MINIX 3 API вызовов ядра

Jorrit N. Herder
<jnherder@cs.vu.nl>
20.10.2005

Аннотация

Главным образом, вызовы ядра позволяют системным процессам запрашивать сервисы ядра, например, для осуществления привилегированных операций. Этот документ сжато обсуждает организацию вызовов ядра в MINIX3 и предоставляет обзор всех вызовов ядра.

Организация вызовов ядра

Вызов ядра означает, что запрос посылается ядру, где он обслуживается одной из задач (tasks) ядра. Детали асемблерного кода сообщения запроса, посылки его ядру, и ожидание ответта удобно сокрыты в системной библиотеке. Заголовочный файлы этой библиотеки - `src/include/minix/syslib.h`, а файлы реализации находятся в каталоге `src/lib/syslib`.

Действительные реализации вызовов ядра определены в одной из задач ядра. В противоположность MINIX2, задача `CLOCK` более не принимает системных вызовов. Вместо этого, все вызовы теперь направляются к задаче `SYSTEM`. Предполагается, что программа делает `sys_call()` системный вызов[1]. Согласно соглашениям, этот вызов трансформируется в сообщение запроса с типом `SYS_CALL`, которое посылается задаче ядра `SYSTEM`. Задача `SYSTEM` обслуживает запрос в функции с именем `do_call()` и возвращает результат.

Карта отображения номеров вызовов ядра на функции обработчики делается во время инициализации задачи `SYSTEM` в `src/kernel/system.c`. Прототипы функций обработчиков декларированы в `src/kernel/system.h`. Их реализации содержатся в отдельных файлах в каталоге `src/kernel/system/`. Эти файлы компилируются в библиотеку `src/kernel/system/system.a`, которая компонуется с ядром.

Номера вызовов ядра, их параметры запроса и параметры ответа, определены в `src/include/minix/com.h`. К сожалению, MINIX2 не следует строгой схеме именования. Поэтому, нумерация типов сообщений и параметров были переименованы в MINIX3. Вызовы ядра сейчас все начинаются с `SYS` и все параметры, которые принадлежат к одному и тому же вызову ядра, теперь разделяют общий префикс.

Обзор вызовов ядра MINIX3

Сжатый обзор вызовов ядра в MINIX3 приведен ниже, на рисунке 1. Статус каждого вызова относительно MINIX2 даётся в последней колонке.

1Здесь оригинал не точен. Каждая функция вызова ядра вида `sys_*()`, на самом деле ретранслируется в вызов функции отправки сообщения запроса `_taskcall()` с соответствующими параметрами (а не `syscall()`, как указано в тексте). Тип запроса `_taskcall()` при этом (2-й параметр) будет соответствовать запрошенной операции (например `SYS_FORK`), а не `SYS_CALL`. Код функции `_taskcall()` вы можете найти: **`/usr/src/lib/syslib/taskcall.c`** (прим. перев.)

Вызов ядра	Цель	Статус
	управление процессами	
SYS_FORK	Fork a process; copy parent process	
SYS_EXEC	Execute a process; initialize registers	
SYS_EXIT	Exit a user process; clear process slot	U
SYS_NICE	Change priority of a user process	N
SYS_PRIVCTL	Change system process privileges	N
SYS_TRACE	Trace or control process execution	
	обслуживание сигналов	
SYS_KILL	Send a signal to a process	U
SYS_GETKSIG	Check for pending kernel signals	N
SYS_ENDKSIG	Tell kernel signal has been processed	
SYS_SIGSEND	Start POSIX-style signal handler	
SYS_SIGRETURN	Return from POSIX-style signal	
	управление памятью	
SYS_NEWMAP	Install new or updated memory map	
SYS_SEGCTL	Add extra, remote memory segment	N
SYS_MEMSET	Write a pattern into physical memory area	N
	копирование данных	
SYS_UMAP	Map virtual to physical address	U
SYS_VIRCOPY	Copy data using virtual addressing	U
SYS_PHYSCOPY	Copy data using physical addressing	U
SYS_VIRVCOPY	Handle vector with virtual copy requests	U
SYS_PHYSVCOPY	Handle vector with physical copy requests	N
	ввод/вывод на устройствах	
SYS_DEVIO	Read or write a single device register	N
SYS_SDEVIO	Input or output an entire data buffer	N
SYS_VDEVIO	Process vector with multiple requests	N
SYS_IRQCTL	Set or reset an interrupt policy	N
SYS_INT86	Make a real-mode BIOS call	N
SYS_IOPENABLE	Give process I/O privilege	N
	управление системой	
SYS_ABORT	Abort MINIX: shutdown the system	U
SYS_GETINFO	Get a copy system info or kernel data	N
	функции часов	
SYS_SETALARM	Set or reset a synchronous alarm timer	U
SYS_TIMES	Get process times and uptime since boot	U

Рисунок 1. Этот рисунок приводит обзор вызовов ядра MINIX3. Обозначения для статуса: N — новый, U — обновлённый (т.е. Полностью обновлённый — все вызовы получили минимальные обновления) со времени MINIX2.

