MINIX 3 内核应用程序接口

约里特-赫德
<jnherder@cs.vu.nl>

2005 年 10 月 20 日

摘要

一般来说，内核调用允许系统进程请求内核服务，例如执行特权操作。本文档简要讨论了 MINIX 3 中内核调用的组织结构，并概述了所有内核调用。

内核调用的组织

内核调用是指将请求发送到内核，由内核任务处理。组装请求信息、将其发送给内核以及等待响应的细节都隐藏在系统库中，非常方便。该库的头文件是 *src/include/minix/syslib.h*，其实现可在 *src/lib/syslib* 中找到。

内核调用的实际执行由内核任务之一定义。与 MINIX 2 不同的是，CLOCK 任务不再接受系统调用。取而代之的是，现在所有调用都直接指向 SYSTEM 任务。假设程序进行了一次 sys call() 系统调用。按照惯例，该调用会转化为 SYS CALL 类型的请求消息，发送给内核任务 SYSTEM。SYSTEM 任务在名为 do call() 的函数中处理请求并返回结果。

内核调用号和处理程序函数的映射在 *src/kernel/system.c* 中的 SYSTEM 任务初始化过程中完成。处理程序函数的原型在 *src/kernel/system.h* 中声明。这些文件被编译成一个与内核链接的库 *src/kernel/system/system.a*。

内核调用编号及其请求和响应参数在 *src/include/minix/com.h* 中定义。不幸的是，MINIX 2 并没有遵循严格的命名方案。因此，许多信息

类型和参数在 MINIX 3 中被重新命名。内核调用现在都以 SYS 开头，属于同一内核调用的所有参数现在都有一个共同的前缀。

MINIX 3 内核调用概述

图 1 简要介绍了 MINIX 3 中的内核调用。最后一栏列出了与 MINIX 2 相比的每个调用的状态。

内核调用	目的	状态
	流程管理	
SYS_FORK	分叉进程；复制父进程	
SYS_EXEC	执行进程；初始化寄存器	
SYS_EXIT	退出用户进程；清除进程槽	U
SYS_NICE	更改用户进程的优先级	N
SYS_PRIVCTL	更改系统进程权限	N
系统跟踪	跟踪或控制进程执行	
	信号处理	
SYS_KILL	向进程发送信号	U
SYS_GETKSIG	检查待处理的内核信号	N
SYS_ENDKSIG	告知内核信号已处理 SYS SIGSEND	
	启动 POSIX 式信号处理器 SYS	
SIGRETURN	从 POSIX 样式信号返回	
	内存管理	
SYS_NEWMAP	安装新的或更新的内存映射	
SYS_SEGCTL	添加额外的远程内存段	N
SYS_MEMSET	将模式写入物理内存区	N
	复制数据	
SYS_UMAP	将虚拟地址映射到物理地址	U
SYS_VIRCOPY	使用虚拟寻址复制数据	U
SYS_PHYSCOPY	使用物理寻址复制数据	U
SYS_VIRVCOPY	处理有虚拟拷贝请求的矢量	U
SYS_PHYSVCOPY	用物理拷贝请求处理矢量	N
	设备 I/O	
SYS_DEVIO	读取或写入单个设备寄存器	N
SYS_SDEVIO	输入或输出整个数据缓冲区	N
SYS_VDEVIO	处理具有多个请求的矢量	N

SYS_IRQCTL	设置或重置中断策略	N
SYS_INT86	调用真实模式 BIOS	N
SYS_IOPENABLE	赋予进程 I/O 权限	N
	系统控制	
SYS_ABORT	终止 MINIX：关闭系统	U
SYS_GETINFO	获取系统信息或内核数据副本	N
	时钟功能	
SYS_SETALARM	设置或重置同步警报计时器	U
系统时间	获取启动后的进程时间和运行时间	U

图 1：此图概述了 MINIX 3 中的内核调用。 自 MINIX 2 以来，内核调用的状态说明为：新调用或已更新调用（即所有调用均已全面修订，并有小幅更新）。

MINIX 3 内核调用接口

MINIX 的内核调用接口详述如下。每个内核调用都指定了信息类型、目的、信息类型、请求和/或响应参数以及返回值。此外，还可能提供有关调用未来状态的附加备注。

Legenda

CONSTANT：已定义常量；表示请求类型或状态的数字 *PARAMETER*：信息参数；请求或响应信息中的一个字段 void sys call(参数)：系统库函数；内核调用的快捷方式

