Домашнее задание 2.
Аллокация памяти.

Я

February 21, 2016

Contents

1 Основное задание

В этом домашнем задании вам необходимо реализовать аллокаторы памяти, которые вы будете использовать в дальнейших домашних заданиях - очень не желательно пропускать это задание, а кроме того учите, что вам придется пользоваться тем интерфейсом, который вы для себя создадите.

Для выполнения этого задания вам, скорее всего, придется настроить Paging под свои нужды, рекомендации по тому как это сделать даны после задания. Кроме того вам *настоятельно* рекомендуется прочитать внимательно раздел об изменениях внесенных в предоставляемые вам файлы, чтобы вы могли разрешить конфиликты слияния без проблем.

1. Получить и вывести карту физической памяти. Добавьте в карту памяти участок физической памяти занятый вашим ядром.

2. Реализовать аллокатор страниц физической памяти. Для аллокации вы можете воспользоваться одним из известных алгоритмов (Buddy Allocator или его модификации) или придумать свой алгоритм. Какой бы вариант вы не выбрали он должен удовлетворять следующим требованиям:

   - вы должны уметь аллоцировать память как минимум постранично;
   - страница должна быть выровнена по размеру страницы, т. е. недостаточно просто вернуть любой блок памяти размером в 4KB (2MB);
   - страница должна возвращается пользователю целиком, т. е. нельзя хранить служебную информацию аллокатора в аллоцированной странице;
   - страницы нужно уметь освобождать (очевидно);

3. Реализовать аллокатор блоков фиксированного размера (что-то похожее на SLAB [Bonwick, 1994]). Опять же вы можете выбирать любой вариант реализации, какой захотите, пока он удовлетворяет следующим требованиям:

   - вы должны уметь создавать аллокатор блоков фиксированного размера для любого размера в пределах от 1B до 4KB (не включительно);
   - аллокатор должен возвращать указатель готовый к использованию;

- должна быть предусмотрена возможность выравнивания аллоцируемых объектов по указанной при создании аллокатора границе;
- указатель должен быть в верхней части адресного пространства, т. е. после "канонической дыры";
- память нужно уметь освобождать (очевидно);

2 Дополнительные задания

- Реализовать аллокатор памяти общего назначения, т. е. с интерфейсом на подобии malloc/free (названия функций можете выбирать на свое усмотрение);
- Реализовать функции отображения и удаления отображения физических страниц в последовательные участки виртуального адресного пространства, т. е. отображение непоследовательных физических страниц в последовательную виртуальную память; При этом должны быть выполнены следующие требования:
  - диапазон виртуальных адресов должен выбираться автоматически, а не передаваться как параметр;
  - нельзя использовать виртуальные адреса меньше "канонической дыры";

3 Начальная таблица страниц

Когда main получает управление код уже исполняется с включенным paging-ом, т. е использует некоторую таблицу страниц. Эта начальная таблица страниц содержит отображения для трех регионов виртуальной памяти на первые 2GB физической памяти. Далее следует краткое описание каждого из них.

Identity Mapping. Для перехода в 64-битный режим документация [INT,] в разделе 9.8.5 Initializing IA-32e Mode требует отображения 1 к 1 в начальной таблице страниц. Т. е. эта часть таблицы страниц отображает первые 4GB виртуальной памяти на первые 4GB физической[1].

[1] В оригинальной версии исходников эта часть отображала только первые 2GB, и была изменена после первого домашнего задания, т. е. не забудьте обновить свою

Участок сразу за "канонической дырой". Этот участок в 4GB отображается на первые 4GB физической памяти. В целом участок виртуальной памяти начиная от "канонической дыры" до ядра предполагается использовать чтобы отобразить его на всю физическую память. Такое отображение сделает преобразование физического адреса в виртуальным и обратно тривиальным для данных ядра. Однако обратите внимание, что использовать тоже самое отображение для кода ядра не получится.

Последние 2GB виртуальной памяти. Этот участок также отображается на первые 2GB физической памяти и в этом виртуальном адресном пространстве работает код ядра, так же обитают статические данные ядра. Этот участок используется потому что ядро использует kernel code model [Matz et al., 2013].

