

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.

/Григорьев Ю.В./

Проверил:
ассистент
«__» декабря 2023 г.

/Третьяков Г.Н./

Оценка « _____ »

Новосибирск 2023

ВЫПОЛНЕНИЕ РАБОТЫ

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

Обычный режим трансляции адресов на диаграмме:



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

Реализация:

1. **Определения и Структуры:** В начале кода определяются структуры **PDE** (Page Directory Entry) и **PTE** (Page Table Entry) для представления записей в каталоге страниц и таблицах страниц соответственно. Эти структуры содержат поля, представляющие различные атрибуты страницы (например, присутствует ли страница в памяти, разрешение на чтение/запись, адрес и т.д.).
2. **Алгоритмы и Функции:**
 - a. **translate_from_logic:** Преобразует логический (виртуальный) адрес в физический адрес.
 - b. **mmu_print:** Выводит информацию о состоянии каталога страниц и таблиц страниц.
 - c. **main:** Инициализирует структуры управления памятью и запускает печать информации о MMU (Memory Management Unit).
3. **Инициализация MMU:** В **main** происходит инициализация и настройка каталога страниц и таблиц страниц. Это включает в себя вычисление количества необходимых страниц и таблиц страниц, а также установку различных атрибутов страниц.

Отображение виртуальной памяти в x86_64

В архитектуре **x86_64**, отображение виртуальной памяти на физическую реализуется через механизм, называемый страничной адресацией. Он использует каталоги страниц и таблицы страниц для управления отображением адресов:

1. **Каталоги страниц (Page Directories):** Содержат записи, каждая из которых указывает на таблицу страниц. Эти записи управляют атрибутами доступа к страницам.
2. **Таблицы страниц (Page Tables):** Каждая запись в таблице страниц соответствует отдельной странице виртуальной памяти и указывает на её физическое расположение.
3. **Процесс перевода адресов:**
 - ЦПУ генерирует виртуальный адрес.
 - С использованием CR3 (регистр управления), система находит соответствующую запись в каталоге страниц.
 - Запись каталога страниц указывает на таблицу страниц.
 - Таблица страниц переводит виртуальный адрес в физический.
4. **Атрибуты страниц:** Атрибуты, такие как права доступа (чтение/запись), управление кэшированием, присутствие в памяти, используются для контроля доступа к памяти и её эффективного использования.

В контексте представленного кода эти концепции используются для создания простой модели управления памятью, которая может быть расширена или модифицирована для более сложных сценариев управления памятью в операционных системах, работающих на архитектуре x86_64.

Демонстрация работы: (результаты выведены в файл)

При запуске вводится аргумент, задающий размер памяти. Даже если размер памяти равен < 4096 байт, необходимо выделить целую страницу памяти (4096 Б = 4 КБ) и разметить её.

```
1 memsize = 10, pgdiraddr = 0x10d0b6000, pgtabcnt = 1, pgcnt = 1
2
3 MMU table
4 present RW user_supervisor writethru cache accessed dirty pgsz ignore pgtableaddr
5 1 1 1 1 1 0 1 1 0 0xd0b7
6 present RW user_supervisor writethru cache accessed dirty PAT ignore physaddr virtaddr
7 1 1 1 1 0 0 0 0 1 0xd4b7 0x0
8 1 1 1 1 0 0 0 0 1 0xd4b8 0x1000
9 1 1 1 1 0 0 0 0 1 0xd4b9 0x2000
10 1 1 1 1 0 0 0 0 1 0xd4ba 0x3000
11 1 1 1 1 0 0 0 0 1 0xd4bb 0x4000
12 1 1 1 1 0 0 0 0 1 0xd4bc 0x5000
13 1 1 1 1 0 0 0 0 1 0xd4bd 0x6000
14 1 1 1 1 0 0 0 0 1 0xd4be 0x7000
15 1 1 1 1 0 0 0 0 1 0xd4bf 0x8000
16 1 1 1 1 0 0 0 0 1 0xd4c0 0x9000
17 1 1 1 1 0 0 0 0 1 0xd4c1 0xa000
18 1 1 1 1 0 0 0 0 1 0xd4c2 0xb000
19 1 1 1 1 0 0 0 0 1 0xd4c3 0xc000
```

(записи продолжаютсся и дальше)

Объяснение всех полей (флагов) в структурах:

В структуре **PDE (Page Directory Entry)**:

1. **P (Present)**: Этот флаг указывает, присутствует ли страница в физической памяти (RAM). Если этот флаг установлен в 1, это означает, что страница находится в памяти. Если 0, это означает, что страница отсутствует и должна быть загружена.
2. **RW (Read-Write)**: Этот флаг определяет, может ли страница быть записана (RW=1) или только для чтения (RW=0).
3. **US (User/Supervisor)**: Этот флаг указывает, может ли страница быть доступна для пользовательских программ (US=1) или только для режима супервизора (US=0).
4. **PWT (Page Write Through)**: Этот флаг определяет, используется ли кэширование с записью через (Write-Through) для этой страницы.
5. **PCD (Page Cache Disable)**: Этот флаг указывает, разрешено ли кэширование страницы. Если установлен в 1, кэширование отключено.
6. **A (Accessed)**: Этот флаг устанавливается, когда страница была доступна (читалась или записывалась).
7. **D (Dirty)**: Этот флаг устанавливается, когда страница была изменена (записана).
8. **PS (Page Size)**: Этот флаг указывает размер страницы. Если установлен в 1, страница имеет размер 2 МБайта, в противном случае - 4 КБайта.
9. **G (Global)**: Глобальность. Этот флаг игнорируется в архитектуре x86_64 и не имеет значения. Ранее, в архитектуре x86 до появления x86_64 (также известной как IA-32e или AMD64), флаг G (Global) в таблице страниц (Page Table Entry, PTE) указывал, что страница, на которую указывает данная запись в таблице страниц, является "глобальной", что означало, что страница не будет удаляться из кэша перевода адресов (Translation Lookaside Buffer, TLB) при выполнении инструкций перехода (например, инструкций перехода между задачами или инструкций перехода между уровнями привилегий). Флаг G (Global) был полезным в многозадачных операционных системах, где необходимо было обеспечивать быстрый доступ к страницам, общим для всех задач, чтобы избежать дорогостоящей операции очистки TLB при каждом переключении задачи. Когда страница была помечена как глобальная, она оставалась в TLB даже после переключения задачи.
10. **Reserved (Зарезервированные биты)**: Эти биты зарезервированы для будущего использования и должны быть установлены в 0.
11. **Address (Адрес таблицы страниц)**: Это поле содержит физический адрес таблицы страниц, к которой относится данная запись.

В структуре **PTE (Page Table Entry)**:

1. **P (Present)**: Этот флаг указывает, присутствует ли страница в физической памяти (RAM). Если этот флаг установлен в 1, это означает, что страница находится в памяти. Если 0, это означает, что страница отсутствует и должна быть загружена.
2. **RW (Read-Write)**: Этот флаг определяет, может ли страница быть записана (RW=1) или только для чтения (RW=0).
3. **US (User/Supervisor)**: Этот флаг указывает, может ли страница быть доступна для пользовательских программ (US=1) или только для режима супервизора (US=0).
4. **PWT (Page Write Through)**: Этот флаг определяет, используется ли кэширование с записью через (Write-Through) для этой страницы.
5. **PCD (Page Cache Disable)**: Этот флаг указывает, разрешено ли кэширование страницы. Если установлен в 1, кэширование отключено.
6. **A (Accessed)**: Этот флаг устанавливается, когда страница была доступна (читалась или записывалась).
7. **D (Dirty)**: Этот флаг устанавливается, когда страница была изменена (записана).
8. **PAT (Page Attributes Table)**: Этот флаг используется для указания атрибутов страницы в соответствии с таблицей атрибутов страниц.
9. **G (Global)**: Этот флаг игнорируется в архитектуре x86_64 и не имеет значения.
10. **Reserved (Зарезервированные биты)**: Эти биты зарезервированы для будущего использования и должны быть установлены в 0.
11. **ADDRESS (Физический адрес страницы)**: Это поле содержит физический адрес страницы в памяти.

В структуре **page**:

page представляет собой структуру, которая используется для хранения физических адресов страниц в памяти. В данном контексте она служит для иллюстрации, как могут быть представлены физические адреса страниц. Она содержит массив **frames**, где каждый элемент массива представляет собой физический адрес страницы.