按字母顺序排列的概览

SYS_ABORT：关闭 MINIX 并返回启动监视器（如果可能）。PM、FS 和 TTY 使用此功能。正常中止通常由用户启动，例如通过 "shutdown "命令或键入 "Ctrl-Alt-Del"。如果 PM 或 FS 出现致命错误，MINIX 也将被关闭。

请求参数

ABRT_HOW：如何终止。*src/include/unistd.h* 中定义的值之一：

- RBT_HALT 停止 MINIX 并返回启动监视器。
- RBT_REBOOT 重启 MINIX。
- RBT_PANIC 发生内核恐慌。
- RBT_MONITOR 在启动监控器上运行指定代码。
- RBT_RESET 硬重置系统。

ABRT_MON_PROC：从中获取启动监控器参数的进程。

ABRT_MON_LEN：启动监控器参数的长度。

ABRT_MON_ADDR：参数的虚拟地址。

返回值

确定：关机程序已启动。

ENIVAL：进程编号无效。

错误：监控参数地址非法。

E2BIG：监控参数超出最大长度。

库函数

int sys_abort(int shutdown_status, ...)；

SYS_DEVIO：代表用户空间设备驱动程序执行设备 I/O。驱动程序可通过此调用请求读取或写入单个端口。另请参阅 SYS_SDEVIO 和 SYS_VDEVIO 内核调用。

请求参数

DIO_REQUEST：输入或输出。

- DIO_INPUT 从 DIO 端口读取数值。

- DIO_输出 将 DIO 值写入 DIO 端口。

DIO_TYPE：表示数值类型的标志。

- DIO_BYTE 字节类型。
- DIO_WORD 字类型。
- DIO_LONG 长类型。

DIO_端口：要读取或写入的端口。

DIO_VALUE：写入给定端口的值。仅适用于 DIO OUTPUT。

响应参数

DIO_VALUE：从给定端口读取的值。仅适用于 DIO INPUT。

返回值

OK（确定）：端口 I/O 已成功完成。

EINVAL：提供了无效的 DIO 请求或 DIO 类型。

库函数

```
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：结束内核信号。PM 使用此调用表示它已处理了通过 SYS GETKSIG 内核调用获得的映射中的内核信号。

响应参数

SIG_PROC：所涉及的程序。

返回值

EINVAL：进程无待处理信号或已退出。

确定：内核清除了所有待处理信号。

库函数

```
int sys_endksig(int proc_nr);
```

SYS_EXEC：在成功执行 exec() POSIX 调用后更新进程的寄存器。FS 将二进制映像复制到内存后，PM 会通知内核新寄存器的详细信息。

请求参数

PR_PROC_NR：执行程序的进程。

PR_STACK_PTR：新的堆栈指针。

PR_IP_PTR：新程序计数器。

PR_NAME_PTR：指向程序名称的指针。

返回值

好的：此调用始终成功。

库函数

int sys_exec(int proc, char *stack_ptr, char *prog_name, vir_bytes pc)；

SYS_EXIT：清除进程槽。通常由 PM 调用，在用户进程退出后进 行 清理。包括 PM 在内的系统进程也可以直接调用该函数退出自己的进程。

请求参数

PR_PROC_NR：如果调用者是 PM，则退出进程的槽号。使用 SELF 退出 PM.

返回值

好的：清理成功。

EINVAL：进程编号不正确。

EDONTREPLY：如果系统进程退出，该调用不会返回。

库函数

int sys_exit(int proc_nr)；

SYS_FORK：在内核进程表中分配一个新的（子）进程，并根据原型（父）进程对其进行初始化。PM 在自己的进程表中为子进程找到了一个空闲的进程槽，现在请求内核更新内核进程表。

请求参数

PR_PROC_NR：子进程表槽。

PR_PPROC_NR：父进程，即分叉的进程。

返回值

确定：成功分配了一个新的进程时隙。

EINVAL： 父进程编号或正在使用的子进程槽无效。

库函数

int sys_fork(int parent_proc_nr, int child_proc_nr)；

SYS_GETINFO：获取内核数据结构的副本。该调用支持需要某些系统信息的用户空间设备驱动程序和服务器。

请求参数

I_REQUEST：请求的系统信息类型。

- GET_IMAGE 复制引导图像表。
- GET_IRQHOOKS 复制包含中断钩子的表格。
- GET_KINFO 复制内核信息结构。

- GET_KMESSAGES 复制内核诊断信息缓冲区。
- GET_LOCKTIMING 复制锁定时间--如果设置了 DEBUG TIME LOCKS。
- GET_MACHINE 复制系统环境。
- GET_MONPARAMS 复制启动监视器设置的参数。
- GET_PRIVTAB 复制系统权限表。
- GET_PROCTAB 复制整个内核进程表。
- GET_PROC 复制单个进程表槽。
- GET_RANDOMNESS 复制内核事件收集的随机性。

- GET_SCHEDINFO 复制就绪队列和流程表。

I_VAL_PTR：信息应复制到的虚拟地址。

I_VAL_LEN：调用者可处理的最大长度。

I_VAL_PTR2：可选，第二个地址。复制调度数据时使用。

I_VAL_LEN2 : 可选，第二个长度。重载用于进程编号。

返回值

确定：信息请求成功。

错误：检测到非法内存地址。

E2BIG：请求的数据超出呼叫方提供的最大值。

库函数

