Радетар – оч. большой файл м. б.

- Здесь речь о 64 разряд. значениях (8 байтах)
- Это несколько коррелирует с дескрипторами страницы.
- Здесь (в радетар?): – если стр-ца находится в физ. памяти, то 63й бит д.б. установлен
  - если стр-ца находится в области свопинга, то 62й бит установлен
- shared – разделяемые стр-цы

(Это было в 4х уров. таблицах)

pte – page table entry. Каждый pte – это дескриптор страницы

- 55й бит – pte ?
- от 0 до 54 разряда – это адрес, если стр-ца находится в памяти, pageframe number
- Если страница не находится в памяти, а находится в обл-ти свопинга, то с 0 по 4 бит отражает swap type

Изменим инф-цию в файле ядра. У нас есть чрс, сокеты, можно взять однопоточного клиента

Семинар 26.03

Загружаемые модули ядра позволяют работать в режиме ядра, с кодом ядра. Затмев- непосредственно

Unix/Linux/Windows — монолит. ядро

· Монолитное ядро — единая программа, в кот. определены структуры и ф-ции (ф-ции ядра). В монолит. ядре взаимодействуют части кода, направленные на определ. задачи (напр. управление памятью).

· Все подсистемы под ядра многоуровневые (от близости к аппаратной составляющей)

· Современные подходы потребовали от разработчиков предоставление пользователям возможностей включение в код ядра нужных пользователю функц.

16:13

В Windows это многоуровневые драйвера. Пользователь может свой драйвер написать.

Этот драйвер м.б. зарегистрирован в системе.

MSDN - здесь видны все зарегистр-ые драйвера.

В Unix/Linux это загружаемые модули ядра. Система предоставляет перекомпилировать ядро (это сложный шаг)
все может рухнуть

Вместо перекомпиляции можно написать свои загр. мод. ядра.

Система предоставляет ф-ции, позволяющие загружать/выгружать модуль из ядра

Код, помещённый в ядро, становится частью кода ядра
→ На этот код распространяются все ограничения ядра

• Мы будем писать загруж. модуль ядра для получения инф-ции о процессе, работать со структурой ядра task struct.

• У загруж. модуля ядра д.б. 2 обязательн. т. входа:

module_init
module_exit – это макросы, которым передаются написанные нами ф-ции init и exit

У драйвера больше точек входа, чем у загруж. мод. ядра

• Точки входа ОС – сис. вызовы, исключения, аппарат. прерывания

команды – тоже т. входа, но опосредованно
ps -ajx – инф-ция берется из стр. ядра

• У ФС много точек входа
read/write() – тоже т. входа

```
static int __init my_init(void)
{
    printk("My module is load\n");
    return 0;
}

static void __exit my_exit(void)
{
    printk("My module exit\n");
}

module_init(my_init);
module_exit(my_exit);
```

двойное подчёркивание – указание, что на этот код система должна обратить внимание

Наш модуль

ф-я ядра (printk)

передаём свои ф-ции

• В ядре библиотеке ядра содерж. ф-ии и ф-ции ядра. функц. ядра

• printk() не входит в какую-либо библиотеку. В нем надо писать уровень протоколир-е (+)

• В ядре определено 8 уровней протоколирования. — это библиотеч. значения
чем меньше значение, тем выше уровень : это библиотеч. значение

Библиотека ядра: → Напр. KERN_INFO информационные сообщения (в методичке)

#include <linux/module.h>

#include <linux/kernel.h> уровни протоколирование

(код ядра был полностью переписан, когда появилась smp-архит-ра)

• Smp – архитектура, определяющая равноправные процессоры, кот. работают с общей памятью. Условно ∃ 1 главный пр-сор, в задачи кот. входит обработка прерываний от сист. таймера. (Иначе одни и те же действия будут 4 раза выполнены (4 процессора) – нет смысла, только вред)

У каждого пр-сора есть опред. объем физ. памяти, к кот. он имеет доступ.

из struct task_struct:

#ifdef CONFIG_NUMA – Non Uniform Memory Access – координированный доступ к памяти

struct task_struct
struct thread_info thread_info;
⋮
randomized_struct... *stack;
sched_entry, ... – планирование

{это потребовало включения в task_struct соотв-х полей}

{Адрес стека в stat и в maps не совпадает т.е. адрес стека в stat находится в диапазоне (это не начальн. адрес диапазона}

(поток – единица диспетчеризации) – пр-сс должен иметь указатель на вершину стека

unsigned int policy – алгоритм планир-я: увидим 3 значения (FIFO, RR; алгоритм планир-я, установ. в системе по умолч.)

• В ядре нет отдельного модуля, который бы собирал всю инф-цию о загруженности ядер и распределял работу м/у ядрами

• Для каждого ядра определен демон migration, кот. анализирует кол-во пр-ссов, которые стоят в очереди к конкрет. ядру и в зависимости от функция значения перемещает пр-сы на другое ядро.

