04.06. 1

Лекция 04.06.

· 5 позиций, выделенных в совр. системах по отношению к очередям работ:

1. Work-работа
2. Workqueue-очередь работ
3. Worker или work_thread
4. Worker-pool (worker-pool и worker 1:N)
5. pwq (pool_workqueue)

$\text{workqueue} \xleftarrow[1]{n} \text{worker-pool}$

} worker-ов в worker-pool м.б. много

· Тасклет быстрее выполняет необход. дей-я. (prio=120, policy=0)

в ядре написано → Тасклет планируется на том пр-ссоре, на кот. выполнялась обработчик аппар. прер-ия и, как правило, по завершении аппар. прер-е тасклет выполняется сразу.

! Ф-ции ядра не м.б. сис. вызовами; ф-ции ядра не м.б. др;

· В ф-ции alloc_workqueue есть флаги.
Это сделано, чтобы освободить разработчика от необходимости указывать перечень флагов →

```
#define create_freeable_workqueue(name) alloc_workqueue
        ("%s", __WQ_LEGACY | WQ_FREEZABLE | WQ_UNBOUND |
        WQ_MEM_RECLAIM, 1, (name))
```

- WQ_UNBOUND - отключает привязку к cpu (но НЮ сказала „ядру")
- WQ_FREEZABLE - м. б. заморожен, приостановлен
- WQ_MEM_RECLAIM - may be used for memory reclaim ~ восстановление памяти?

Управление внеш. устр-вами

· Говоря о внеш. устройствах в Unix, на 1ую позицию выступает понятие " спец. файлы устройств"

буква "c" - спец. файл символьного устр-ва
буква "b" - спец. файл блочного устр-ва

Сделано это чтобы обеспечить работу с внеш. устр-вами как с файлами (обычными)

✓ Benefit от этого: когда мы пишем/читаем инфор-цию, обращаясь к внеш. устр-ву, мы используем сис. вызовы для чтения и записи в файл read(), write()

04.06.

○ Файл устройства, так же как обычный файл, (условно) м.б. открыт, закрыт, в него можно писать и из него читать.

○ Как правило, ОС Unix/Linux каждому внеш. устр-ву ставит в соответствие один спец. файл. Обычно их можно увидеть в каталоге /dev.

· Если смотреть инф-цию через ls-l в каталоге /dev, то можно увидеть: crw-rw-rw- 1 root root 1, 3 April 11 2002 null

это null-устройство, куда отправились файлы в демоне (0,1,2 отправляются в демоне, чтобы ошибки не возникали)

——— " ——— 1,5 — " ——— zero

· Каждое устр-во имеет атрибуты — это 2 числа, кот. отражают идентификацию устр-ва в системе. Для идентификации устройства в системе принята принята система старшего и младшего номеров (major и minor)

· Традиционно, старший и младший номер идентифицируют драйвер, связанный с устр-вом. Напр., /dev/null и /dev/zero управляются драйвером 1 (стар. номер), а виртуал. консоли и последовательные терминалы управляются старшим номером 4. (это про tty). Младший номер позволяет различать эти устр-ва (null и zero) (также как младший номер у tty позволяет различать консоли).

Яркий пример: жёсткий диск (он имеет старший номер, а partisions - младшие номера)

○ В ядре определён тип dev_t в <linux/types.h>

```
typedef --Kernel_dev_t dev_t;
```

Начиная с версии 2.6.0 это 32 разрядное число. Формат типа не оговаривается. Стандарт Posix определяет существование типа, но не оговаривает формат хранения

○ 32-разрядное число в Linux форматируется так:

12 бит отведено для старшего номера
20 бит — для младшего

04.06 3

• В ядре делается предупреждение:
Ваш код не должен делать каких-либо предположение о внутр. организации номеров устройств. (Принимайте как данность) Он должен просто использ-ть набор макросов из библиотеки linux kdev_t.h ? (cuf.h)

• Чтобы получить старшую и младшую часть типа dev_t, есть макросы:
• MAJOR(dev_t dev)
• MINOR(dev_t dev)

Есть обратное преобразование.
Т.е. если у имеются старший и младший номера, то нужно преобразовать в этом типе dev_t (...что?)
макрос MKDEV(int major, int minor) – объединяет старш. и младш. номера в двоичного типа dev_t

• Начиная с 2.6. ядро Linux может поддерживать огромное число номеров устройств: 255 млад. номеров, 255 старших

• Самый частый пример, где предположение распространения типа старшие и младш. номера устройств – char_device

• Старший и младш. номера характеризуют драйвер, а не устр-во. (устр-во м.б. 1, а драйверов, написанных для него, много)

(*) int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count, const char * name);

Ф-ция выделяет диапазон номеров символьных устройств. Старший номер выбирается динамически и ф-ция возвращает его одновременно с первым младшим номером в dev. Если получилось получить старший диапазон младших номеров, то ф-ция возвр. 0, иначе возвр. отриц. код.

