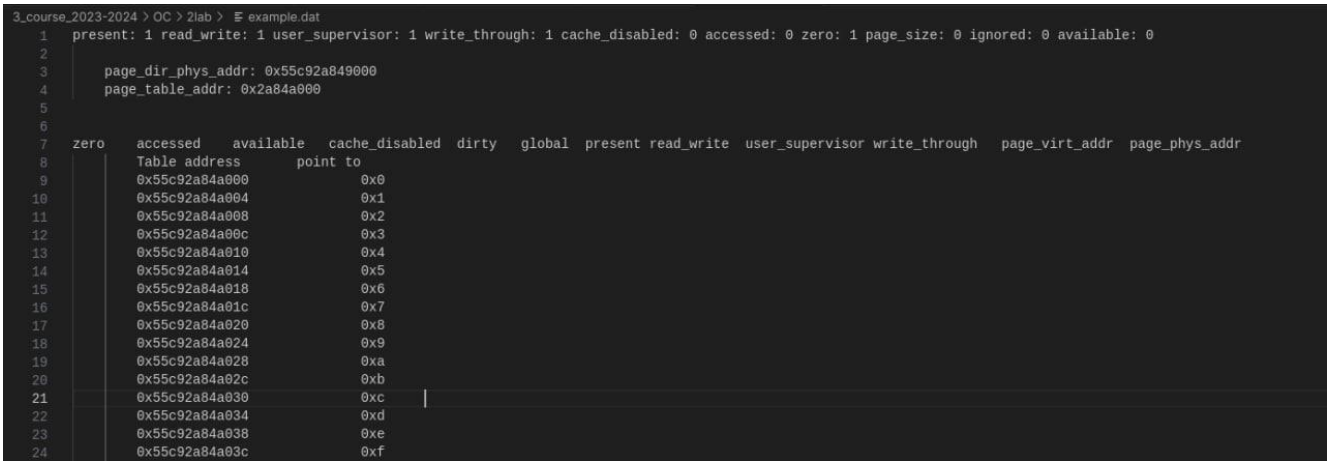
Этот вывод демонстрирует, как операционная система могла бы использовать структуры данных каталога страниц и таблицы страниц для управления виртуальной памятью. В реальных системах эти механизмы используются для отображения виртуальных адресов на физические, обеспечивая изоляцию и защиту памяти между разными процессами.

**Демонстрация работы программы:** При запуске программы, в ней указан размер памяти, если размер меньше 4096 байт, необходимо выделить целую страницу памяти (4096 Б = 4 КБ) и разметить её.

```
//Размеры

#define MMU_PAGE_TABLE_ENTRIES_COUNT 1024 //изменяем ее для наглядности

// PDE struct
    struct page_directory_entry_t {
unsigned char p : 1;
```



```
3_course_2023-2024 > OC > 2lab > ≡ example.dat
present: 1 read_write: 1 user_supervisor: 1 write_through: 1 cache_disabled: 0 accessed: 0 zero: 1 page_size: 0 ignored: 0 available: 0

    page_dir_phys_addr: 0x55c92a849000
    page_table_addr: 0x2a84a000


zero    accessed    available   cache_disabled  dirty   global  present read_write  user_supervisor write_through   page_virt_addr  page_phys_addr
        Table address       point to
        0x55c92a84a000              0x0
        0x55c92a84a004              0x1
        0x55c92a84a008              0x2
        0x55c92a84a00c              0x3
        0x55c92a84a010              0x4
        0x55c92a84a014              0x5
        0x55c92a84a018              0x6
        0x55c92a84a01c              0x7
        0x55c92a84a020              0x8
        0x55c92a84a024              0x9
        0x55c92a84a028              0xa
        0x55c92a84a02c              0xb
        0x55c92a84a030              0xc
        0x55c92a84a034              0xd
        0x55c92a84a038              0xe
        0x55c92a84a03c              0xf
```