```
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：检查是否存在必须发出信号的进程。PM 在收到有内核信号待处理的通知后会重复执行此操作。

响应参数

SIG_PROC：返回下一个有待处理信号的进程或无信号。

SIG_MAP：包含待处理内核信号的位图。

返回值

好的：此调用始终成功。

库函数

```
int sys_getksig(int *proc_nr, sigset_t *sig_map);
```

SYS_INT86：代表用户空间设备驱动程序制作实模式 BIOS。它暂时从 32 位保护模式切换到 16 位实数模式，以访问 BIOS 调用。它用于支持 BIOS WINI 设备驱动程序。

请求参数

INT86、*REG86*：呼叫方的请求地址。

返回值

OK：BIOS 调用成功完成。

错误：请求地址无效。

库函数

-

SYS_IOPENABLE：为给定进程启用 CPU 的 I/O 权限级别位，使其可以直接在用户空间执行 I/O。

请求参数

PROC_NR：赋予 I/O 权限的进程。

返回值

好的：总是成功。

库函数

-

SYS_IRQCTL：设置或重置指定 IRQ 线路的硬件中断策略，并启用或禁用该线路的中断。该调用允许用户空间设备驱动程序抓取一个钩子，供内核通用中断处理程序使用。内核中断处理程序只是通过 HARD INT 消息通知驱动程序中断，并在策略允许的情况下重新启用 IRQ 线路。通知信息将包含调用者作为参数提供的 "id"。一旦制定了策略，驱动程序就可以启用或禁用中断。

请求参数

IRQ_REQUEST：要执行的中断控制请求。

- IRQ_SETPOLICY 为通用中断处理程序设置中断策略。
- IRQ_RMPOLICY 删除先前设置的中断策略。
- IRQ_ENABLE 启用给定 IRQ 线路的 IRQ。
- IRQ_DISABLE 禁用给定 IRQ 线路的 IRQ。

