МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Московский физико-технический институт (государственный университет)»

«УТВЕРЖДАЮ
Проректор по учебной и методической работе
_____ Д.А. Зубцов

Рабочая программа дисциплины (модуля)

по дисциплине: Конструирование ядра операционной системы
по направлению: Прикладные математика и физика (магистратура)
профиль подготовки: Системное программирование
факультет: управления и прикладной математики
кафедра: системного программирования
курс: 1
квалификация: магистр

Семестр, формы промежуточной аттестации: 1(Осенний) - Дифференцированный зачет

Аудиторных часов: 30 всего, в том числе:
лекции: 0 час.
практические (семинарские) занятия: 0 час.
лабораторные занятия: 30 час.

Самостоятельная работа: 6 час.

Всего часов: 36, всего зач. ед.: 1

Программу составил: А.В. Хорошилов, к.ф.-м.н.

Программа обсуждена на заседании кафедры

10 июля 2015 г.

СОГЛАСОВАНО:

Декан факультета управления и прикладной математики

Начальник учебного управления И.Р. Гарайшина

1. Цели и задачи

Цель *дисциплины*

- изучение основных принципов внутреннего устройства ядра операционной системы, механизмов аппаратной поддержки работы ядра, а также получение навыков проектирования и программирования компонентов ядра операционной системы и отладки программ в привилегированном режиме работы процессора.

Задачи дисциплины

- освоение студентами основных принципов внутреннего устройства ядра операционной системы;
- приобретение теоретических знаний и практических умений и получение навыков проектирования и программирования компонентов ядра операционной системы и отладки программ в привилегированном режиме работы процессора..

2. Место дисциплины (модуля) в структуре образовательной программы

Дисциплина " Конструирование ядра операционной системы " относится к вариативной части образовательной программы

Дисциплина «Конструирование ядра операционной системы» базируется на дисциплинах:
Прикладные физико-технические и компьютерные методы исследований. Системное программное обеспечение; Информатика.

Дисциплина «Конструирование ядра операционной системы» предшествует изучению дисциплин:
Научно-исследовательская работа.

3. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесенных с планируемыми результатами освоения образовательной программы

Освоение дисциплины направлено на формирование следующих общекультурных, общепрофессиональных и профессиональных компетенций:

способность применять современные методы анализа, обработки и представления информации в сфере профессиональной деятельности (ОПК-5);

способность осуществлять научный поиск и разработку новых перспективных подходов и методов к решению профессиональных задач, способностью к профессиональному росту (ОПК-6);

способность самостоятельно и (или) в составе исследовательской группы разрабатывать, исследовать и применять математические модели для качественного и количественного описания явлений и процессов и (или) разработки новых технических средств (ПК-1);

способность применять на практике умения и навыки в организации исследовательских и проектных работ, способностью самостоятельно организовывать и проводить научные исследования и внедрять их результаты в качестве члена или руководителя малого коллектива (ПК-3);

способность профессионально работать с исследовательским и испытательным оборудованием, приборами и установками в избранной предметной области в соответствии с целями программы специализированной подготовки магистра (ПК-4).

В результате освоения дисциплины обучающиеся должны

знать:

Принципы внутреннего устройства ядра операционной системы. Механизмы аппаратной поддержки работы ядра. Механизмы обеспечения защиты ядра операционной системы от приложений и приложений друг от друга. Методы управления и распределения аппаратными ресурсам. Методы и средства виртуализации аппаратных ресурсов.

уметь:

Проектировать компоненты ядра операционной системы. Программировать на языке Си и на языке ассемблера с использованием привилегированных инструкций процессора. Отлаживать программы, работающие в привилегированном режиме работы процессора.

владеть:

Технологиями разработки компонентов ядра операционной системы.

4. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием отведенного на них количества академических часов и видов учебных занятий

4.1. Разделы дисциплины (модуля) и трудоемкости по видам учебных занятий

№	Тема (раздел) дисциплины	Виды учебных занятий, включая самостоятельную работу				
		Лекции	Практич. (семинар.) задания	Лаборат. работы	Задания, курсовые работы	Самост. работа
1	Введение. Устройство ядра JOS.			5		1
2	Описатели процессов в JOS. Прерывания в x86. Инициализация IDT.			5		1
3	Обработка вложенных прерываний в x86.			5		1
4	Переключение между режимами работы процессора.			5		1
5	Примитивная файловая система			5		1
6	Комплексное практическое задание			5		1
Итого часов				30		6
Подготовка к экзамену		0 час.				
Общая трудоёмкость		36 час., 1 зач.ед.				

4.2. Содержание дисциплины (модуля), структурированное по темам (разделам)

Семестр: 1 (Осенний)

1. Введение. Устройство ядра JOS.

Введение. Карта физической памяти x86. Процесс загрузки и инициализации PC.

BIOS, инициализация основных устройств. Загрузчик JOS. Загрузка ядра.

Устройство ядра JOS. Отладка кода ядра JOS. Компиляция первой собственной функции, вывод строк на консоль.

2. Описатели процессов в JOS. Прерывания в x86. Инициализация IDT.

Описатели процессов в JOS. Создание процессов в JOS, загрузка приложений в память из бинарных секций образа ядра. Переключение контекстов. Кооперативное разделение времени. Примитивный планировщик FIFO без приоритетов.

Прерывания в x86. Инициализация IDT. Обработка прерываний таймера. Вытесняющее разделение времени. Примитивный планировщик Round Robin без приоритетов.

3. Обработка вложенных прерываний в x86.

Обработка вложенных прерываний в x86. Средства синхронизации, состояние гонок, деадлоки. Запрет прерываний, семафоры.

Управление распределением физических страниц. Виртуальная память. Сегментная и страничная трансляция. Таблицы трансляции.

4. Переключение между режимами работы процессора.

Переключение между режимами работы процессора. Прерывания и системные вызовы. Вложенные прерывания. Изменения в создании процессов, переключении между контекстами. Передача данных между программой и ядром.

Управление процессами. Системный вызов fork(). Механизмы межпроцессного взаимодействия.

5. Примитивная файловая система

Примитивная файловая система. Реализация системных вызовов open(), close(), read(), write(), exec().

Механизмы и виды виртуализации. Аппаратная поддержка виртуализации.

6. Комплексное практическое задание

Комплексное практическое задание

5. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Используются компьютер и мультимедийный проектор для демонстрации слайдов в ходе лекций. Для самостоятельной работы студенты используют компьютеры в машинных залах и/или личные компьютеры.

6. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины (модуля)

Основная литература

1. Э. Таненбаум "Архитектура компьютера", Питер, 2012 г.
2. Э. Таненбаум "Современные операционные системы", Питер, 2011 г.
3. Э. Таненбаум "Операционные системы. Разработка и реализация", Питер, 2006 г.
4. А. Робачевский, С. Немнюгин, О. Стесик «Операционная система UNIX», БХВ Санкт-Петербург, 2010 г.

Дополнительная литература

1. C. A. R. Hoare, Monitors: An Operating System Structuring Concept. Communications of the ACM, 17(10):549--557, 1974.
2. Thomas W. Doeppner "Operating Systems In Depth: Design and Programming" Wiley, 2010
3. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. "Linux Device Drivers".
4. Ellen Siever, Stephen Figgins, Robert Love, Arnold Robbins. "Linux in a Nutshell". O'Relly Media, 2009.
5. Скотт Шакон. «Про Git». Apress. Перевод на русский. (http://git-scm.com/book/ru). в) программное обеспечение и Интернет-ресурсы

Программное обеспечение и Интернет-ресурсы

1. Веб-страница дисциплины: http://forge.ispras.ru/projects/os-course-YEAR/wiki
2. IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture. Basic 80x86 architecture and programming environment. http://developer.intel.com.
3. IA-32 Intel Architecture Software Developer's Manual Volume 3A: System Programming Guide. Operating system support, including segmentation, paging, tasks, interrupt and exception handling. http://developer.intel.com.
4. Tool Interface Standard Executable and Linking Format (ELF) Specification Version 1.2.
5. Свободно распространяемый эмулятор x86 компьютеров. (http://qemu.org)

7. Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)

http://forge.ispras.ru/projects/os-course-YEAR/wiki

8. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем (при необходимости)

http://forge.ispras.ru/projects/os-course-YEAR/wiki

9. Методические указания для обучающихся по освоению дисциплины

Студент, изучающий дисциплину, должен с одной стороны, овладеть общим понятийным аппаратом, а с другой стороны, должен научиться применять теоретические знания на практике.

В результате изучения дисциплины студент должен знать основные определения, понятия, аксиомы, методы доказательств.

Успешное освоение курса требует напряжённой самостоятельной работы студента. В программе курса приведено минимально необходимое время для работы студента над темой. Самостоятельная работа включает в себя:

- чтение и конспектирование рекомендованной литературы;
- проработку учебного материала (по конспектам лекций, учебной и научной литературе), подготовку ответов на вопросы, предназначенных для самостоятельного изучения, доказательство отдельных утверждений, свойств;
- выполнение лабораторных работ, для осознание связей между теорией и практическими навыками;
- подготовку к дифференцированному зачету.

Руководство и контроль за самостоятельной работой студента осуществляется в форме индивидуальных консультаций.

Важно добиться понимания изучаемого материала, а не механического его запоминания. При затруднении изучения отдельных тем, вопросов, следует обращаться за консультациями к лектору.

В рамках курса слушателям предлагается выполнить 11 лабораторных работ. Они выполняются студентами в рамках практикума и самостоятельной работы.

Лабораторная работа №1. Процесс загрузки ЭВМ.
Лабораторная работа №2. Отладка кода ядра ОС в эмуляторе ЭВМ Qemu.
Лабораторная работа №3. Кооперативное разделение времени.
Лабораторная работа №4. Прерывания и вытесняющее разделение времени.
Лабораторная работа №5. Механизмы синхронизации.
Лабораторная работа №6. Управление распределением физических страниц.
Лабораторная работа №7. Виртуальная память.
Лабораторная работа №8. Системные вызовы.
Лабораторная работа №9. Системный вызов fork() и межпроцессное взаимодействие.
Лабораторная работа №10. Файловые системы.

Пример учебно-методического обеспечения для лабораторной работы

Для получения файлов, необходимых для выполнения лабораторной работы №3, выполните следующие действия в директории с исходным кодом ядра JOS:

$ git pull origin

$ git checkout lab3

$ git checkout -b working-lab3 working-lab2

$ git merge lab3

При возникновении конфликтов (случаев, когда при выполнении предыдущих работ вы изменили файлы, изменившиеся в ветке lab3) команда git merge сообщит, какие файлы находятся в состоянии конфликта, и перед выполнением работы вам потребуется избавиться от конфликта, отредактировав эти файлы, и выполнить git commit -a.

В этой лабораторной работе вы будете реализовывать основные возможности ядра, необходимые для запуска процессов ("окружений") в режиме ядра без виртуальной памяти. Вы добавите создание в ядре JOS структур для отслеживания окружений, создания окружений, загрузки образа программы в окружение и его запуска. Окружения на текуем этапе будут функционировать в режиме ядра(ring 0). В некоторых местах код защищен макросом CONFIG_KSPACE. Это необходимо для того, чтобы в дальнейшем, когда окружения будут работать в кольце 3, этот специфичный код не использовался или использовалась его альтернатива.

Примечание: В этой лабораторной работе термины "окружение" и "процесс" взаимозаменяемы - они имеют примерно один и тот же смысл. Термин "окружение" вместо традиционного термина "процесс" вводится для того, чтобы подчеркнуть, что окружения JOS не обеспечивают ту же семантику, что процессы UNIX, хотя и сопоставимы.