Интерфейс вызовов ядра

Интерфейс вызовов ядра детализируется ниже. Для каждого вызова ядра специфицируется его назначение, тип сообщения, параметры запроса и/или ответа, возвращаемое вызовом значение кода возврата. Стенографические дополнительные ремарки (примечания) о будущем статусе вызова также могут быть приведены.

Обозначения в тексте

`CONSTANT`: определение числовой константы; числовой индикатор типа запроса или его возможных значений;

`PARAMETER` : параметр сообщения; поля параметров в сообщениях запроса или ответа.

`void sys_call(arguments)`: функция из системной библиотеки; заметка о выполнении вызова ядра;

Алфавитный обзор

`SYS_ABORT`: Завершить MINIX и возвратиться в монитор загрузки — если возможно. Эта возможность используется PM, FS и TTY. Нормально завершение инициируется обычно пользователем, например, обозначая команду `shutdown`, или `<Ctrl><Alt><Del>`. MINIX также будет осуществлять это действие, если обнаружится фатальная ошибка в PM или FS.

- параметры запроса:

`ABRT_HOW`: как остановить, одно из значений, определённых в `src/include/unistd.h` :

" `RBT_HALT` — остановить MINIX и вернуться в монитор загрузки.
" `RBT_REBOOT` — перезагрузить MINIX.
" `RBT_PANIC` — наблюдается 'kernel panic'.
" `RBT_MONITOR` — выполнить специфицированный код в мониторе загрузки.
" `RBT_RESET` — аппаратный сброс системы.

`ABRT_MON_PROC` : процесс, из которого получить параметры для монитора загрузки.

`ABRT_MON_LEN` : длина списка параметров монитора загрузки.

`ABRT_MON_ADDR` : виртуальный адрес параметров.

- возвращаемый результат:

`OK`: последовательность завершения стартовала.
`ENIVAL`: ошибочный номер процесса.
`EFAULT`: недопустимый адрес параметра монитора.
`E2BIG`: параметры монитора превышают максимальную длину.

- библиотечные функции:

`int sys_abort(int shutdown status, ...);`

`SYS_DEVIO`: Осуществить ввод/вывод на устройстве от имени драйвера устройства из пользовательского пространства. Драйвер может запросить одиночный порт, который будет прочитан или записан этим вызовом. Смотри также вызовы `SYS_SDEVIO` и `SYS_VDEVIO`.

- параметры запроса:

`DIO_REQUEST` : ввод или вывод.

" `DIO_INPUT` прочитать значение из `DIO_PORT`.
" `DIO OUTPUT` записать `DIO_VALUE` в `DIO_PORT`.

`DIO_TYPE` : флаг, индицирующий тип значения.

" `DIO BYTE` байт.
" `DIO WORD` слово.
" `DIO LONG` Long тип.

DIO_PORT : порт для чтения или записи.
DIO_VALUE : записываемое значение, только для DIO_OUTPUT.
- параметры ответа:
DIO VALUE : значение, сосчитанное из указанного порта, только для DIO_INPUT.
- возвращаемый результат:
OK: порт ввода/вывода успешно обслужен.
EINVAL: недопустимые *DIO_REQUEST* или *DIO_TYPE* было установлено.
- библиотечные функции:

```
int sys_in(port t port, unsigned long value, int io type);
int sys_inb(port t port, u8 t *byte);
int sys_inw(port t port, u16 t *word);
int sys_inl(port t port, u32 t *long);
int sys_out(port t port, unsigned long *value, int io type);
int sys_outb(port t port, u8 t byte);
int sys_outw(port t port, u16 t word);
int sys_outl(port t port, u32 t long);
```

SYS_ENDKSIG: Завершить обработку сигнала ядром. PM использует этот вызов, чтобы обозначить этим, что он обработал то, что ядро сигнализировало в отображении через SYS_GETKSIG вызов.
- параметры запроса:
SIG PROC : процесс, которого это касается.
- возвращаемый результат:
EINVAL: процесс не имеет не обработанных сигналов, или завершается.
OK: ядро очистило все ожидающие сигналы.
- библиотечные функции:

```
int sys_endksig(int proc nr);
```