IRQ_VECTOR：必须控制的 IRQ 线路。

IRQ_POLICY：带标志的位图，表示 IRQ 策略。

IRQ_HOOK_ID：当设置策略时，它提供中断时发送给调用者的索引。对于其他请求，它是内核返回的内核挂钩标识符。

响应参数

IRQ_HOOK_ID：与驱动程序相关的内核挂钩标识符。

返回值

EINVAL：无效请求、IRQ 线路、挂钩 ID 或进程编号。EPERM：只有挂钩所有者才能切换中断或释放挂钩。ENOSPC：找不到空闲的 IRQ 挂钩。

确定：请求已成功处理。

库函数

```
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：代表系统服务器向一个进程发出信号。系统进程可通过此调用向另一个进程发出信号。内核会将待处理信号通知 PM，以便进一步处理。(请注意，kill() POSIX 调用是由 PM 直接处理的）。PM 使用此调用间接向系统进程发送信号信息。当使用 sigaction() POSIX 调用设置了特殊 SIG MESS 信号处理器的系统进程收到信号时，就会发生这种情况。

请求参数

SIG_PROC_NR：要发出信号的进程。

SIG_NUMBER：信号编号。范围从 0 到 NSIG。

返回值

好的：呼叫成功。

EINVAL：非法进程或信号编号。

EPERM：无法向内核任务发送信号。PM 无法向用户进程发送通知消息。

库函数

int sys_kill(int proc_nr, int sig_nr)；

SYS_MEMSET：将 4 字节模式写入指定内存区域。该调用由 PM 在 exec() POSIX 调用中用于将 BSS 段清零。出于性能考虑，要求内核完成这项工作。

请求参数

MEM_PTR：内存区域的物理基地址。*MEM_COUNT*：以字节为单位的内存区域长度。*MEM_PATTERN*：要写入的 4 字节模式。

返回值

好的：呼叫总是成功。

库函数

int sys_memset(long pattern, phys_bytes base, phys_bytes length)；

SYS_NEWMAP：为新分叉的进程安装新的内存映射，或在进程的内存映射发生变化时安装新的内存映射。内核会从 PM 获取新的内存映射，并更新其数据结构。

请求参数

PROC_NR：为该流程安装新地图。

PR_MEM_PTR：指向 PM 处内存映射的指针。

返回值

确定：新地图已成功安装。

错误：新内存映射的地址不正确。

无效：进程编号无效。

库函数

int sys_newmap(int proc_nr, struct mem_map *ptr)；

SYS_NICE：更改进程的优先级。方法是在 PRIO_MIN（负值）和 PRIO_MAX（正值）之间传递一个 nice 值。漂亮值为 0 时，优先级将重置为默认值。

请求参数

PROC_NR：应改变优先级的程序

PR_PRIORITY：流程优先级的新值

返回值

确定：新优先级已成功设置。

无效：进程编号或优先级无效。

EPERM：无法更改内核任务的优先级。

库函数

int sys_nice(int proc_nr, int priority)；

SYS_PHYSCOPY：使用物理地址复制数据。源地址和/或目标地址可以是虚拟地址，如 SYS VIRCOPY，但 PHYS SEG 可以接受任意物理地址。

请求参数

CP_SRC_SPACE：信号源段。

CP_SRC_ADDR： 虚拟源地址

CP_SRC_PROC_NR：源进程的进程编号。

CP_DST_SPACE：目的地航段。

CP_DST_ADDR： 虚拟目标地址

CP_DST_PROC_NR： 目标进程的进程编号。

CP_NR_BYTES：要复制的字节数。

返回值

好的：复制完成。

EDOM：无效副本计数。

EFAULT： 虚拟到物理映射失败。

错误：程序段类型或程序编号不正确。

EPERM：只有 REMOTE SEG 的所有者才能向其复制或从其复制。

库函数

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：使用物理寻址复制多个数据块。从调用方获取请求向量，并像处理常规 SYS PHYSCOPY 请求一样处理每个元素。复制将持续进行，直到处理完所有元素或出现错误。

请求参数

VCP_VEC_SIZE：请求向量中的元素个数。

VCP_VEC_ADDR：请求向量在调用方的虚拟地址。

响应参数

VCP-NR ΘK：成功复制的元素数量。

返回值

好的：复制完成。

EDOM：无效副本计数。

EFAULT： 虚拟到物理映射失败。

EINVAL：复制矢量过大、段不正确或进程无效。

EPERM：只有 REMOTE SEG 的所有者才能向其复制或从其复制。

库函数

int sys_physvcopy(phys_cp_req *copy_vec, int vec_size, int *nr_ok)；

SYS_PRIVCTL：获取私有权限结构并更新进程的权限。用于动态启动系统服务。