В работе содержатся новые файлы, рассмотрите их:

inc/ env.h Публичные определения для пользовательских окружений
lib.h Публичные определения для библиотеки
kern/ env.h Внутренние определения для окружений
env.c Реализация окружений
trap.h Внутренние определения для обработки исключений
lib/ Makefrag Фрагмент makefile для сборки библиотеки пользовательского режима obj/lib/libuser.a
entry.S Точка входа для пользовательских окружений на языке ассемблера
libmain.c Код установки библиотеки пользовательского режима, вызываемый из entry.S
prog/ * Различные тестовые программы для проверки кода работы №3
doc/ ELF_Format.pdf Спецификация Elf формата. Понадобится для выполнения последнего задания.

Ассемблерные вставки

В этой работе вы можете найти полезной возможность GCC встраивать в программы фрагменты на языке ассемблера, хотя работа может быть выполнена и без их использования. Как минимум, вы должны быть в состоянии понять подобные фрагменты (оператор asm), которые уже существуют в исходном коде.

Пользовательские окружения и обработка исключений

Новый заголовочный файл inc/env.h содержит основные определения для пользовательских окружений в JOS. Просмотрите это сейчас. Ядро использует структуру Env для отслеживания каждого пользовательского окружения. В этой работе вы изначально создадите только одно окружение, но вам необходимо будет внести поддержку нескольких окружений в ядро JOS. Как вы можете видеть в kern/env.c, ядро поддерживает три основных глобальных переменных, относящихся к окружениям:

```
struct Env* envs; // Все окружения
struct Env * curenv = NULL; // Текущее окружение
```

```
static struct Env *env_free_list; // Список свободных окружений
```

После того, как JOS запущена, указатель envs указывает на массив структур Env, соответствующих всем окружениям в системе. Ядро JOS будет поддерживать максимум NENV одновременно активных окружений, хотя, как правило, обычно это количество будет гораздо меньше. (NENV - константа, определенная в inc/env.h.) После того, как массив envs выделен, он будет содержать один экземпляр структуры Env для каждого из NENV возможных окружений. Ядро JOS содержит все неактивные структуры Env в env_free_list. Такая конструкция дает возможность легкого выделения и освобождения окружений, так как они просто должны быть добавлены или удалены из списка свободных.

Ядро также использует curenv для отслеживания выполняемого в данный момент окружений. Во время загрузки, до первого запуска окружения, curenv равен NULL. Структура Env определена в inc/env.h следующим образом:

```
struct Env {
        struct Trapframe env_tf;
        struct Env *env_link;
        envid_t env_id;
        envid_t env_parent_id;
        enum EnvType env_type;
        unsigned env_status;
        uint32_t env_runs;
    };
```

Рассмотрим подробнее поля этой структуры:

- env_tf:

Эта структура, определенная в inc/trap.h, содержит сохраненные значения регистров для окружения в момент, когда окружение не работает, то есть когда работает ядро или другое окружение. Ядро сохраняет эти значения при переключении окружений, чтобы работа окружения в дальнейшем могла быть возобновлена в том же месте, где она была прервана.

- env_link:

Ссылка на следующий Env в env_free_list. Сама переменная env_free_list указывает на первое свободное окружение в списке.

- env_id:

Ядро хранит здесь значение, которое однозначно идентифицирует окружение, которое в настоящее время использует данную структуру Env (то есть использует данный слот в массиве envs). После завершения окружения ядро может использовать ту же структуру Env для нового окружения, но новое окружение будет иметь другое значение env_id.

- env_parent_id:

Ядро хранит здесь env_id окружения, которое создало данное. Таким образом, окружения формируют "родословную", которая может быть полезна для принятия решений о том, какие окружения какие действия могут производить.

- env_type:

Это поле используется, чтобы отличить особые окружения. Сейчас это поле может принимать только одно значение ENV_TYPE_KERNEL. Будет введено еще несколько типов окружений в дальнейших работах.

