NAME

kcmp - compare two processes to determine if they share a kernel resource

SYNOPSIS

#include <linux/kcmp.h>

int kcmp(pid_t *pid1*, **pid_t** *pid2*, **int** *type*,
unsigned long *idx1*, **unsigned long** *idx2*);

Note: There is no glibc wrapper for this system call; see NOTES.

DESCRIPTION

The **kcmp**() system call can be used to check whether the two processes identified by *pid1* and *pid2* share a kernel resource such as virtual memory, file descriptors, and so on.

The *type* argument specifies which resource is to be compared in the two processes. It has one of the following values:

KCMP_FILE
Check whether a file descriptor *idx1* in the process *pid1* refers to the same open file description (see open(2)) as file descriptor *idx2* in the process *pid2*.

KCMP_FILES
Check whether the process share the same set of open file descriptors. The arguments *idx1* and *idx2* are ignored.

KCMP_FS
Check whether the processes share the same filesystem information (i.e., file mode creation mask, working directory, and filesystem root). The arguments *idx1* and *idx2* are ignored.

KCMP_IO
Check whether the processes share I/O context. The arguments *idx1* and *idx2* are ignored.

KCMP_SIGHAND
Check whether the processes share the same table of signal dispositions. The arguments *idx1* and *idx2* are ignored.

KCMP_SYSVSEM
Check whether the processes share the same list of System V semaphore undo operations. The arguments *idx1* and *idx2* are ignored.

KCMP_VM
Check whether the processes share the same address space. The arguments *idx1* and *idx2* are ignored.

Note the **kcmp**() is not protected against false positives which may occur if tasks are running. One should stop tasks by sending **SIGSTOP** (see signal(7)) prior to inspection with this system call to obtain meaningful results.

RETURN VALUE

The return value of a successful call to **kcmp**() is simply the result of arithmetic comparison of kernel pointers (when the kernel compares resources, it uses their memory addresses).

The easiest way to explain is to consider an example. Suppose that *v1* and *v2* are the addresses of appropriate resources, then the return value is one of the following:

0 *v1* is equal to *v2*; in other words, the two processes share the resource.

1 *v1* is less than *v2*.

2 *v1* is greater than *v2*.

3 *v1* is not equal to *v2*, but ordering information is unavailable.

On error, -1 is returned, and *errno* is set appropriately.

kcmp() was designed to return values suitable for sorting. This is particularly handy if one needs to compare a large number of file descriptors.

ERRORS

EBADF
type is **KCMP_FILE** and *fd1* or *fd2* is not an open file descriptor.

EINVAL
type is invalid.

EPERM
Insufficient permission to inspect process resources. The **CAP_SYS_PTRACE** capability is required to inspect processes that you do not own.

ESRCH
Process *pid1* or *pid2* does not exist.

VERSIONS

The **kcmp**() system call first appeared in Linux 3.5.

CONFORMING TO

kcmp() is Linux-specific and should not be used in programs intended to be portable.

NOTES

Glibc does not provide a wrapper for this system call; call it using syscall(2).

This system call is available only if the kernel was configured with **CONFIG_CHECKPOINT_RESTORE**. The main use of the system call is for the checkpoint/restore in user space (CRIU) feature. The alternative to this system call would have been to expose suitable process information via the proc(5) filesystem; this was deemed to be unsuitable for security reasons.

See clone(2) for some background information on the shared resources referred to on this page.

SEE ALSO

clone(2), unshare(2)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.