Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'kcmp.2' command

$ man kcmp.2

KCMP(2) Linux Programmer's Manual KCMP(2)

NAME

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

source

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.

Permission to employ kcmp() is governed by ptrace access mode PTRACE_MODE_READ_REALCREDS checks against both pid1 and pid2; see ptrace(2).

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 descrip? tor idx2 in the process pid2. The existence of two file de? scriptors that refer to the same open file description can occur

as a result of dup(2) (and similar) fork(2), or passing file de?
scriptors via a domain socket (see unix(7)).

KCMP_FILES

Check whether the processes share the same set of open file de?
scriptors. The arguments idx1 and idx2 are ignored. See the
discussion of the CLONE_FILES flag in clone(2).

KCMP_FS

Check whether the processes share the same filesystem informa?
tion (i.e., file mode creation mask, working directory, and
filesystem root). The arguments idx1 and idx2 are ignored. See
the discussion of the CLONE_FS flag in clone(2).

KCMP_IO

Check whether the processes share I/O context. The arguments
idx1 and idx2 are ignored. See the discussion of the CLONE_IO
flag in clone(2).

KCMP_SIGHAND

Check whether the processes share the same table of signal dis?
positions. The arguments idx1 and idx2 are ignored. See the
discussion of the CLONE_SIGHAND flag in clone(2).

KCMP_SYSVSEM

Check whether the processes share the same list of System V sem?
aphore undo operations. The arguments idx1 and idx2 are ig?
nored. See the discussion of the CLONE_SYSVSEM flag in
clone(2).

KCMP_VM

Check whether the processes share the same address space. The
arguments idx1 and idx2 are ignored. See the discussion of the
CLONE_VM flag in clone(2).

KCMP_EPOLL_TFD (since Linux 4.13)

Check whether the file descriptor idx1 of the process pid1 is
present in the epoll(7) instance described by idx2 of the
process pid2. The argument idx2 is a pointer to a structure
where the target file is described. This structure has the

form:

```
struct kcmp_epoll_slot {
    __u32 efd;
    __u32 tfd;
    __u64 toff;
};
```

Within this structure, efd is an epoll file descriptor returned from epoll_create(2), tfd is a target file descriptor number, and toff is a target file offset counted from zero. Several different targets may be registered with the same file descriptor number and setting a specific offset helps to investigate each of them.

Note the kcmp() is not protected against false positives which may occur if the processes are currently running. One should stop the processes 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.

EFAULT The epoll slot addressed by idx2 is outside of the user's ad?
dress space.

EINVAL type is invalid.

ENOENT The target file is not present in epoll(7) instance.

EPERM Insufficient permission to inspect process resources. The
CAP_SYS_PTRACE capability is required to inspect processes that
you do not own. Other ptrace limitations may also apply, such
as CONFIG_SECURITY_YAMA, which, when /proc/sys/ker?
nel/yama/ptrace_scope is 2, limits kcmp() to child processes;
see ptrace(2).

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 secu?
rity reasons.

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

EXAMPLES

The program below uses kcmp() to test whether pairs of file descriptors
refer to the same open file description. The program tests different
cases for the file descriptor pairs, as described in the program out?
put. An example run of the program is as follows:

```
$ ./a.out
```

```
        Parent PID is 1144
        Parent opened file on FD 3
        PID of child of fork() is 1145
            Compare duplicate FDs from different processes:
                kcmp(1145, 1144, KCMP_FILE, 3, 3) ==> same
        Child opened file on FD 4
            Compare FDs from distinct open()s in same process:
                kcmp(1145, 1145, KCMP_FILE, 3, 4) ==> different
        Child duplicated FD 3 to create FD 5
            Compare duplicated FDs in same process:
                kcmp(1145, 1145, KCMP_FILE, 3, 5) ==> same
```

Program source

```
    #define _GNU_SOURCE
    #include <sys/syscall.h>
    #include <sys/wait.h>
    #include <sys/stat.h>
    #include <stdint.h>
    #include <stdlib.h>
    #include <stdio.h>
    #include <unistd.h>
    #include <fcntl.h>
    #include <linux/kcmp.h>
    #define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE); \
                            } while (0)
    static int
    kcmp(pid_t pid1, pid_t pid2, int type,
         unsigned long idx1, unsigned long idx2)
    {
        return syscall(SYS_kcmp, pid1, pid2, type, idx1, idx2);
    }
    static void
    test_kcmp(char *msg, pid_t pid1, pid_t pid2, int fd_a, int fd_b)
    {
```

```
    printf("\t%s\n", msg);
    printf("\t\tkcmp(%jd, %jd, KCMP_FILE, %d, %d) ==> %s\n",
            (intmax_t) pid1, (intmax_t) pid2, fd_a, fd_b,
            (kcmp(pid1, pid2, KCMP_FILE, fd_a, fd_b) == 0) ?
                    "same" : "different");
}
int
main(int argc, char *argv[])
{
    int fd1, fd2, fd3;
    char pathname[] = "/tmp/kcmp.test";
    fd1 = open(pathname, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    if (fd1 == -1)
        errExit("open");
    printf("Parent PID is %jd\n", (intmax_t) getpid());
    printf("Parent opened file on FD %d\n\n", fd1);
    switch (fork()) {
    case -1:
        errExit("fork");
    case 0:
        printf("PID of child of fork() is %jd\n", (intmax_t) getpid());
        test_kcmp("Compare duplicate FDs from different processes:",
                getpid(), getppid(), fd1, fd1);
        fd2 = open(pathname, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
        if (fd2 == -1)
            errExit("open");
        printf("Child opened file on FD %d\n", fd2);
        test_kcmp("Compare FDs from distinct open()s in same process:",
                getpid(), getpid(), fd1, fd2);
        fd3 = dup(fd1);
        if (fd3 == -1)
            errExit("dup");
        printf("Child duplicated FD %d to create FD %d\n", fd1, fd3);
```

```
            test_kcmp("Compare duplicated FDs in same process:",
                      getpid(), getpid(), fd1, fd3);
            break;
        default:
            wait(NULL);
        }
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

clone(2), unshare(2)

COLOPHON

This page is part of release 5.10 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 https://www.kernel.org/doc/man-pages/.