

## Зеркальное производство - mirror

page

map - отображение каждой вирт. страницы процесса на физ. адрес или в обл. мб свопинга;

В UNIX/Linux потоки отн. малые;

Windows заточены на потоки,  
а процесс можно создать только  
из командной строки

Qt создаёт для себя 5 доп. потоков;  
(не хорошо, т.к. потоки требуют ресурсы,  
в первую очередь память);

еро// для ускорения обработки  
судимений (мультитаскирование  
высокой точности);

- root  
/proc/[pid]/root

root - soft link - указывает на корень  
ф.с. процесса;

- stat - файл - информация о процессе;

Записывать info о своих процессах  
в файл и показывать на лабе;

envIRON:

```

#include <stdio.h>
#define BUF_SIZE 0x100
int main (int args, char * argv[])
{
    char buf[BUF_SIZE];
    int len, i;
    FILE * f;
    f = open("/proc/setf/environ", "r");
    while ((len = fread(buf, 1, BUF_SIZE, f)) > 0)
    {
        for (i = 0; i < len; i++)
            if (buf[i] == 0)
                buf[i] = 10; // for 0x0A
        buf[len] = 0;
        printf("%s", buf);
    }
    fclose(f);
    return 0;
}

```

строки в файле  
environ разделены  
не \n, а \0; (\n = 10 = 0x0A)

maps можно открыть cat

можно сравнить 2 программы, скомпи-  
лированные gcc и Qt;

Отзывчивость программы очень важна  
→ реакция на интерактивные события  
#  
реальное время

↳ ан. опр-е реального времени  
POSIX (было ранее):

Работа системы таким образом, что она обрабатывает запросы внешних по отношению к системе процессов (или др. систем) за опред. интервал времени;

В ядрах мы будем работать с root и в user mode, а в kernel mode.

## Загружаемые модули ядра

Linux имеет модульное ядро.

Чтобы внести в него изменения, надо его перекompilировать (паши);  
↳ опасно, т.к. можно получить неработоспособное ядро

В UNIX/Linux можно вносить изменения без перекompilляции с помощью загр. модулей ядра;

Это ИО, которое пишется по опред. шаблону;

Структура загр. модуля ядра на примере hello world:

```

#include <linux/module.h>
#include <linux/kernel.h>
#define <linux/init.h>
// указание лицензий обязательно
MODULE_LICENSE("GPL");
static int __init my_module_init()
{
    printk(KERN_INFO "Hello, world!\n");
    return 0;
}
static void __exit my_module_exit()
{
    printk(KERN_INFO "Stop\n");
}
module_init(my_module_init);
module_exit(my_module_exit);

```

↗ kernel, имеет в сист. лог, доступна в ядре  
 ↳ уровень прото- ↳ 'не нужна  
 коллирования  
 в модулях ядра  
 можно использовать  
 либо только под-  
 шотки ядра, стан-  
 дартная библиотека  
 не доступна

↳ макрос: ядро информируется о том, что  
 в ядре теперь есть эти оп-ции;

makefile (простейший):  
 obj-m += module.o  
 all:  
 make -C /lib/modules/\$(shell uname -r)  
 /build M=\$(pwd) modules

clean:  
 make -C /lib/modules/\$(shell uname -r)  
 /build M=\$(pwd) clean

Будем опираться на статьи Олега Цурганова  
 (номер для IBM.com)

- C - опция смены каталога и переход в каталог исходных кодов ядра
- M - возврат обратно в каталог исходных кодов модуля;

Будем выводить в лог ps -ajx

```
#include <linux/module.h>
#include <linux/kernel.h>
#define <linux/sched.h>
MODULE_LICENSE("GPL");
// дескрипторы процессов организации в ядре в калывевоу списоу;
// начало - init_task, переход на след. дескриптор - next_task;
int init_module(void)
{
    struct task_struct *task = &init_task;
    do
    {
        printk(KERN_INFO "- %s, pid - %d,
            parent: - %s, pid - %d\n", task->comm,
            task->pid, task->parent->comm,
            task->parent->pid);
    } while ((task = next_task(task)) != &init_task);
    return 0;
}

int exit_module()
{
    return 0;
}
```



insmod - загрузить модуль ядра;  
lsmod - посмотреть список модулей ядра;  
rmmod - выгрузить загруженный модуль из ядра;

Все эти действия доступны только с правами superuser;

Вывести инфор. из логa: dmesg

При этом надо исп. связь current  
(текущий процесс)

используется while (который надо написать после)

Всего 8 ур. протоколирования  
(уровней вывода сообщений):

- 0 - KERN\_EMERGE (опасность, <sup>сист.</sup>упада)
- 1 - KERN\_ALERT (тревога, <sup>сист.</sup>скоро упадет)
- 2 - KERN\_CRIT
- 3 - KERN\_ERR
- 4 - KERN\_WARNING
- 5 - KERN\_NOTICE
- 6 - KERN\_INFO
- 7 - KERN\_DEBUG

В дополнение к примеру надо выво-  
дить state (принимать про его значения),  
flags, prio (не единственное поле, <sup>и что это</sup> определяю-  
щее приоритет), policy;

Также есть ещё след. поля:

struct mm\_struct \* mm } выводить  
struct mm\_struct \* active-mm } не надо

Диз. страницы загрузаются в рез-те итер-и,  
за исключением первых страниц

comm - поле типа char;

В дескрипторе процесса есть поля  
struct fs\_struct и struct files\_struct;  
ан. мануал

Выведем инф. о ф.с. (struct fs\_struct \*fs):  
данные из неё  
получить foot ←

Процесс до запуска был файлом ⇒  
⇒ принадлежит какой-то конкретной  
ф.с., у которой был foot

Также в struct task\_struct есть поле  
struct thread\_struct  
похожее (ан. мануал)  
+ поле, которое обязательно быть в конце  
структуры  
(могут созд. доп. поля, поэтому  
эти структуры располагаются  
в конце)

Зад. №2 по загр. модулям ядра:  
взаимодействие загр. модулей  
ядра (по Цирковичу)  
↳ там заплата

Есть корреляция с ассемблером  
(любая на много модульные программы)

Тыгышук объявление

→ extern + header + макрос export\_symbol