SYS_EXEC: Обновить регистры процесса после успешного exec() POSIX-вызова. После того, как FS копировал бинарный образ в память, PM информирует ядро о новой детализации регистров.
- параметры запроса:
PR_PROC_ NR : процесс который выполняет программу.
PR_STACK_PTR : новый указатель стека.
PR_IP_PTR : новый счётчик команд.
PR_NAME_PTR : указатель на имя прграммы.
- возвращаемый результат:
OK: этот вызов всегда успешен.
- библиотечные функции:

```
int sys_exec(int proc, char *stack ptr,
             char *prog name, vir bytes pc);
```

SYS EXIT: Очистить слот (в таблице) процессов. Это обычно исходит от PM посде завершения пользовательского процесса. Системные процессы, включая PM, могут также непосредственно вызывать эту функцию по их завершению.
- параметры запроса:
PR_PROC_NR : номер слота завершаемого процесса, если вызывающей стороной есть PM.
Используйте SELF, если завершается сам PM.

- возвращаемый результат:
    OK: очистка успешна.
    EINVAL: некорректный номер процесса.
    EDONTREPLY: этот вызов не возвращается, если системный процесс завершается.
- библиотечные функции:
    int sys_exit(int proc nr);

SYS_FORK: Инициировать новый (дочерний) процесс в таблице процессов ядра, и инициализировать его, базируясь на прототипе (родителе) процесса. PM находит свободный процессорный слот для дочернего процесса в своей таблице процессов, и тут же запрашивает ядро обновить таблицу процессов ядра.
- параметры запроса:
PR_PROC_NR : дочернего процесса слот в таблице.
PR_PPROC_NR : родительский процесс, выполняющий fork().
- возвращаемый результат:
    OK: новый процессорный слот успешно присвоен.
    EINVAL: неверный номер родительского процесса, или дочерний слот занят.
- библиотечные функции:
    int sys_fork(int parent proc nr, int child proc nr);

SYS_GETINFO: Получить копию структуры данных ядра. Этот вызов поддерживает драйверы и сервера пользовательского пространства, нуждающиеся в достоверной системной информации.
- параметры запроса:
I_REQUEST : тип системной информации, которая запрашивается.
    " GET_IMAGE копировать таблицу загрузочного образа.
    " GET_IRQHOOKS копировать таблицу с перехватчиками прерываний.
    " GET_KINFO копировать информационную структуру ядра.
    " GET_KMESSAGES копировать буфер с диагностическими сообщениями ядра.
    " GET_LOCKTIMING копировать lock times—if DEBUG TIME LOCKS is set.
    " GET_MACHINE копировать системное окружение.
    " GET_MONPARAMS копировать набор параметров монитора загрузки.
    " GET_PRIVTAB копировать таблицу системных привилегий.
    " GET_PROCTAB копировать полностью таблицу процессов ядра.
    " GET_PROC копировать одиночный слот таблицы процессов.
    " GET_RANDOMNESS копировать неупорядоченность (randomness) собранную событиями ядра.
    " GET_SCHEDINFO копировать очереди готовности и процессорную таблицу.
I_VAL_PTR : виртуальный адрес куда должна быть скопирована информация.
I_VAL_LEN : максимальная длина с которой вызывающий может оперировать.
I_VAL_PTR2 : опционально, второй адрес. Используется когда копируются диспетчера данные.
I_VAL_LEN2 : опционально, вторая длина. Перезагружается номером процесса.
- возвращаемый результат:
    OK: успешный информационный запрос.
    EFAULT: был диагностирован недопустимый адрес памяти.
    E2BIG: запрашиваемые данные превышают максимум, обеспечиваемый вызывающим.
- библиотечные функции:
    int_sys_getinfo(int request, void *ptr, int len,

```
                    void *ptr2, int len2);
   int_sys_getirqhooks(struct irq hook *ptr);
   int_sys_getimage(struct boot image *ptr);
   int_sys_getkinfo(struct kinfo *ptr);
   int_sys_getkmessages(struct kmessages *ptr);
   int_sys_getlocktimings(struct lock timingdata *ptr);
   int_sys_getmachine(struct machine *ptr);
   int_sys_getmonparams(char *ptr, int max len);
   int_sys_getprivtab(struct priv *ptr);
   int_sys_getproctab(struct proc *ptr);
   int_sys_getproc(struct proc *ptr, int proc nr);
   int_sys_getrandomness(struct randomness *ptr);
   int_sys_getschedinfo(struct proc* ptr, struct proc *ptr2);
```

