**Лабораторная работа 2**

**Описание**

1. Для начала задаем размер страницы:

#define PAGE\_DEFAULT\_SIZE 1024

1. Далее определяем структуру страниц, исходя из их размера:  
     
   struct page { uint32\_t frames[PAGE\_DEFAULT\_SIZE]; };
2. Определяем структуру PDE (Page Directory Entry), которая имеет следующие биты:

* P (Present): Бит присутствия. Если установлен в 1, это означает, что страница присутствует в памяти.
* RW (Read/Write): Бит чтения/записи. Если установлен в 1, разрешено чтение и запись в страницу.
* US (User/Supervisor): Бит пользователя/супервизора. Если установлен в 1, страница доступна для пользовательского режима.
* PWT (Page-Level Write-Through): Бит уровня страницы для записи через. Определяет стратегию кэширования.
* PCD (Page-Level Cache Disable): Бит уровня страницы для отключения кэша.
* A (Accessed): Бит доступа. Устанавливается процессором при каждом обращении к странице.
* D (Dirty): Бит "грязи". Устанавливается процессором, если в странице производится запись.
* PS (Page Size): Бит размера страницы. Если установлен в 1, это означает, что используется большая страница (4 МБ) вместо стандартной (4 КБ).
* G (Global): Бит глобальной страницы. Если установлен в 1, страница не удаляется из кэша TLB при смене контекста.
* reserved: Зарезервированные биты. Не используются, зарезервированы для будущего использования.
* address: Адрес базы физической страницы (20 бит).

Также не забываем про атрибут \_\_attribute\_\_((packed)), который указывает компилятору упаковывать структуру так, чтобы не вставлять дополнительные байты для выравнивания.

1. Структура PTE:

* P (Present): Бит присутствия. Если установлен в 1, это означает, что страница присутствует в памяти.
* RW (Read/Write): Бит чтения/записи. Если установлен в 1, разрешено чтение и запись в страницу.
* US (User/Supervisor): Бит пользователя/супервизора. Если установлен в 1, страница доступна для пользовательского режима.
* PWT (Page-Level Write-Through): Бит уровня страницы для записи через. Определяет стратегию кэширования.
* PCD (Page-Level Cache Disable): Бит уровня страницы для отключения кэша.
* A (Accessed): Бит доступа. Устанавливается процессором при каждом обращении к странице.
* D (Dirty): Бит "грязи". Устанавливается процессором, если в странице производится запись.
* PAT (Page Attribute Table): Бит таблицы атрибутов страницы.
* G (Global): Бит глобальной страницы. Если установлен в 1, страница не удаляется из кэша TLB при смене контекста.
* reserved: Зарезервированные биты. Не используются, зарезервированы для будущего использования.
* address: Адрес базы физической страницы (20 бит).

1. Обе структуры PDE (Page Directory Entry) и PTE (Page Table Entry) представляют записи в управлении памятью x86 и содержат атрибуты страниц. Однако есть несколько ключевых различий:

* Контекст использования:
* PDE используется для описания записей в таблице страниц директории (Page Directory).
* PTE используется для описания записей в таблице страниц (Page Table).
* Адреса страниц:
* PDE содержит адрес базы физической страницы размером 4 МБ (20 бит), что означает, что одна запись в таблице директории может адресовать область 4 МБ памяти.
* PTE содержит адрес базы физической страницы размером 4 КБ (20 бит), что означает, что одна запись в таблице страниц может адресовать область 4 КБ памяти.
* Бит PS (Page Size):
* PDE имеет бит PS, который указывает, используется ли большая страница (4 МБ) или стандартная страница (4 КБ).
* PTE не имеет этого бита, так как она предназначена для описания только стандартных страниц (4 КБ).
* Бит PAT (Page Attribute Table):
* PTE включает бит PAT, который относится к таблице атрибутов страницы.
* PDE не содержит бит PAT.