请求参数

CTL_PROC_NR：应更新权限的进程。

返回值

好的：呼叫成功。

无效：进程编号无效。

ENOSPC：未找到自由权限结构。

发言

将对该系统调用进行扩展，以便为必须动态加载的服务器或设备驱动程序提供更好的支持和安全检查。这是未来的工作。

库函数

-

SYS_SDEVIO：代表用户空间设备驱动程序执行设备 I/O。请注意，该调用仅支持字节和字粒度。驱动程序可以请求整个缓冲区的输入或输出。另请参阅 SYS DEVIO 和 SYS VDEVIO 内核调用。

请求参数

DIO_REQUEST：输入或输出。

- DIO_INPUT 从 DIO 端口读取数值。
- DIO_输出 将 DIO 值写入 DIO 端口。

DIO_TYPE：表示数值类型的标志。

- DIO_BYTE 字节类型。
- DIO_WORD 字类型。

DIO_PORT：要读取或写入的端口。*DIO PROC NR*：缓冲区所在的进程编号。*DIO VEC ADDR*：缓冲区的虚拟地址。

DIO_VEC SIZE：输入或输出的元素数目。

响应参数

DIO_VALUE：从给定端口读取的值。仅适用于 DIO INPUT。

返回值

OK（确定）：端口 I/O 已成功完成。EINVAL：请求或端口粒度无效。EPERM：无法为内

核任务执行 I/O。

错误：缓冲区虚拟地址无效。

库函数

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：向进程的 LDT 及其远程内存映射添加内存段。调用会返回一个选择器和偏移量，可用于直接访问远程段，以及远程内存映射的索引，可用于 SYS VIRCOPY 内核调用。

请求参数

SEG_PHYS：网段的物理基地址。

SEG_SIZE（段码大小）：段码大小。

响应参数

SEG_INDEX：远程内存映射的索引。

SEG_SELECT：用于 LDT 输入的分段选择器。

SEG_OFFSET：数据段内的偏移。零，除非使用 4K 粒度。

返回值

ENOSPC：远程内存映射和 LDT 中没有空闲插槽。

确定：段描述符已成功添加。

库函数

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

SYS_SIGRETURN：从 POSIX 式信号处理器返回。PM 要求内核在信号进程恢复执行前将事情安排妥当。另请参阅 SYS SIGSEND 内核调用，它将信号上下文帧推入堆栈。

请求参数

SIG_PROC：表示发出信号的进程。

SIG_CTXT PTR：用于 POSIX 风格信号处理的上下文结构指针。

响应参数

SIG_PROC：返回下一个有待处理信号的进程或无信号。

返回值

OK（确定）：信号处理操作成功执行。

无效：进程编号或上下文结构无效。

错误：上下文结构地址无效，或无法复制信号帧。

库函数

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

SYS_SIGSEND：通过将上下文结构放入堆栈，代表 PM 向进程发出信号。内核会获取该结构，对其进行初始化，并将其复制到用户堆栈。

请求参数

SIG_PROC：表示发出信号的进程。

SIG_CTXT_PTR：用于 POSIX 风格信号处理的上下文结构指针。

响应参数

SIG_PROC：返回下一个有待处理信号的进程或无信号。

返回值

OK（确定）：信号处理操作成功执行。

无效：进程编号无效。

EPERM：无法向内核任务发出信号。

错误：上下文结构地址无效，或无法复制信号帧。

库函数

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

SYS_SETALARM：设置或重置同步警报计时器。计时器到期时，将向调用者发送一条以当前运行时间为参数的 SYN ALARM 通知消息。只有系统进程才能请求同步警报。

请求参数

alrm exp time：该警报的绝对或相对过期时间（以 ticks 为单位）。

alrm abs time：如果过期时间相对于当前运行时间，则为零。

响应参数

ALRM TIME LEFT（剩余时间）：上一个闹钟的剩余时间。

返回值

OK（确定）：警报设置成功。

EPERM：用户进程不能请求警报。

库函数

int sys_setalarm(clock_t expire_time, int abs_flag)；

SYS_TIMES（系统时间）：获取内核启动后的运行时间和进程执行时间。

请求参数

T_PROC NR：获取时间信息的进程，或无。

响应参数

T_USER TIME（用户时间）：进程的用户时间，如果是有效数字，单位为 ticks。

t 系统时间：进程的系统时间（以 ticks 为单位），如果是有效数字。

T_BOOT TICKS：MINIX 启动后的刻度数。

返回值

好的：总是成功。

库函数

int sys_times(int proc_nr, clock_t *ptr)；