SYS_GETKSIG: Проверяет, этот ли есть процесс, который должен получить сигнал. Это многократно выполняется PM после того, как он получит уведомление, что присутствуют задержанные сигналы ядра.
- параметры запроса:
SIG_PROC : возвратить следующий процесс с задержанными сигналами или NONE.
SIG_MAP : битовая карта с задержанными сигналами ядра.
- возвращаемый результат:
    OK: этот вызов всегда успешен.
- библиотечные функции:
```
   int sys_getksig(int *proc nr, sigset t *sig map);
```

SYS_INT86: выполнить BIOS операцию реального режима от имени и по поручению драйвера устройства. Этот вызов временно переключает процессор из 32-бит защищённого режима в 16-бит реальный режим для доступа к вызову операции BIOS. Он присутствует для поддержки драйвера устрйства BIOS_WINI.
- параметры запроса:
INT86_REG86 : адрес запроса ввызывающей стороны.
- возвращаемый результат:
    OK: BIOS вызов успешно произведен.
    EFAULT: недопустимый адрес запроса.
- библиотечные функции:
    -

SYS_IOPENABLE: разрешить CPU биты I/O уровня привилегий для заданного процесса, что позволяет ему непосредственно осуществлять I/O операции в пользовательском пространстве.
- параметры запроса:
    PROC_NR : процесс которому предоставляются привилегии.
- возвращаемый результат:
    OK: этот вызов всегда успешен.
- библиотечные функции:
    -

SYS_IRQCTL: установить или сбросить стратегию аппаратного прерывания для заданной

IRQ линии, и разрешить или запретить прерывание от этой линии. Этот вызов позволяет драйверу пользовательского пространства захватывать ловушку для использования с типовым обработчиком прерываний ядра. Обработчик прерывания ядра просто уведомляет драйвер о прерывании сообщением HARD_INT, и переразрешает прерывание с IRQ линии, если стратегия требует этого. Уведомляющее сообщение будет содержать 'id', предоставляемое вызывающей стороной как аргумент. Как только стратегия введена в действие, драйвера могут разрешать и запрещать прерывания.

- параметры запроса:

IRQ_REQUEST : управление прерыванием, которое необходимо осуществить.

" IRQ_SETPOLICY установить стратегию прерывания для типового обработчика прерывания.

" IRQ_RMPOLICY удалить ранее установленнкю стратегию прерывания.

" IRQ_ENABLE разрешить прерывания для заданной IRQ линии.

" IRQ_DISABLE запретить прерывания для заданной IRQ линии.

IRQ_VECTOR : IRQ линия подлежащая управлению.

IRQ_POLICY : битовая карта с флагами индицирующими стратегию IRQ.

IRQ_HOOK_ID : когда устанавливается стратегия, здесь предоставляется индекс, который посылается вызывающей стороне при возникновении прерывания. Для всех других запросов это идентификатор ловушки ядра, возвращаемый ядром.

- параметры ответа:

IRQ_HOOK_ID : идентификатор ловушки ядра, ассоциированный с драйвером.

- возвращаемый результат:

EINVAL: ошибка запроса: IRQ линия, *IRQ_HOOK_ID*, или номер процесса[2].

EPERM: только владелец *IRQ_HOOK_ID* может переключать прерывания или освобождать ловушку.

ENOSPC: не может быть найдено сободных ловушек.

OK: запрос успешно обслужен.

- библиотечные функции:

```
int sys_irqctl(int request, int irq_vec, int policy,
               int *hook_id);
int sys_irqsetpolicy(int irq_vec, int policy, int *hook_id);
int sys_irqrmpolicy(int irq_vec, int *hook_id);
int sys_irqenable(int hook_id);
int sys_irqdisable(int hook_id);
```

SYS_KILL: послать сигнал процессу от лица системного сервера. Системный процесс может послать сигнал другому процессу этим вызовом. Ядро уведомляет PM о задержанном сигнале для последующей обработки. (Отметим, что POSIX вызов kill() напрямую обрабатывается PM.) PM использует этот вызов, чтобы опосредовано послать сигнальное сообщение системному процессу. Такое случается, когда сигнал поступает для системного процесса, который установил специальный SIG_MESS сигнальный обработчик с помощью POSIX вызова sigaction().

- параметры запроса:

SIG_PROC_NR : процесс, которому будет послан сигнал.

SIG_NUMBER : номер сигнала. Диапазон от 0 до NSIG.

- возвращаемый результат:

OK: вызов успешный.

EINVAL: недопустимый процесс или номер сигнала.

EPERM: невозможно послать сигнал задаче ядра; PM не может послать сигнал

2 Так в оригинале, хотя номер процесса в запросе не фигурирует.