- env_status:

Эта переменная содержит одно из следующих значений:

- ENV_FREE: Указывает, что структура неактивна и, следовательно, находится в env_free_list.
- ENV_RUNNABLE: Указывает, что структура соответствует окружению, которое ждет возможности работать на процессоре.
- ENV_RUNNING: Указывает, что структура соответствует окружению, работающему в данный момент.
- ENV_NOT_RUNNABLE: Указывает, что структура соответствует окружению, которое активно, но в настоящее время не готово к работе: например, ожидает межпроцессного взаимодействия (IPC) с другим окружением.
- ENV_DYING: Указывает, что структура соответствует зомби-окружению. Зомби-окружение будет освобождено в следующий раз, когда оно вызовет исключение ядра. Мы не будем использовать это значение в данной работе.

Как и Unix-процесс, окружение JOS связывает понятия "поток" и "адресное пространство". Поток определяется прежде всего сохранёнными регистрами (поле env_tf), а адресное пространство в отсутствие виртуальной памяти закрепляется за каждым процессом индивидуальной областью памяти. Области памяти разных процессов не должны пересекаться. Сами адреса, которые закреплены за процессом задаются на этапе линковки. Чтобы запустить окружение, ядро должно установить в процессоре сохранённые регистры.

Создание и запуск окружения

Теперь Вы будете писать код в kern/env.c, необходимый для запуска пользовательского окружения. Так как мы еще не имеем файловой системы, ядро будет загружать статический двоичный образ программы, который встроен в само ядро. JOS встраивает его в ядро как исполняемый ELF-образ.

GNUmakefile данной работы создает ряд двоичных образов в obj/prog/. Если вы посмотрите на kern/Makefrag, вы заметите немного "магии", которая компонует эти файлы непосредственно в ядро,

как если бы они были файлами .o. Параметр командной строки компоновщика -b binary заставляет его компоновать эти файлы как "сырые" неинтерпретированные двоичные файлы. (С точки зрения компоновщика, эти файлы могут вообще не быть ELF-образами - они могут быть любыми, например, текстовыми файлами или фотографиями) Если вы посмотрите на файл obj/kern/kernel.sym после сборки ядра, вы заметите, что компоновщик "волшебным образом" создал ряд символов с непонятными названиями, такими как _binary_obj_prog_test1_start, _binary_obj_prog_test1_end и _binary_obj_prog_test1_size. Компоновщик генерирует эти названия символов из названий двоичных файлов; символы обеспечивают в обычном коде ядра возможность ссылаться на эти встроенные двоичные файлы.

В функции i386_init() в kern/init.c вы увидите код для запуска этих двоичных образов в качестве окружения. Тем не менее, критически важные функции настройки пользовательских окружений не завершены, вы должны написать их самостоятельно.

Упражнение 1. В файле env.c допишите следующие функции:

- env_init()

Инициализирует все структуры Env в массиве envs и добавляет их в env_free_list. Также вызывает env_init_percpu, которая настраивает оборудование для сегментации с отдельными сегментами для уровня привилегий 0 (ядро) и уровня привилегий 3 (пользователь).

- load_icode()

Вам необходимо декодировать двоичный ELF-образ, так же, как это уже делает загрузчик, и загрузить его содержимое в адресное пространство нового окружения. Пока не следует обращать внимание на функцию bindfunctions.

- env_create()

Выделяет окружение с помощью envalloc и загружает в него двоичный ELF-образ путем вызова load_icode.

- env_run()

Запускает данное окружение.

В этих функциях вам может быть полезной новая строка форматирования cprintf %e - она выводит описание, соответствующее коду ошибки. Например,

r = -E_NO_MEM;

panic("env_alloc: %e", r);

будет останавливать работу ядра с сообщением ошибки "env_alloc: out of memory".

Ниже приведен граф вызовов кода до точки, в которой запускается код окружения. Убедитесь, что вы понимаете цели каждого шага.