Приложение

Содержимое файла **main.c**:

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>

#define attribute(value) __attribute__((value))
#define MMU_PTE_COUNT 1024
#define MMU_PDE_COUNT 1024
#define malloc_a(size, align) kmalloc_a(size, align)
```



```

for (uint32_t i = 0; i < page_tab_count; i++) {
    // pages directory
    printf("%d\t\t", (page_dir[i]).P);
    printf("%d\t", (page_dir[i]).RW);
    printf("%d\t\t\t\t", (page_dir[i]).US);
    printf("%d\t\t\t", (page_dir[i]).PWT);
    printf("%d\t\t", (page_dir[i]).PCD);
    printf("%d\t\t\t", (page_dir[i]).A);
    printf("%d\t\t", (page_dir[i]).D);
    printf("%d\t\t", (page_dir[i]).PS);
    printf("%d\t\t", (page_dir[i]).G);
    printf("0x%x\n", (uint32_t)((page_dir[i]).address));
    // pages table

printf("present\tRW\tuser_supervisor\twritethru\tcache\taccessed\tdirty\tPAT\ti
gnore\tphysaddr\tvirtaddr\n");
    for (uint32_t j = 0; j < MMU_PTE_COUNT; j++) {
        printf("%d\t\t", (page_tab[i][j]).P);
        printf("%d\t", (page_tab[i][j]).RW);
        printf("%d\t\t\t\t", (page_tab[i][j]).US);
        printf("%d\t\t\t", (page_tab[i][j]).PWT);
        printf("%d\t\t", (page_tab[i][j]).PCD);
        printf("%d\t\t\t", (page_tab[i][j]).A);
        printf("%d\t\t", (page_tab[i][j]).D);
        printf("%d\t", (page_tab[i][j]).PAT);
        printf("%d\t\t", (page_tab[i][j]).G);
        printf("%p\t\t", (void *) (uintptr_t)((page_tab[i][j]).ADDRESS));
        printf("0x%x\n", (uint32_t)((i << 22) + ((j << 22) >> 10)); // virtual
address
    }
}

int main(int argc, char **argv) {
    cr3 = (uintptr_t)page_dir >> 12 << 12; // control register 3 - no cache, PAE
off

    uint32_t bin_size; // size of memory in bytes
    if (argc > 1) { // if cli arg[1] exists
        uint8_t i = 0;
        while (argv[1][i] <= '9' && argv[1][i] >= '0') {
            bin_size = bin_size * 10 + (uint32_t)(argv[1][i] - '0');
            i++;
        }
        if (argv[1][i] == 'K' || argv[1][i] == 'k') {
            bin_size *= 1024;
        } else if (argv[1][i] == 'M' || argv[1][i] == 'm') {
            bin_size *= (1024 * 1024);
        }
    } else { // default - 4 KB * 16
        bin_size = 4096 * 16;
    }

    uint32_t pages_count = bin_size / 4096;
    if ((bin_size % 4096) > 0)
        pages_count++;

    uint32_t page_tab_count = pages_count / 1024;
    if ((pages_count % 1024) > 0)
        page_tab_count++;

    for (uint16_t i = 0; i < page_tab_count; i++) {
        page_dir[i].P = 1;

```

```

    page_dir[i].RW = 1;
    page_dir[i].US = 1;
    page_dir[i].PWT = 1;
    page_dir[i].PCD = 1;
    page_dir[i].A = 0;
    page_dir[i].D = 1;
    page_dir[i].PS = 1; // 4 KB
    page_dir[i].address = (uintptr_t)(&page_tab[i]) >> 12;
    for (uint32_t j = 0; j < MMU_PTE_COUNT; j++) {
        page_tab[i][j].D = 0;
        page_tab[i][j].A = 0;
        page_tab[i][j].PAT = 0;
        page_tab[i][j].PCD = 0;
        page_tab[i][j].G = 1;
        page_tab[i][j].P = 1;
        page_tab[i][j].RW = 1;
        page_tab[i][j].US = 1;
        page_tab[i][j].PWT = 1;
        page_tab[i][j].ADDRESS = ((uintptr_t)(&page_0) + j * 4096) >> 12; //
        assume 4 KB pages
    }
}
printf("memsize = %d, ", bin_size);
printf("pgdiraddr = %p, ", (void *)page_dir);
printf("pgtabcnt = %d, ", page_tab_count);
printf("pgcnt = %d\n\n", pages_count);
printf("MMU table\n");
mmu_print(page_tab_count, pages_count);
return 0;
}

```