пользовательскому процессу посредством уведомляющего сообщения.
- библиотечные функции:

```
int sys_kill(int proc nr, int sig nr);
```

SYS_MEMSET: записать 4-байтный шаблон в индицируемую область памяти. Этот вызов используется PM для обнуления BSS сегмента при POSIX вызове `exec()`. Ядро запрашивается для выполнения этой работы из соображений производительности.
- параметры запроса:
MEM_PTR : физический базовый адрес области памяти.
MEM_COUNT : протяжённость в байтах области памяти.
MEM_PATTERN : 4-х байтовый шаблон, который должен быть записан.
- возвращаемый результат:
OK: этот вызов всегда успешен.
- библиотечные функции:

```
int sys_memset(long pattern, phys bytes base,
               phys bytes length);
```

SYS_NEWMAP: установить новое отображение памяти для нового процесса, создаваемого `fork()`, или если отображени памяти процесса изменяется. Ядро выбирает новое отображение памяти из PM и обновляет свои структуры данных.
- параметры запроса:
PR_PROC_NR : Install new map for this process.
PR_MEM_PTR : Pointer to memory map at PM.
- возвращаемый результат:
OK: новое отображение было успешно установлено.
EFAULT: некорректный адрес нового отображения памяти.
EINVAL: неверный номер процесса.
- библиотечные функции:

```
int sys_newmap(int proc nr, struct mem map *ptr);
```

SYS_NICE: Изменить приоритет процесса. Это достигается передачей значения изменения приоритета между PRIO_MIN (отрицательное) и PRIO_MAX (положительное). Значение 0 восстанавливает приоритет в умалчиваемое значение.
- параметры запроса:
PR PROC NR : процесс, чей приоритет должен быть изменён.
PR PRIORITY : новое изменение приоритета для процесса.
- возвращаемый результат:
OK: новое значение приоритета установлено.
EINVAL: недопустимое значение номера процесса или приоритета.
EPERM: невозможно изменить приоритет задачи ядра.
- библиотечные функции:

```
int sys nice(int proc nr, int priority);
```

SYS_PHYSCOPY: копировать данные, используя физическую адресацию. Источник и приёмник могут быть виртуальными подобно SYS_VIRCOPY, но в дополнение произвольный физический адрес принимается в качестве PHYS_ SEG.
- параметры запроса:
CP_SRC_SPACE : сегмент источника.

CP_SRC_ADDR : виртуальный адрес источника.
CP_SRC_PROC_NR : номер процесса процесса источника.
CP_DST_SPACE : сегмент получателя.
CP_DST_ADDR : виртуальный адрес получателя.
CP_DST_PROC_NR : номер процесса процесса получателя.
CP_NR_BYTES : число байт для копирования.
- возвращаемый результат:
    OK: копирование произведено.
    EDOM: неверный счётчик копирования.
    EFAULT: виртуально в физическое отображение потеряно.
    EINVAL: неукорректный сегмента тип, или номер процесса.
    EPERM: только владелец REMOTE_SEG может копировать в него или из него.
- библиотечные функции:

```
int sys_abscopy(phys_bytes_src phys, phys_bytes dst phys,
                phys_bytes count);
int sys physcopy(int src_proc, int src_seg, vir_bytes src_vir,
                 int dst_proc, int dst_seg,
                 vir_bytes dst_vir, phys_bytes count);
```

SYS_PHYSVCOPY: копировать множественные блоки данных, используя физическую адресацию. Запрашиваемый вектор выбирается вызывающей стороной, и каждый элемент вектора обслуживается подобно запросу SYS_PHYSCOPY. Копирование продолжается до тех пор, пока все элементы будут обслужены, или будет наблюдаться ошибка.
- параметры запроса:
VCP_VEC_SIZE : число элементов в запрошенном векторе.
VCP_VEC_ADDR : виртуальный адрес вектора, запрошенного вызывающей стороной.
- параметры ответа:
VCP_NR_OK :число элементов успешно скопированных.
- возвращаемый результат:
    OK: копирование выполнено.
    EDOM: недопустимый счётчик копирования.
    EFAULT: отображение виртуального в физическое потеряно.
    EINVAL: копируемый вектор слишком велик, некорректный сегмент,
        или недопустимый процесс.
    EPERM: тольк собственник REMOTE_SEG может копировать в него, или из него.
- библиотечные функции:

```
int sys physvcopy(phys_cp_req *copy_vec,
                  int vec_size, int *nr_ok);
```

SYS_PRIVCTL: получение частных структур привилегий, и обновление привилегий процесса. Это используется при динамическом старте системных сервисов.
- параметры запроса:
CTL_PROC_NR : процесс, чьи привилегии должны быть обновлены.
- возвращаемый результат:
    OK: вызов успешен.
    EINVAL: недопустимый номер процессаr.
    ENOSPC: не найдено свободной структуры привилегий.
- примечания:
    Этот системный вызов будет расширять обеспечивать улучшенную, с двух сторон, и

поддержку и проверку защищённости, для серверов или драйверов, которые должны загружаться динамически. Это работа на будущее.