- start (kern/entry.S)
- i386_init (kern/init.c)
- cons_init
- env_init
- env_create
- sched_yield
- env_run
- env_pop_tf

Как только вы закончите, вы должны скомпилировать ядро и запустить его под QEMU. Если все пойдёт хорошо, ваша система должна дойти до функции env_pop_tf и упасть на первой её инструкции push. Эта функция осуществляет загрузку регистров приложения на процессор и передачу программе управления через инструкцию ret. Последняя требует наличия адреса перехода на стеке. Так как к моменту его загрузки на стек остальные регистры окружения уже загружены, то используется стек окружения, а не ядра, который ещё не был задан (упражнение 2). Из-за этого будет сгенерировано исключение общей защиты. Ядро обнаружит, что его оно также не может обработать, что создаёт двойное исключение, которое на данном этапе развития системы также не обрабатывается. Наконец, процессор генерирует то, что известно как "тройная ошибка" (triple fault). Обычно в таких случаях процессор сбрасывается и система перезагружается. Это усложняет разработку ядра, поэтому с модифицированной версией QEMU вы вместо этого будете видеть дамп регистров и сообщение "Triple fault". Чтобы временно обойти эту проблему раскомментируйте строку // e->env_tf.tf_esp = 0xf0210000; из упражнения 2.

Сейчас мы можем использовать отладчик, чтобы убедиться, что осуществляется передача управления окружению. Используйте make qemu-gdb и установите точку прерывания в env_pop_tf, которая должна быть последней функцией до передачи управления процессу. Пройдите эту функцию с помощью команды si; инструкция ret передаёт управление окружению. После этого вы должны увидеть первую инструкцию в исполняемом окружении -инструкцию push $0 на метке start в lib/entry.S. Если вы не можете выполнить код до этого места, что-то не так с вашей настройкой окружений или с кодом загрузки программы; вернитесь и исправьте эту проблему, прежде чем продолжать. Если же ошибка происходит в функции sched_yield на вызове env_run, то либо выполните сначала упражнение №3, либо не выполняя его, замените вызов env_run(envs) на env_run(&envs[нужный элемент массива]). Это зависит от вашей реализации env_init. Функция env_init должна быть написана так, чтобы первый вызов env_alloc возвращал envs[0].

Упражнение 2.

В функции env_alloc (kern/env.c) содержатся строки:

// LAB 3: your code here

// e->env_tf.tf_esp = 0xf0210000;

Здесь задаётся адрес стека для процессов. У каждого процесса этот адрес должен быть свой. Необходимо это учитывать. Раскомментируйте эту строку и сделайте так, чтобы при каждом вызове env_alloc задавался уникальный адрес стека.

Примечание: Не имеет смысла выделять под стек большие объёмы памяти. Как правило, должно хватать двух страничных кадров.

После выполнения этого упражнения программы должны выполняться до того места, где пытаются вызвать функции ядра (cprintf, sys_yield, sys_exit). Они делают это через свои глобальные переменные, которые сейчас ещё не проинициализированы и равны нулю. Это проблема решается выполнением упражнения №5.

Упражнение 3.

В функции sched_yield (kern/sched.c) должен быть реализован алгоритм планировщика. В данном случае используется простейший алгоритм Round-robin. Необходимо чтобы учитывалось как состояние процессов ENV_RUNNABLE, так и и ENV_RUNNING. Исходите из того, что в системе JOS не поддерживается процессорной многоядерности. Обратите внимание на комментарии в функции sched_yield.

Упражнение 4.

В файле kern/entry.S есть реализация функции sys_yield. Эту функцию процессы вызывают, когда хотят добровольно отдать управление ядру. После происходит сохранение контекста процесса, переключение на контекст ядра и передача управления планировщику. Рядом с функцией sys_yield есть функция sys_exit. Последняя вызывается, когда процессы завершают свою работу. По аналогии с sys_yield допишите sys_exit так, чтобы её работа осуществлялась верным образом.

Примечание: в sys_exit сохранение контекста процесса, отдающего управление, не требуется. Упражнение 5.