Все три отображения используют страницы размеров 2MB. Вам рекомендуется создать свою таблицу страниц взамен ограниченной начальной страницы, после того как вы реализуете аллокатор физических страниц по следующим правилам:

- отобразите верхние 2GB виртуального адресного пространства на первые 2GB физической памяти используя любой размер страницы[1];
- отобразите виртуальную память сразу за "канонической дырой" на всю физическую память, размер которой вам известен из карты памяти (опять же размер страницы остается на ваше усмотрение);
- не нужно задумываться о "специальных" участках физической памяти, которые могут отображаться на регистры устройств или дырах в физической памяти при построении отображения - они не доставят проблем пока вы не будете их использовать;
- отображайте все страницы с правами на запись (PTE_WRITE в предоставленном файле paging.h) и без права доступа непривилегированным пользователям (без флага PTE_USER из файла paging.h) - нам нет особого смысла заморачиваться защитой памяти ядра от самого ядра;

версию.

[1]Выбор у вас небольшой 2MB или 4KB, очевидно использование страниц в 2MB сэкономит память и будет быстрее.

Детальная информация о структуре таблицы страниц, поддерживаемых размерах страниц и флагах доступна в официальной документации [INT,] в главе 4 Paging, нас более всего интересует в этой главе описание IA-32e режима Paging-a, который мы и используем.

Сброс TLB. Если вы меняете отображение, то вам необходимо сбросить соответствующие записи в TLB. Причем, если отображения для какой-то страницы отсутствовало в таблице страниц и вы его создали, то вы, по хорошему, должны сбрасывать TLB и в этом случае тоже[1].

Для сброса TLB есть две возможности:

- перезаписать значение в cr3[2] - в этом случае сбрасывается весь TLB, чего, по возможности, стоит избегать;
- использовать инструкцию invlpg, которая старается сбрасывать из TLB только записи соответствующие переданному ей виртуальному адресу [3];

В предоставляемым вместе с заданием файле paging.h описаны вспомогательные функции для работы с таблицами страниц - вам предлагается ознакомиться с ними самостоятельно. Среди них есть функции для сброса TLB (flush_tlb_addr и flush_tlb) и записи/чтения физического адреса корневой таблицы страниц в/из регистра cr3 (load_pml4 и store_pml4 соответственно).

4 Карта памяти

Получение карты памяти - это чисто формальная часть задания. На самом деле карта памяти предоставляется multiboot загрузчиком. Те кто не использует предоставленный bootstrap.S должны будут разобраться с получением карты памяти самостоятельно. Для всех остальных эта инструкция.

[1] На лекции я говорил по другому, но это относилось только Intel-овским CPU, есть и другие производители x86-like процессоров - и нет, это не только AMD - которые могут требовать сброса TLB даже в таком нелепом случае.

[2] Тут тоже не все так гладко, но я не смог вспомнить/найти проблемы связанные с этим способом - вроде какие-то совсем странные производители процессоров в этой ситуации не сбрасывали TLB.

[3] Если инструкцию invlpg реализовать как сброс всего TLB, то это тоже будет корректным поведением, но на нормальных CPU сброс одной записи обойдется дешевле во многих смыслах.

4.1 Получение карты памяти

Собственно, bootstrap.S уже получает *физический* адрес multiboot information structure, формат которого приведен в [mul,] (раздел 3.3 Boot information format), вам остается только проверить, что поля mmap_length и mmap_addr валидны[1] и распарсить и использовать эту информацию.

Чтобы получить адрес multiboot information structure добавьте в свой код следующие строки:

```
1 extern const uint32_t mboot_info;
```

После этого вы можете использовать mboot_info - в этой переменной и содержится адрес multiboot information structure. Обратите внимание, на тип - uint32_t:

- вам нужно подключить заголовок stdint.h, чтобы использовать этот тип;
- адерс 32-битный, т. е. находится в первых 4GB физической памяти.

Несмотря на то, что адрес multiboot information structure физический мы можем преобразовать его к указателю, так как у нас есть настроенный identity mapping в начальной таблице страниц, который покрывает первые 4GB памяти.

4.2 Резервирование памяти ядра

Карта памяти получается (условно) от аппаратуры компьютера, которая ничего не знает о ядре ОС. Соответственно, в этой карте памяти не отображено, что часть памяти занята ядром ОС - вам необходимо зарезервировать эту память самостоятельно. В противном случае ваши аллокаторы могут ее аллоцировать и вы перезапишите уже занятую память.