- библиотечные функции:

-

SYS_SDEVIO: осуществить I/O на устройстве, по поручению драйвера устройства пользовательского пространства. Заметим, что этот запрос поддерживает только байтовую иди пословную гранулярность операции. Драйвер может запросить ввод или вывод полного буфера. Дополнительно смотри вызовы ядра SYS_DEVIO и SYS_VDEVIO.

- параметры запроса:

DIO_REQUEST : ввод или вывод.

" DIO_INPUT считать значение из DIO_PORT.

" DIO_OUTPUT записать DIO_VALUE в DIO_PORT.

DIO_TYPE : флаг, индицирующий тип значения.

" DIO_YTE байт.

" DIO WORD слово.

DIO_PORT : порт для чтения или записи.

DIO_PROC_NR : номер процесса где находится буфер.

DIO_VEC_ADDR : виртуальный адрес буфера.

DIO_VEC_SIZE : число элементов ввода или вывода.

- параметры ответа:

DIO_VALUE : значение, считанное с заданного порта, для DIO_INPUT только.

- возвращаемый результат:

OK: операция I/O на порту успешно выполнена.

EINVAL: недопустимый запрос или для порта гранулярность.

EPERM: не возможно выполнение I/O для задачи ядра.

EFAULT: неверный виртуальный адрес буфера.

- библиотечные функции:

```
int sys_insb(port_t port, u8_t buffer, int count);
int sys_insw(port_t port, u16_t buffer, int count);
int sys_outsb(port_t port, u8_t buffer, int count);
int sys_outsw(port_t port, u16_t buffer, int count);
int sys_sdevio(int req, long port, int io_type,
               void *buffer, int count);
```

SYS SEGCTL: добавить сегмент памяти к LDT процесса, и его удалённой картой памяти. Этот вызов возвращает селектор и смещение, которые могут быть использованы для непосредственного доступа к удалённому сегменту, так же, как индекс в удалённой карте памяти, который может быть использован в вызове ядра SYS_VIRCOPY.

- параметры запроса:

SEG_PHYS : физический базовый адрес сегмента.

SEG_SIZE : размер сегмента.

- параметры ответа:

SEG_INDEX : индекс в удалённой карте памяти.

SEG_SELECT : селектор сегмента для LDT входа.

SEG_OFFSET : смещение в пределах сегмента; ноль, в противном случае
гранулярность 4K используется.

- возвращаемый результат:

ENOSPC: нет свободного слота в удалённой карте памяти и LDT.

OK: дескриптор сегмента успешно добавлен.
- библиотечные функции:

```
int sys_segctl(int *index, u16_t *seg, vir_bytes *off,
               phys_bytes phys, vir_bytes size);
```

SYS_SIGRETURN: возврат из обработчика сигнала стиля POSIX. PM запрашивает ядро привести все атрибуты в порядок, прежде, чем сигнализировать что процесс может продолжить выполнение. Также смотри вызов ядра SYS_SIGSEND, который заталкивает фрейм сигнального контекста в стек.
- параметры запроса:
SIG_PROC : индицирует процесс, который уведомляется.
SIG_CTXT_PTR : указатель но контекстную структуру обработчика сигнала стиля POSIX.
- параметры ответа:
SIG_PROC : возвращает следующий процесс с отложенными сигналами,
или NONE возвращаемое значение.
- возвращаемый результат:
OK: действие по обработке сигнала успешно осуществлено.
EINVAL: неверные номер процесса, или контекстная структура.
EFAULT: ошибочный адрес контекстной структуры, или невозможно
скопировать сигнальный фрейм.
- библиотечные функции:

```
int sys_sigreturn(int proc_nr, struct sigmsg *sig_context);
```

SYS_SIGSEND: сигнализировать процессу от лица PM о помещении контекстной структуры в стек. Ядро выбирает структуру, инициализирует её, и копирует её в пользовательский стек.
- параметры запроса:
SIG_PROC : индицирует процесс, который уведомляется.
SIG_CTXT_PTR : указатель но контекстную структуру обработчика сигнала стиля POSIX.
- параметры ответа:
SIG_PROC : возвращает следующий процесс с отложенными сигналами,
или NONE возвращаемое значение.
- возвращаемый результат:
OK: действие по обработке сигнала успешно осуществлено.
EINVAL: неверный номер процесса.
EPERM: нельзя сигнализировать задаче ядра.
EFAULT: ошибочный адрес контекстной структуры, или невозможно
скопировать сигнальный фрейм.
- библиотечные функции:

```
int sys_sigsend(int proc_nr, struct sigmsg *sig_context);
```