• struct list_head tasks; – определяет двусвязный список процессов

! двусвязный – для ускорение поиска (можно в 2 стороны ходить)

• struct mm_struct *mm; – структура, описывающ. вирт. адрес. простр-во процесса (у пр-сса нет памяти, у него вирт. АП)

/proc: регионы выводим для кода и сегментов данных

4х уровневую указатель на таблицу страниц (Это адрес таблицу страниц процесса, который загружается в регистр CR3.)

-страницы заполнены не полностью – скрытая фрагментация

• state тоже выводим: 3 поле
→ exit_state, exit_code, exit_signal

• maps – вирт. АП

• comm – имя файла (до того как стать выполняется пр-сс это программа а у нее есть имя)

• struct list_head children; в системе существует дерево процессов (связи иерархии в отношении "предок-потомок")

• struct fs_struct *fs; // инф-ция об ФС, кот. ∈ процесс

• struct files_struct *files;

∀ пр-сс имеет собств. таблицу открытых файлов

• В ядре ∃ только 1 ф-ция exec и это execve().

• Только пр-сс может открыть файл в системе

• Процесс может максимум открыть 512 файлов – чтобы разделить данные по назначению.

PSI – это шина

Пр: если lstat == -1, то ф-цию вызывать не надо!

Просто проверяет, ∃ ли этот файл ст. зрения ∀ типа файла.

lstat в цикле – бесконечно

По task_struct : надо flags

1 стр = 4096 байт

в stat st_blksize — падает на heap. Перевести 10ое представление в 16ое

• Макс. адрес м.б. получен в системе. Но это не значит, что все эти адреса будут использованы.

Семинар 03.04.

Взаимодействие загруж. модулей ядра

Статьи Олег Цилюрик.

[illegible]

Пильщиков в книге рассматривает многомодульн. программы (Ассемблер)

Как могут взаимодейств-ть модули программы.

Ответ. — Один модуль может обращаться к данным другого модуля.

для этого необходимо использ-ть **модификатор extern**

• Необходимо в модуле, кот. обращается к данным, писать extern. Extern нужен чтобы прошла компиляция модуля.

и public в модуле, в котором эти данные объявлены и проинициализированы

public – общедоступный т.е. в модуле данные определены как общедоступные (доступные ф-ям модулей)

• Extern – нужен, чтобы прошла компиляция модуле. Компилятору указывается, что данное (либо переменная, ф-ция) объявлено как внешнее (где-то вне)

• Листинг 1 (mb1) Вызываемый модуль – модуль, в котором объявлены и проинициализированы данные, к которым обращается другой модуль.

header – mb.h – в нем переменная mb1 data и ф-ция определены как extern

mb1_local () – локальная, только в данном модуле

mb1_noexport – не экспортируемая

• Макрос EXPORT_SYMBOL (имя символа) – этому макросу передается имя перем-ой/ф-ции, к которой обращается другой модуль

• Листинг 2 (mb2)

mb3 – копия mb2, но init возвр. -1 – не препятствует компиляции

mb1.c mb2.c mb3.c – будут откомпилированы, получим исполняемые файлы.

Модуль mb2, кот. обращается к данным mb1 по экспортируемому имени, должен получить абсолют. адрес.

Абсолютный адрес – это физический адрес.

mb2 должен получить физич. адрес данных, описанных в mb1.

Данные получат адреса после загрузки

потому что указывает на конкретное место в

(это прямой адрес)

- Только после загрузки модуля во оперативн. память данные загружаются в соотв. адреса и после по этим адресам можно и данным обратиться
Поэтому имеет значение последов-ть загрузки и выгрузки модулей.

• Модуль можно выгрузить если на него не осталось ни 1 ссылки.

• Х загружена в физич. память. Модуль после загрузки становится частью ОС и этот загружаемый модуль оперирует физ. адресами

• У Х адрес простр-во делится на стр-цы и ~~[illegible]~~

•

2ая часть.

Защита Лр. — историю ошибок продемонстрировать. Чтобы проще находить строки их предваряем „+".

След. Лр:

Передача данных из ядра в простр-во пользов-ля — 2 способа

Потом пишем свою ОС

Потом пишем собств. обработчик прерыв-е + отчет действия

task-struct

Чтобы получить доп. инфу: для 2-3 потоков ядра расписать state и flags. •с 98 строки-state (elixir)

• Потоки требуют создания региона для каждого потока в адрес. простр-ве процесса — это затрата.

что формулировка не точная смысл

процессы интересно смотреть у них:
· размещение по policy
· prio < 120
· FIFO ..?

мод. загр. модули ядра

• при fork не выполняется exec

кворкер - чей поток? - поток ядра (выполняет код ядра).

• Код ядра нельзя заменить другим кодом, поэтому fork по exec

флаг →

• no freeze — потоки не могут блокироваться

• state у кворкера 1026 (у Чаня) - надо расписать

policy = 1 - FIFO - выполнение ~~обслуживание~~ от начала до конца без переключений

Доступно = адресуемо

policy = 2 - RR

стек ↑ регистр