NAME

_syscall − invoking a system call without library support (OBSOLETE)

SYNOPSIS

#include <linux/unistd.h>

A _syscall macro

desired system call

DESCRIPTION

The important thing to know about a system call is its prototype. You need to know how many arguments, their types, and the function return type. There are seven macros that make the actual call into the system easier. They have the form:

```
_syscallX(type,name,type1,arg1,type2,arg2,...)
```

where

X is 0–6, which are the number of arguments taken by the system call

type is the return type of the system call

name is the name of the system call

typeN is the Nth argument's type

argN is the name of the Nth argument

These macros create a function called *name* with the arguments you specify. Once you include the _syscall() in your source file, you call the system call by *name*.

FILES

/usr/include/linux/unistd.h

STANDARDS

The use of these macros is Linux-specific, and deprecated.

NOTES

Starting around kernel 2.6.18, the _syscall macros were removed from header files supplied to user space. Use **syscall**(2) instead. (Some architectures, notably ia64, never provided the _syscall macros; on those architectures, **syscall**(2) was always required.)

The _syscall() macros *do not* produce a prototype. You may have to create one, especially for C++ users.

System calls are not required to return only positive or negative error codes. You need to read the source to be sure how it will return errors. Usually, it is the negative of a standard error code, for example, −*EPERM*. The _syscall() macros will return the result*r* of the system call when *r* is nonne gative, but will return −1 and set the variable *errno* to −*r* when *r* is negative. For the error codes, see **errno**(3).

When defining a system call, the argument types *must* be passed by-value or by-pointer (for aggregates like structs).

EXAMPLES

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <linux/unistd.h>       /* for _syscallX macros/related stuff */
#include <linux/kernel.h>       /* for struct sysinfo */

_syscall1(int, sysinfo, struct sysinfo *, info);

int
main(void)
{
    struct sysinfo s_info;
```

```
        int error;

        error = sysinfo(&s_info);
        printf("code error = %d\n", error);
        printf("Uptime = %lds\nLoad: 1 min %lu / 5 min %lu / 15 min %lu\n"
               "RAM: total %lu / free %lu / shared %lu\n"
               "Memory in buffers = %lu\nSwap: total %lu / free %lu\n"
               "Number of processes = %d\n",
               s_info.uptime, s_info.loads[0],
               s_info.loads[1], s_info.loads[2],
               s_info.totalram, s_info.freeram,
               s_info.sharedram, s_info.bufferram,
               s_info.totalswap, s_info.freeswap,
               s_info.procs);
        exit(EXIT_SUCCESS);
    }
```

Sample output

```
    code error = 0
    uptime = 502034s
    Load: 1 min 13376 / 5 min 5504 / 15 min 1152
    RAM: total 15343616 / free 827392 / shared 8237056
    Memory in buffers = 5066752
    Swap: total 27881472 / free 24698880
    Number of processes = 40
```

SEE ALSO

intro(2), **syscall**(2), **errno**(3)