SYS_SETALARM: установить или сбросить синхронный таймер тревоги. Когда выдержка таймера истечёт, это приведёт в действие то, что SYN_ALARM уведомляющее сообщение, с текущим оперативным временем как аргумент, будет послано вызывающей стороне. Только системные процессы могут запрашивать синхронные тревоги.
- параметры запроса:
ALRM_EXP_TIME : абсолютная или относительная выдержка времени, в тиках,
для этой тревоги.
ALRM_ABS_TIME : ноль, если выдержка времени есть относительной

к текущему оперативному времени.
- параметры ответа:
ALRM_TIME_LEFT : тиков, оставшихся на предыдущей тревоге.
- возвращаемый результат:
OK: тревога успешно установлена.
EPERM: пользовательский процесс не может запрашивать тревоги.
- библиотечные функции:

```
int sys_setalarm(clock_t expire_time, int abs_flag);
```

SYS_TIMES: получить оперативное время ядра со времени загрузки и времена выполнения процесса.
- параметры запроса:
T_PROC_NR : процесс, для которого получить информацию по временам, или NONE.
- параметры ответа:
T_USER_TIME : время процесса в пользовательском режиме, если валидный номер.
T_SYSTEM_TIME : время процесса в режиме системы, если валидный номер.
T_BOOT_TICKS : число тиков с последней загрузки MINIX.
- возвращаемый результат:
OK: этот вызов всегда успешен.
- библиотечные функции:

```
int sys_times(int proc_nr, clock_t *ptr);
```

SYS_TRACE: мониторинг или управление выполнением заданного процесса. Обслуживает отладочные команды, поддерживаемые системным вызовом ptrace().
- параметры запроса:
CTL_REQUEST : трассировочный запрос.
" T_STOP остановить процесс.
" T_GETINS возвратить значение из пространства команд.
" T_GETDATA возвратить значение из пространства данных.
" T_GETUSER возвратить значение из таблицы процесса.
" T_SETINS установить значение в пространстве команд.
" T_SETDATA установить значение в пространстве данных.
" T_SETUSER установить значение в таблице процесса.
" T_RESUME возобновить выполнение.
" T_STEP установить бит трассировки.
CTL_PROC_NR : номер процесса, который будет трассироваться.
CTL_ADDRESS : виртуальный адрес в пространстве трассируемого процесса.
CTL_DATA: данные, которые должны быть записаны.
- параметры ответа:
CTL_DATA: данные, которые возвращены.
- возвращаемый результат:
OK: успешная операция трассировки.
EIO: устанавливаемое или запрашиваемое значение утеряно.
EINVAL: не поддерживаемый запрос трассировки.
PERM: трассироваться может только пользовательский процесс.
- библиотечные функции:

```
int sys_trace(int request, int proc_nr,
              long addr, long *data_ptr);
```

SYS_UMAP: отобразить виртуальный адрес в физический и возвратить физический адрес. Виртуальный адрес может быть из LOCAL_SEG, REMOTE_SEG, или BIOS_SEG. Смещение в байтах может быть передано, для того, чтобы убедиться, что оно действительно попадает внутрь сегмента.
- параметры запроса:
CP_RC_PROC_NR : номер процесса, к которому относится адрес.
CP_SRC_SPACE : идентификатор сегмента.
CP_SRC_ADDR : смещение в рамках сегмента.
CP_NR_BYTES : число байт от начала.
- параметры ответа:
CP_DST_ADDR : физический адрес, если отображение успешно.
- возвращаемый результат:
OK: копирование было выполнено[3].
EFAULT: отображение виртуального в физический утеряно.
EINVAL: некорректный тип сегмента или номер процесса.
- примечания:
Нулевой адрес внутри BIOS_SEG возвращает EFAULT, тогда как нулевой BIOS вектор прерывания фактически является валидным адресом.
- библиотечные функции:

```
int sys_umap(int proc_nr, int seg, vir_bytes vir_addr,
             vir_bytes count, phys_bytes *phys_addr);
```