Листинг программы:

```c
#include <assert.h>
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

#define attribute(value) __attribute__((value))

//Размеры

#define MMU_PAGE_TABLE_ENTRIES_COUNT 1024 //изменяем ее для наглядности

// PDE struct
    struct page_directory_entry_t {
  unsigned char p : 1;
  unsigned char r_w : 1;
  unsigned char u_s : 1;
  unsigned char w_t : 1;
  unsigned char c_d : 1;
  unsigned char a : 1;
  unsigned char z : 1;
  unsigned char p_s : 1;
  unsigned char i : 1;
  unsigned char ava : 3;
  unsigned int p_table_addr : 20;
} attribute(packed);

// PTE struct
struct page_table_entry_t {
  unsigned char p : 1;
  unsigned char r_w : 1;
  unsigned char u_s : 1;
  unsigned char w_t : 1;
  unsigned char c_d : 1;
  unsigned char a : 1;
```

```
  unsigned char d : 1;
  unsigned char z : 1;
  unsigned char g : 1;
  unsigned char ava : 3;
  unsigned int p_phys_addr : 20;
} attribute(packed);

//Структуры

static struct page_directory_entry_t p_dir attribute(aligned(4096));

static struct page_table_entry_t p_tab[MMU_PAGE_TABLE_ENTRIES_COUNT]
attribute(aligned(4096));


//Инициализация каталога страниц // 4mb
void mmu_init()
{
  //Инициализация каталога нулями
  memset(&p_dir, 0, sizeof(struct page_directory_entry_t));

  //Каталог(флаги)
  p_dir.z = 1;
  p_dir.a = 0;
  p_dir.ava = 0;
  p_dir.c_d = 0;
  p_dir.i = 0;
  p_dir.p_s = 0;
  p_dir.p = 1;
  p_dir.r_w = 1;
  p_dir.u_s = 1;
  p_dir.w_t = 1;
  //Адрес первой таблицы
  p_dir.p_table_addr = (uintptr_t)p_tab >> 12;

  //Таблица страниц(флаги)
  for (int i = 0; i < MMU_PAGE_TABLE_ENTRIES_COUNT; ++i) {
    //Таблица страниц(флаги)
    p_tab[i].z = 0;
    p_tab[i].a = 0;
    p_tab[i].ava = 0;
    p_tab[i].c_d = 0;
    p_tab[i].d = 0;
    p_tab[i].g = 1;
    p_tab[i].p = 1;
    p_tab[i].r_w = 1;
    p_tab[i].u_s = 1;
    p_tab[i].w_t = 1;
    //Физический адрес
    p_tab[i].p_phys_addr = (i * 4096) >> 12; //
  }
}
//Получение каталога страниц
```

```
extern struct page_directory_entry_t *mmu_get_dir() { return &p_dir;
}
//Получение таблицы страниц
extern struct page_table_entry_t *mmu_get_tab() { return p_tab; }
//Вывод на экран
void mmu_print() {
  printf("present: %d ", p_dir.p);
  printf("read_write: %d ", p_dir.r_w);
  printf("user_supervisor: %d ", p_dir.u_s);
  printf("write_through: %d ", p_dir.w_t);
  printf("cache_disabled: %d ", p_dir.c_d);
  printf("accessed: %d ", p_dir.a);
  printf("zero: %d ", p_dir.z);
  printf("page_size: %d ", p_dir.p_s);
  printf("ignored: %d ", p_dir.i);
  printf("available: %d\n\n", p_dir.ava);
  printf("\tpage_dir_phys_addr: %p\n", (void *)mmu_get_dir());
  printf("\tpage_table_addr: 0x%x000\n\n\n", p_dir.p_table_addr);

printf("zero\taccessed\tavailable\tcache_disabled\tdirty\tglobal\tpr
esent\tread_write\tuser_supervisor\twrite_through\tpage_virt_addr\tp
age_phys_addr\n");

//Таблица страниц
printf("\t\tTable address\t\tpoint to\n");
    for (int i = 0; i < MMU_PAGE_TABLE_ENTRIES_COUNT; ++i) {
        // printf("%1d\t", p_tab[i].z);
        // printf("%5d\t", p_tab[i].a);
        // printf("%9d\t", p_tab[i].ava);
        // printf("%9d\t", p_tab[i].c_d);
        // printf("%13d\t", p_tab[i].d);
        // printf("%5d\t", p_tab[i].g);
        // printf("%5d\t", p_tab[i].p);
        // printf("%5d\t", p_tab[i].r_w);
        // printf("%9d\t", p_tab[i].u_s);
        // printf("%13d\t", p_tab[i].w_t);
        printf("\t\t%p\t", (void *)&p_tab[i]);
    printf("\t\t\t0x%-4x\t", p_tab[i].p_phys_addr);
    printf("\n");
    }
}

int main(int argc, const char *argv[]) {

  mmu_init();
  mmu_print();

  printf("\n");

  return 0;
}
```

.