Зеркальное производство - плохо

page
map - отображение каждой вирт. страницы процесса на физ - фрейм или в обл - ть свопинга;

В UNIX/Linux потоки отн. тяжёлые;

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);

epoll для ускорения обработки соединений (мультиплексирование вместо многопоточности);

- root
/proc/[pid]/root

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

- stat - файл - информация о процессе;

Записывать инфу о своих процессах в файл и показывать на табе;

environ:

```c
#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, скомпилированные gcc и Qt;

Отзывчивость программы → реакция на интерактивные события очень важна

реальное время

↳ см. стр-е реального времени POSIX (было ранее):

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

В лабах мы будем работать с proc и в user mode, и в kernel mode.

Загружаемые модули ядра

Linux имеет монолитное ядро.
Чтобы внести в него изменения, надо его перекомпилировать (патчи);
↳ опасно, т.к. можно получить неработающее ядро
В UNIX/Linux можно вносить изменения без перекомпиляции с помощью загр. модулей ядра;
Это ПО, которое пишется по опр. шаблонам;

Структура загр. модуля ядра на примере hello world:

```
#include <linux/module.h>
#include <linux/kernel.h>
#define <linux/init.h>
// Указание лицензии 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);
```

printk → kernel, пишет в сист. лог, доступна в ядре

KERN_INFO → уровень протоколирования

→ ',' не нужна

В модулях ядра можно использовать только библиотеки ядра, стандартная библиотека недоступна

module_exit → макрос: ядро информируется о том, что в ядре теперь есть эти ф-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 Цирюлика (пишет 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 организованы в ядре в кольцевой список;
//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 справами superuser;

Вывести инф. из лога: dmesg
При этом надо исп. ссылку current (текущий процесс)
Используется printk (который надо написать после while)

Всего 8 ур. протоколирования (уровней вывода сообщений):
0 – KERN_EMERGE (опасность, сист. упала)
1 – KERN_ALERT (тревога, сист. скоро упадёт)
2 – KERN_CRIT
3 – KERN_ERR
4 – KERN_WARNING
5 – KERN_NOTICE
6 – KERN_INFO
7 – KERN_DEBUG

В дополнение к примеру надо выводить state (прочитать про его значения и что это), flags, prio (не единственное поле, определяющее приоритет), policy;

Также есть ещё след. поля:
struct mm_struct *mm
struct mm_struct *active_mm
} выводить не надо

Физ. страницы загружаются в рез-те прер-й, за исключением первых страниц

comm – поле типа char;

В дескрипторе процесса есть поля struct fs_struct и struct files_struct; (см. мануал)

Выведем инф. о ф.с. (struct fs_struct *fs): (должны из неё получить root)

Процесс до запуска был оффлайн ⇒ ⇒ принадлежал какой-то конкретной ф.с., у которой был root

Также в struct task_struct есть поле struct tread_struct + поле, которое обязано быть в конце структуры (похожее (см. мануал)
(могут созд. доп. потоки, поэтому эти структуры располагаются в конце)

Зад. №2 по загр. модулям ядра:
взаимодействие загр. модулей ядра (по Цирюлику)

↳ там зафейлил

Есть корреляция с ассемблером
(либа на многомодульные программы)

Пильщик

объявление в header + макрос EXPORT_SYMBOL

extern +