SYS_VDEVIO: осуществить серию I/O операций от лица пользовательского процесса. Вызов принимает указатель на массив пар (порт, значение) которые должны быть обслужены одновременно. Аппаратные прерывания временно запрещаются, чтобы воспрепятствовать возможности пекету I/O вызовов быть прерванным. Смотри также SYS_DEVIO и SYS_SDEVIO.
- параметры запроса:
DIO_REQUEST : ввод или вывод.
" DIO_INPUT читать значение из DIO_PORT.
" DIO_OUTPUT писать DIO_VALUE в DIO_PORT.
DIO_TYPE : флаг, индицирующий тип значений.
" DIO_BYTE тип Byte.
" DIO_WORD тип Word.
" DIO_LONG тип Long.
DIO_VEC_SIZE : число портов, которое должно быть обслужено.
DIO_VEC_ADDR : виртуальный адрес массива пар (порт, значение)
в пространстве вызывающей стороны.
- возвращаемый результат:
OK: операции I/O были успешно выполнены.
EINVAL: ошибочный запрос или гранулярность.
E2BIG: размер вектора превышает максимум, который может быть обслужен.
EFAULT: адрес пар (порт, значение) ошибочный.
- библиотечные функции:

```
int sys_voutb(pvb_pair_t *pvb_vec, int vec_size);
int sys_voutw(pvw_pair_t *pvw_vec, int vec_size);
int sys_voutl(pvl_pair_t *pvl_vec, int vec_size);
int sys_vinb(pvb_pair_t *pvb_vec, int vec_size);
```

3 Так в оригинале, но, очевидно, текст сообщения «переполз» из другого вызова.

```
int sys_vinw(pvw_pair_t *pvw_vec, int vec_size);
int sys_vinl(pvl_pair_t *pvl_vec, int vec_size);
```

SYS_VIRCOPY: копировать данные используя виртуальную адресацию. Виртуальный адрес может принадлежать трём сегментам: LOCAL_SEG (сегменты кода, стека, данных), REMOTE_SEG (такие как RAM диск, видео память), и BIOS_SEG (BIOS вектора прерываний, BIOS область данных). Это наиболее общий системный вызов относительно копирования.
- параметры запроса:
CP_SRC_SPACE : сегмент источника.
CP_SRC_ADDR : виртуальный адрес источника.
CP_SRC_PROC_NR : номер процесса для процесса источника.
CP_DST_SPACE : сегмент назначения.
CP_DST_ADDR : виртуальный адрес назначения.
CP_DST_PROC_NR : номер процесса для процесса назначения.
CP_NR_BYTES : число байт для копирования.
- возвращаемый результат:
OK: копирование успешно выполнено.
EDOM: ошибочный счётчик копирования.
EFAULT: отображение виртуального в физический потеряно.
EPERM: нет прав использования PHYS_SEG.
EINVAL: некорректный тип сегмента, или номер процесса.
EPERM: только собственник REMOTE_SEG может копировать в него и из него.
- библиотечные функции[4]:

```
int sys_biosin(vir_bytes bios_vir, vir_bytes dst_vir,
               vir_bytes bytes);
int sys_biosout(vir_bytes src_vir, vir_bytes bios_vir,
                vir_bytes bytes);
int sys_datacopy(vir_bytes src_proc, vir_bytes src_vir,
                 dst_proc, dst_vir, vir_bytes bytes);
int sys_textcopy(vir_bytes src_proc, vir_bytes src_vir,
                 dst_proc, dst_vir, vir_bytes bytes);
int sys_stackcopy(vir_bytes src_proc, vir_bytes src_vir,
                  dst_proc, dst_vir, vir_bytes bytes);
int sys_vircopy(int src_proc, int src_seg, vir_bytes src_vir,
                int dst_proc, int dst_seg, vir_bytes dst_vir,
                phys_bytes bytes);
```

SYS_VIRVCOPY: копировать множество блоков данных используя виртуальную адресацию. Запрашиваемый вектор выбирается на вызывающей стороне, и каждый элемент обслуживается подобно отдельному запросу SYS_VIRCOPY. Копирование продолжается до тех пор, пока все элементы будут откопированы, или наступит ошибка.
- параметры запроса:
VCP_VEC_SIZE : число элементов запрашиваемого вектора.
VCP_VEC_ADDR : виртуальный адрес запрашиваемого вектора на вызывающей стороне.
- параметры ответа:
VCP_VEC_OK : число элементов успешно скопированных.
- возвращаемый результат:
OK: копирование успешно выполнено.

4 Здесь в прототипах функций у автора — путаница в параметрах.

EDOM: ошибочный счётчик копирования.
EFAULT: отображение виртуального в физический потеряно.
EPERM: нет прав использования PHYS_SEG.
EINVAL: вектор копирования слишком велик, некорректный тип сегмента,
или недопустимый процесс.
EPERM: только собственник REMOTE_SEG может копировать в него и из него.

- библиотечные функции:

```
int sys_virvcopy(vir_cp_req *copy_vec, int vec_size,
                 int *nr_ok);
```

Перевод: О.Цилюрик , ред. от 18.12.2009
Оригинал находится по адресу:
http://www.minix3.ru/docs/kernel-api.pdf
«MINIX 3 Kernel API»