1. Далее создаем массивы и структуры, связанные с управлением памятью:

* page\_dir: массив page\_dir, представляющий таблицу страниц директории (PDE). Этот массив имеет размер PAGE\_DEFAULT\_SIZE и выравнивается по границе размера страницы (4 КБ) умноженной на PAGE\_DEFAULT\_SIZE. Каждый элемент массива имеет тип struct PDE.
* page\_tab: двумерный массив page\_tab, представляющий таблицу страниц (PTE). Размерность массива PAGE\_DEFAULT\_SIZE x PAGE\_DEFAULT\_SIZE. Также, как и page\_dir, он выравнивается по границе размера страницы (4 КБ) умноженной на PAGE\_DEFAULT\_SIZE. Каждый элемент массива имеет тип struct PTE.
* page\_0: структура page\_0, представляющая страницу памяти. Эта структура будет использоваться для инициализации таблиц страниц.
* cr3: переменная cr3 типа uint32\_t, которая будет содержать значение регистра управления страницей (CR3). Этот регистр хранит базовый адрес таблицы страниц директории и будет использоваться при трансляции логических адресов в физические.

1. Функция print\_mmu предназначена для визуализации иерархии страниц в управлении памятью x86. Она выводит информацию о таблицах страниц и соответствующих адресах виртуальной и физической памяти:

* void print\_mmu(uint32\_t page\_tab\_count, uint32\_t page\_count);
* page\_tab\_count - количество таблиц страниц, и page\_count - общее количество страниц.
* Функция использует стандартный вывод printf для вывода информации в консоль.
* Вложенные циклы и выражения используются для обхода и вывода адресов каждой страницы в соответствующей таблице.
* При выводе адресов, они сдвигаются на 12 бит влево, учитывая размер страницы в 4 КБ.

1. Функция main:

* Значение cr3 устанавливается как базовый адрес таблицы страниц директории, сдвинутый на 12 бит вправо и затем влево. Это значение будет использоваться для загрузки адреса таблицы страниц директории в регистр CR3 процессора:

|  |  |
| --- | --- |
| 1 | cr3 = ((**uintptr\_t**)page\_dir >> **12**) << **12**; |

* Определяется размер бинарного файла (bin\_size), а затем вычисляется количество страниц (page\_count) на основе размера страницы (4 КБ):

|  |  |
| --- | --- |
| 1  2  3 | **uint32\_t** bin\_size = **4096** \* **16**;  **uint32\_t** page\_count = bin\_size / **4096**;  **if** ((bin\_size % **4096**) > **0**) page\_count++; |

* Определяется количество таблиц страниц директории (page\_tab\_count) на основе общего количества страниц. Каждая таблица содержит 1024 записи:

|  |  |
| --- | --- |
| 1  2 | **uint32\_t** page\_tab\_count = page\_count / **1024**;  **if** ((page\_count % **1024**) > **0**) page\_tab\_count++; |

* Итерируется по таблицам страниц директории, и каждая запись инициализируется с установленными атрибутами (например, права доступа, размер страницы). Вложенный цикл инициализирует каждую страницу в таблице страниц:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **for** (**uint16\_t** i = **0**; i < page\_tab\_count; i++) {  page\_dir[i] = (**struct** PDE) {  .P = **1**, .RW = **1**, .US = **1**, .PWT = **1**,  .PCD = **1**, .A = **0**, .D = **1**,  .PS = **1**,  .address = (**uintptr\_t**)(&page\_tab[i]) >> **12**};  **for** (**uint32\_t** j = **0**; j < PAGE\_DEFAULT\_SIZE; j++) {  page\_tab[i][j] = (**struct** PTE){ .D = **0**,  .A = **0**, .PAT = **0**, .PCD = **0**,  .G = **1**, .P = **1**, .RW = **1**, .US = **1**, .PWT = **1**,  .address = ((**uintptr\_t**)(&page\_0) + j \* **4096**) >> **12**};  }  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | printf("memsize = %d, ", bin\_size);  printf("pgdiraddr = %p, ", (**void** \*)page\_dir);  printf("pgtabcnt = %d, ", page\_tab\_count);  printf("pgcnt = %d**\n\n**", page\_count);  printf("MMU table**\n**");  print\_mmu(page\_tab\_count, page\_count); |

* Выводится информация о размере памяти, адресе таблицы страниц директории, количестве таблиц страниц и общем количестве страниц. Вызывается функция print\_mmu для вывода информации о таблицах MMU:

**Листинг кода**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | #include <inttypes.h>  #include <stdio.h>  #include <stdlib.h>  #define PAGE\_DEFAULT\_SIZE 1024  **struct** page {  **uint32\_t** frames[PAGE\_DEFAULT\_SIZE];  };  **struct** PDE {  **uint32\_t** P : **1**;  **uint32\_t** RW : **1**;  **uint32\_t** US : **1**;  **uint32\_t** PWT : **1**;  **uint32\_t** PCD : **1**;  **uint32\_t** A : **1**;  **uint32\_t** D : **1**;  **uint32\_t** PS : **1**;  **uint32\_t** G : **1**;  **uint32\_t** reserved : **3**;  **uint32\_t** address : **20**;  } \_\_attribute\_\_((packed));  **struct** PTE {  **uint32\_t** P : **1**;  **uint32\_t** RW : **1**;  **uint32\_t** US : **1**;  **uint32\_t** PWT : **1**;  **uint32\_t** PCD : **1**;  **uint32\_t** A : **1**;  **uint32\_t** D : **1**;  **uint32\_t** PAT : **1**;  **uint32\_t** G : **1**;  **uint32\_t** reserved : **3**;  **uint32\_t** address : **20**;  } \_\_attribute\_\_((packed));  **struct** PDE page\_dir[PAGE\_DEFAULT\_SIZE] \_\_attribute\_\_((aligned(PAGE\_DEFAULT\_SIZE \* **4**)));  **struct** PTE page\_tab[PAGE\_DEFAULT\_SIZE][PAGE\_DEFAULT\_SIZE] \_\_attribute\_\_((aligned(PAGE\_DEFAULT\_SIZE \* **4**)));  **struct** page page\_0;  **uint32\_t** cr3;  **void** **print\_mmu**(**uint32\_t** page\_tab\_count, **uint32\_t** page\_count) {  printf("table addr**\n**");  **for** (**uint32\_t** i = **0**; i < page\_tab\_count; ++i) {  printf("0x%x**\n**", (**uint32\_t**)((page\_dir[i]).address));  printf("physical**\t**virtual**\n**");  **for** (**uint32\_t** j = **0**; j < PAGE\_DEFAULT\_SIZE; ++j) {  printf("%p**\t\t**", (**void** \*)(**uintptr\_t**)((page\_tab[i][j]).address << **12**));  printf("%p**\t\t**", (**void** \*)(**uintptr\_t**)(((i << **10**) | j) << **12**));  }  }  }  **int** **main**(**int** argc, **char** \*\*argv) {  cr3 = ((**uintptr\_t**)page\_dir >> **12**) << **12**;  **uint32\_t** bin\_size = **4096** \* **16**;  **uint32\_t** page\_count = bin\_size / **4096**;  **if** ((bin\_size % **4096**) > **0**) page\_count++;  **uint32\_t** page\_tab\_count = page\_count / **1024**;  **if** ((page\_count % **1024**) > **0**) page\_tab\_count++;  **for** (**uint16\_t** i = **0**; i < page\_tab\_count; i++) {  page\_dir[i] = (**struct** PDE) {  .P = **1**, .RW = **1**, .US = **1**, .PWT = **1**,  .PCD = **1**, .A = **0**, .D = **1**,  .PS = **1**,  .address = (**uintptr\_t**)(&page\_tab[i]) >> **12**};  **for** (**uint32\_t** j = **0**; j < PAGE\_DEFAULT\_SIZE; j++) {  page\_tab[i][j] = (**struct** PTE){ .D = **0**,  .A = **0**, .PAT = **0**, .PCD = **0**,  .G = **1**, .P = **1**, .RW = **1**, .US = **1**, .PWT = **1**,  .address = ((**uintptr\_t**)(&page\_0) + j \* **4096**) >> **12**};  }  }    printf("memsize = %d, ", bin\_size);  printf("pgdiraddr = %p, ", (**void** \*)page\_dir);  printf("pgtabcnt = %d, ", page\_tab\_count);  printf("pgcnt = %d**\n\n**", page\_count);  printf("MMU table**\n**");  print\_mmu(page\_tab\_count, page\_count);  } |