В функции load_icode (kern/env.c) необходимо реализовать загрузку образов окружений по необходимым им адресам. К текущему моменту эта задача уже должна быть решена. После необходимо реализовать связывание(функция bind_functions) для загружаемых программ, чтобы они могли пользоваться некоторыми функциями ядра. Обратите внимание на закомментированные строки:

```
*((int *) 0xf0302008) = (int) &cprintf;
*((int *) 0xf0302010) = (int) &sys_exit;
*((int *) 0xf0302004) = (int) &sys_yield;
*((int *) 0xf020200c) = (int) &sys_exit;
*((int *) 0xf0202004) = (int) &sys_yield;
```

По сути, всё сводится именно к этому. Слева стоят адреса глобальных переменных программ, которым присваиваются адреса функции ядра. Вам необходимо сначала получить адреса глобальных переменных с конкретными названиями. Сейчас они получены из файлов obj/prog/test1.sym и obj/prog/test2.sym, и могут не совпадать с теми, которые будут у вас на машине. Такой способ извлечения данных с последующим ручным заполнением этих адресов необходимыми значениями адресов функций ядра не является достаточно удобным, чтобы его можно было применять широко. Вам необходимо в функции bindfunctions получать адреса глобальных переменных (подобно тому, как это делают программы nm -n, objdump -x), и каждой глобальной переменной, чьё имя совпадает с именем одной из функций ядра присваивать адрес последней. Адреса функций ядра должны быть получены из отладочной информации, с которой вы уже научились работать в прошлой рабораторной работе. Для этого необходимо реализовать функцию findfunction (kern/kdebug.c). Строковые имена функций хранятся в stabstr, их адреса в stab (поле nvalue).

Примечание: обратите внимание на функции в файле lib/string.c, на раздел Symbol Table в спецификации Elf формата.

После реализации всех заданий, при выполнении make qemu на экран должна выводиться
примерно следующая информация:

```
[00000000] new env 00001000
[00000000] new env 00001001
envrun RUNNABLE: 1
HERE
envrun RUNNABLE: 0
envrun  RUNNABLE:  1
envrun  RUNNABLE:  0
envrun  RUNNABLE:  1
envrun  RUNNABLE:  0
envrun  RUNNABLE:  1
envrun RUNNABLE: 0
[00001000] free env 00001000
envrun RUNNABLE: 1
envrun RUNNING: 1 [00001001]
free env 00001001
```

10. Фонд оценочных средств для проведения промежуточной аттестации по итогам обучения

Приложение

ПРИЛОЖЕНИЕ

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ОБУЧАЮЩИХСЯ ПО ДИСЦИПЛИНЕ

по направлению: Прикладные математика и физика (магистратура)
профиль подготовки: Системное программирование
факультет: управления и прикладной математики
кафедра (название): системного программирования
курс: 1
квалификация: магистр

Семестр, формы промежуточной аттестации: 1(Осенний) - Дифференцированный зачет

Разработчик: А.В. Хорошилов, к.ф.-м.н.

1. Компетенции, формируемые в процессе изучения дисциплины

Освоение дисциплины направлено на формирование у обучающегося следующих общекультурных (ОК), общепрофессиональных (ОПК) и профессиональных (ПК) компетенций:

способность применять современные методы анализа, обработки и представления информации в сфере профессиональной деятельности (ОПК-5);

способность осуществлять научный поиск и разработку новых перспективных подходов и методов к решению профессиональных задач, способностью к профессиональному росту (ОПК-6);

способность самостоятельно и (или) в составе исследовательской группы разрабатывать, исследовать и применять математические модели для качественного и количественного описания явлений и процессов и (или) разработки новых технических средств (ПК-1); способность применять на практике умения и навыки в организации исследовательских и проектных работ, способностью самостоятельно организовывать и проводить научные исследования и внедрять их результаты в качестве члена или руководителя малого коллектива (ПК-3);

способность профессионально работать с исследовательским и испытательным оборудованием, приборами и установками в избранной предметной области в соответствии с целями программы специализированной подготовки магистра (ПК-4).