SYS_TRACE：监视或控制给定进程的执行。处理 ptrace() 系统调用支持的调试命令。

请求参数

CTL_REQUEST：跟踪请求。

- T_STOP 停止进程。
- T_GETINS 从指令空间返回值。
- T_GETDATA 从数据空间返回值。
- T_GETUSER 从用户进程表中返回值。

- T_SETINS 设置指令空间的值。
- T_SETDATA 设置数据空间的值。
- T_SETUSER 设置用户进程表中的值。
- T_RESUME 恢复执行。
- T_STEP 设置跟踪位。

CTL_PROC_NR：正在跟踪的进程编号。*CTL_ADDRESS*：被跟踪进程空间中的虚拟地址。*CTL_DATA*（*数据*要写入的数据。

响应参数

CTL_DATA：返回数据。

返回值

确定：跟踪操作成功。

EIO：设置或获取值失败。

EINVAL：不支持跟踪请求。

PERM：只能跟踪用户进程。

库函数

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

SYS_UMAP：将虚拟地址映射到物理地址并返回物理地址。虚拟地址可以是本地 SEG、远程 SEG 或 BIOS SEG。可以传递以字节为单位的偏移量，以验证该地址是否也在段内。

请求参数

CP_SRC_PROC_NR：地址相关的流程编号。

CP_SRC_SPACE：分段标识符。

CP_SRC_ADDR：段内偏移。

CP_NR_BYTES：起始字节数。

响应参数

CP_DST_ADDR：映射成功时的物理地址。

返回值

好的：复制完成。

EFAULT： 虚拟到物理映射失败。

错误：程序段类型或程序编号不正确。

发言

BIOS SEG 中的零地址返回 EFAULT，而事实上 BIOS 中断向量的第零地址是一个有效地址。

库函数

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

SYS_VDEVIO：代表用户进程执行一系列设备 I/O。该调用接受一个指向（端口、值）对数组的指针，该数组将被一次性处理。硬件中断会被暂时禁用，以防止 I/O 调用被中断。另请参阅 SYS DEVIO 和 SYS SDEVIO。

请求参数

DIO_REQUEST：输入或输出。

- DIO_INPUT 从 DIO 端口读取数值。
- DIO_输出 将 DIO 值写入 DIO 端口。

DIO_TYPE：表示数值类型的标志。

- DIO_BYTE 字节类型。
- DIO_WORD 字类型。
- DIO_LONG 长类型。

DIO_VEC_SIZE：要处理的端口数。

DIO_VEC_ADDR：调用者空间中 (port,value)-pairs 的虚拟地址。

返回值

OK（确定）：端口 I/O 已成功完成。

无效：请求或粒度无效。

E2BIG： 矢量大小超出可处理的最大值。

错误： (port,value)-pairs 的地址有误。

库函数

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

SYS_VIRCOPY：使用虚拟寻址复制数据。虚拟可分为三段：本地 SEG（文本、堆栈、数据段）、远程 SEG（如 RAM 磁盘、视频存储器）和 BIOS SEG（BIOS 中断向量、BIOS 数据区）。这是与复制有关的最常见系统调用。

请求参数

CP_SRC_SPACE：信号源段。

CP_SRC_ADDR： 虚拟源地址

CP_SRC_PROC_NR：源进程的进程编号。

CP_DST_SPACE：目的地航段。

CP_DST_ADDR： 虚拟目标地址

CP_DST_PROC_NR： 目标进程的进程编号。

CP_NR_BYTES：要复制的字节数。

返回值

好的：复制完成。

EDOM：无效副本计数。

EFAULT： 虚拟到物理映射失败。

EPERM：不允许使用 PHYS SEG。 -

EINVAL：段类型或进程编号不正确。

EPERM：只有 REMOTE SEG 的所有者才能向其复制或从其复制。

库函数

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：使用虚拟寻址复制多个数据块。从调用方获取请求向量，并像处理常规 SYS VIRCOPY 请求一样处理每个元素。复制将持续进行，直到处理完所有元素或出现错误。

请求参数

VCP_VEC_SIZE：请求向量中的元素个数。

VCP_VEC_ADDR：请求向量在调用方的虚拟地址。

响应参数

VCP_VEC_OK：成功复制的元素数量。

返回值

好的：复制完成。

EDOM：无效副本计数。

EFAULT： 虚拟到物理映射失败。

EPERM：不允许使用 PHYS SEG。

EINVAL：复制矢量过大、段不正确或进程无效。

EPERM：只有 REMOTE SEG 的所有者才能向其复制或从其复制。

库函数

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