• char_device – загруж. модуль ядра, оформленный в виде драйвера char-устройства. Его можно использ-ть для передачи данных, каких-то действий в ядре.

Вызов ф-ции (*) заменит ряд действий. Её код ⟶

04.06 (4)

```
int alloc_chrdev_region(dev_t *dev, unsigned baseminor,   ← начальн. номер
{                         unsigned count, const char *name)
  struct char_device_struct *cd;
  cd = __register_chrdev_region(0, baseminor, count, name);
  if (IS_ERR(cd)) return PTR_ERR(cd);
  *dev = MKDEV(cd->major, cd->baseminor);
  return 0;
}
```

// в dev будет записан 1го номер выделенного диапазона

null устройство - это символьное устр-во

- Пример драйвера, кот. вешается на линию:

```
#define IRQ_NO 11
  ...
  static DECLARE_WORK(work, workqueue_fn);
  ...
```

{ Важный момент этого кода — определение точек входа драйвера (read, write, open, release)

```
static struct file_operations fops =
{  .owner = THIS_MODULE,
   .read = etx_read,
   .write = etx_write,
   .open = etx_open,
   .release = etx_release
};
static int __init etx_driver_init(void)
{ /* Allocating major number */
  if ((alloc_chrdev_region(&dev, 0, 1, "etx_dev")) < 0)
  { printk(KERN_INFO "can't");
    return -1;
  }
  printk(KERN_INFO "major = %d, minor = %d\n", MAJOR(dev), MINOR(dev));
/* creating cdev structure */
cdev_init(&etx_cdev, &fops);
if ((cdev_add(&etx_cdev, dev, 1)) < 0)
{ printk(...);
  unregister_chrdev_region(dev, 1);
  cdev_del(&etx_cdev);
  return -1; }
```

добавление char-device в систему

04.06

(5)

Создание класс

```
/* creating struct class*/
if ((dev_class = class_create (THIS_MODULE, "etx_class")) == NULL)
{...}
/* creating device*/
if (device-create(dev_class, NULL, dev, NULL, "etx_device")) == NULL))
{...}
...
if (request_irq (IRQ_NO, irq-handler, IRQF_SHARED, "etx_device",
    (void *) (irq-handler))) {...}   ← + free_irq ()?
                                      (если не удалось установить обработ-к прерыв.)
/* creating workqueue*/

}
```

3 типа драйверов устр-в:

① Драйверы, встроенные в ядро:
Соответ-щие устройства автоматически обнаруживаются системой и становятся доступны приложениям
(напр контроллер IDE?)

② Драйверы, реализованные как загр. модули ядра:
Часто такими драйверами явл драйвера для управления такими устр-вами, как звуковые/сетевые карты, SCSI-адаптеры.

Файлы модулей ядра располагаются в подкаталогах каталога /lib/modules.

Обычно при инсталляции системы задаётся перечень модулей, кот. будут автоматически подгружаться на этапе загрузки. Список загр. модулей хранится в файле /etc/modules.

Для подключения/отключения модулей в работающей системе ∃ спец. утилиты: insmod, rmmod, lsmod, modprobe

Чтобы отобразить текущ. конфигурацию всех модулей, можно использ. modprobe -c

③ Код драйверов этого типа поделён м/у ядром и спец. утилитами
Напр, 1) драйвер принтера: ядро отвечает за взаимодействие с параллельным портом, а формирование управляющ. сигналов для принтера осуществляет демон печати lpd.

2) драйвер модема

. Часто подсистему, предназнач. для работы с внеш. устр-вами, наз-т подсистемой вв/вывода. Эта самая сложная подсистема в ОС.

В системе определено большое к-во структур для работы с внеш. устр-вами.

. Самая низкоуровнев. стр-ра Linux — struct device

struct device

```
{
  struct kobject kobj;
  struct device *parent;
  ...
  const char * init_name;
  const struct device_type *type;
  struct bus_type *bus;
  struct device_driver *driver;
  ...
 #ifdef CONFIG_GENERIC_MSI_IRQ
        struct list_head msi_list;
 #endif
 const struct dma_map_ops *dma_ops;
  ...
  struct list_head dma_pools;
  ...
#ifdef CONFIG_NUMA
     int numa_node;
#endif
  dev_t devt;
  u32 id;
  ...
}
```

- MSI — в совр. системах способ доставки процессорам информации об аппар. прерываниях. (message signal interrupts)
  /proc/interrupts

- Numa — совр. архитектура процессоров

struct device_driver — the driver core (из эликсира)