2. Показатели оценивания компетенций

В результате изучения дисциплины «Конструирование ядра операционной системы» обучающийся должен:

знать:

Принципы внутреннего устройства ядра операционной системы. Механизмы аппаратной поддержки работы ядра. Механизмы обеспечения защиты ядра операционной системы от приложений и приложений друг от друга. Методы управления и распределения аппаратными ресурсам. Методы и средства виртуализации аппаратных ресурсов.

уметь:

Проектировать компоненты ядра операционной системы. Программировать на языке Си и на языке ассемблера с использованием привилегированных инструкций процессора. Отлаживать программы, работающие в привилегированном режиме работы процессора.

владеть:

Технологиями разработки компонентов ядра операционной системы.

3. Перечень типовых контрольных заданий, используемых для оценки знаний, умений, навыков

Промежуточная аттестация по дисциплине «Конструирование ядра операционной системы» осуществляется в форме экзамена (зачета). Экзамен (зачет) проводится в письменной (устной) форме.

1. Процесс загрузки и инициализации персональных ЭВМ архитектуры x86. Задачи, решаемые BIOS, инициализация основных устройств. Функции загрузчика. Загрузка ядра.
2. Виды архитектур ядра ОС. Монолитные и микроядерные архитектуры.
3. Переключение контекстов. Кооперативное разделение времени. Примитивный планировщик FIFO без приоритетов.
4. Прерывания в x86. Инициализация IDT.
5. Обработка прерываний таймера. Вытесняющее разделение времени. Примитивный планировщик Round Robin без приоритетов. Алгоритмы планирования процессорного времени.
6. Обработка вложенных прерываний в x86. Средства синхронизации, состояние гонок, деадлоки. Запрет прерываний, спинлоки, мьютексы, семафоры. Read-Copy-Update.
7. Карта физической памяти x86. Управление распределением физических страниц.
8. Виртуальная память. Сегментная и страничная трансляция x86. Таблицы трансляции.
9. Переключение между режимами работы процессора. Прерывания и системные вызовы. AMD syscall/sysreturn и Intel sysenter/sysexit. Выполнение системных вызовов без переключения в привилегированный режим.
10. Управление процессами. Системные вызовы fork() и exec(). Механизмы межпроцессного взаимодействия.
11. Механизмы межпроцессного взаимодействия в ОС Linux (сигналы, разделяемая память, семафоры, очереди сообщений, программные каналы, сетевые интерфейсы, взаимодействие на основе файловых систем, KDBUS).
12. Файловые системы. Основные задачи файловых систем. Организации работы файловых систем в ОС Linux: VFS, драйвера файловых систем, bio, кэширование. Обзор современных файловых систем ОС Linux (ext4, btrfs, XFS, jffs2, f2fs).
13. Механизмы и виды виртуализации. Аппаратная поддержка виртуализации (Intel VT-x и AMD AMD-V). Вложенные таблицы трансляции и идентификаторы виртуальных процессоров

4. Критерии оценивания

Оценка по курсу устанавливается в зависимости от суммы технических баллов, набранных студентом в ходе семестра. Каждая лабораторная работа оценивается от 0 до 2 технических баллов. Общая сумма по десяти лабораторным работам может составить от 0 до 20 технических баллов. За выполнение

комплексного индивидуального задания студенту может быть начислено от 0 до 80 технических баллов. За выполнение итоговой письменной (экзаменационной) работы студент может получить от 0 до 50 технических баллов. Таким образом, максимально возможная сумма набранных технических баллов составляет 150. Оценка «отлично» ставится студентам, набравшим от 100 баллов и выше. Оценка «хорошо» ставится студентам, набравшим от 70 до 99 технических баллов. Оценка «удовлетворительно» ставится студентам, набравшим от 40 до 69 технических баллов. Оценка «неудовлетворительно» ставится студентам, набравшим менее 40 технических баллов.

5. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности

Во время проведения зачета обучающиеся могут пользоваться программой дисциплины, а также справочной литературой, вычислительной техникой, конспектами лекций. Дифференцированный зачет может проводиться по итогам текущей успеваемости и сдачи заданий, или путем организации специального опроса, проводимого в устной форме.