Для того чтобы этого не случилось вам нужно получить границы памяти занятой ядром ОС, для этого добавьте в код следующие строки:

```
1 extern char text_phys_begin[];
2 extern char bss_phys_end[];
```

После чего вы можете преобразовать text_phys_begin в 64 битное число[2] которое содержит физический адрес начала вашего ядра, а если вы преобразуете bss_phys_end в число, то получите адрес конца ядра.

[1] Для этого разберитесь с описанием поля flags

[2] На самом деле старшие 32 бита должны быть нулевыми.

5 Boot Allocator

Если вы решите использовать одну из модификация Buddy Allocator-а, то вам, вероятно, понадобится еще один аллокатор для аллокации служебной информации, чтобы инициализировать Buddy Allocator. Ситуация усугубляется тем, что вы к этому моменту еще не успеете загрузить свою полноценную таблицу страниц [1].

Проблема однако решается довольно просто - вам нужен еще один, очень простой аллокатор, который будет аллоцировать участки физической памяти из тех областей, для которых уже есть отображение в начальной таблице страниц (на первые 4GB физической памяти у вас отображены два участка виртуальной - вам предлагается использовать участок сразу за "канонической дырой") используя информацию из карты памяти.

На этот аллокатор не налагается серьезных ограничений, т. е.:

- вы можете ограничить количество объектов, которые этот аллокатор может аллоцировать в принципе (хранить информацию о них в статическом массиве);
- вам не обязательно поддерживать освобождение (память нужная Buddy Allocator-у никогда не будет освобождена);

Однако вам необходимо проследить, чтобы аллоцированные участки памяти были зарезервированы (так же как это должно быть сделано для памяти ядра), чтобы никто больше не мог их использовать.

6 Изменения в предоставляемом коде

В предоставленный для первого задания код были внесенные кое-какие изменения:

- в kernel.ld было добавлена память под начальную таблицу страниц - если вы не вносили изменений kernel.ld, то конфликтов быть не должно;
- bootstrap.S - identity mapping и виртуальная память после "канонической дыры" теперь отображены на первые 4GB физической памяти;

[1] Для того, чтобы ее создать, скорее всего, нужен аллокатор страниц, которого еще нет.

- в файл memory.h были добавлены функции pa и va, которые позволяют получить по виртуальному адресу из участка сразу после "канонической дыры" физический адрес и наоборот.

Кроме того к коду был добавлен еще один заголовочный файл - paging.h. Этот файл содержит следующие функции:

- pte_preset, pte_user, pte_write и pte_large - проверяют, установлен ли соответствующий бит в записи таблицы страниц (бит описываются соответствующими define-ами);
- pte_phys - достает из записи таблицы страниц физический адрес таблицы следующего уровня или страницы памяти, на которую отображен соответствующий виртуальный адрес;
- pml_i - функции позволяющие получить индекс в таблице соответсвующего уровня (короче говоря, они парсят виртуальный адрес);
- page_off - возвращает по виртуальному адресу смещение внутри 4KB страницы (оставшаяся часть парсинга виртуального адреса);
- canonical и linear - преобразовывают виртуальный адрес в не каноническом формате в канонический, и наоборот;
- load_pml4 - загружается в cr3 переданный физический адрес;
- store_pml4 - возвращает значение записанное в cr3;
- flush_tlb_addr - сбрасывает запись TLB соответствующую переданному виртуальному адресу;
- flush_tlb - сбрасывает все записи в TLB (просто перезаписывает cr3 старым значением).

7 Вопросы и ответы

References

[INT,] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume3: System Programming Guide. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

[mul,] Multiboot specification version 0.6.96. https://www.gnu.org/software/grub/manual/multiboot/multiboot.html.

[Bonwick, 1994] Bonwick, J. (1994). The slab allocator: An object-caching kernel memory allocator. https://www.usenix.org/legacy/publications/library/proceedings/bos94/full_papers/bonwick.a.

[Matz et al., 2013] Matz, M., Hubicka, J., Jaeger, A., and Mitchell, M. (2013). System v application binary interface. amd64 architecture processor supplement. draft version 0.99.6. http://www.x86-64.org/documentation/abi.pdf.