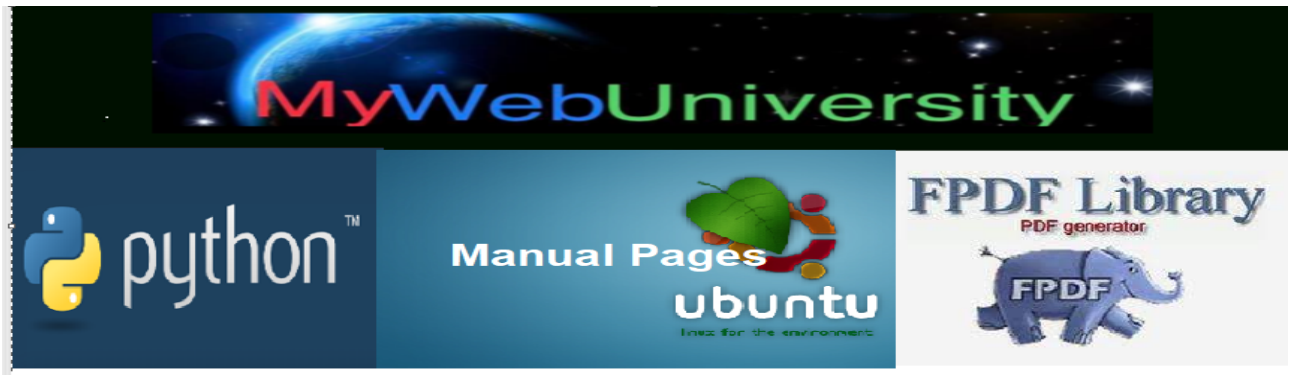

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'keyctl.2'

$ man keyctl.2

keyctl(2) System Calls Manual keyctl(2)

NAME

keyctl - manipulate the kernel's key management facility

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <linux/keyctl.h>  /* Definition of KEY* constants */
#include <sys/syscall.h>   /* Definition of SYS_* constants */
#include <unistd.h>
long syscall(SYS_keyctl, int op, ...);
```

DESCRIPTION

keyctl() allows user-space programs to perform key manipulation.

The operation performed by keyctl() is determined by the value of the op argument. Each of these operations is wrapped by the libkeyutils library (provided by the keyutils package) into individual functions (see keyctl(3)) to permit the compiler to check types.

The permitted values for op are:

KEYCTL_GET_KEYRING_ID(2const)

KEYCTL_JOIN_SESSION_KEYRING(2const)

KEYCTL_UPDATE(2const)

KEYCTL_REVOKE(2const)

KEYCTL_CHOWN(2const)

KEYCTL_SETPERM(2const)

KEYCTL_DESCRIBE(2const)

KEYCTL_CLEAR(2const)

KEYCTL_LINK(2const)

KEYCTL_UNLINK(2const)

KEYCTL_SEARCH(2const)

KEYCTL_READ(2const)

KEYCTL_INSTANTIATE(2const)

KEYCTL_INSTANTIATE_IOV(2const)

KEYCTL_NEGATE(2const)

KEYCTL_REJECT(2const)

KEYCTL_SET_REQKEY_KEYRING(2const)

KEYCTL_SET_TIMEOUT(2const)

KEYCTL_ASSUME_AUTHORITY(2const)

KEYCTL_GET_SECURITY(2const)

KEYCTL_SESSION_TO_PARENT(2const)

KEYCTL_INVALIDATE(2const)

KEYCTL_GET_PERSISTENT(2const)

KEYCTL_DH_COMPUTE(2const)

KEYCTL_RESTRICT_KEYRING(2const)

RETURN VALUE

For a successful call, the return value depends on the operation.

On error, -1 is returned, and errno is set to indicate the error.

ERRORS

EACCES The requested operation wasn't permitted.

EDQUOT The key quota for the caller's user would be exceeded by creating a key or linking it to the keyring.

EINVAL size of the string (including the terminating null byte) specified in arg3 (the key type) or arg4 (the key description) exceeded the limit (32 bytes and 4096 bytes re? spectively).

EKEYEXPIRED

An expired key was found or specified.

EKEYREJECTED

A rejected key was found or specified.

EKEYREVOKED

A revoked key was found or specified.

ENOKEY No matching key was found or an invalid key was specified.

ENOMEM One of kernel memory allocation routines failed during the execution of the syscall.

ENOTDIR

A key of keyring type was expected but the ID of a key with a different type was pro?

vided.

VERSIONS

A wrapper is provided in the libkeyutils library. (The accompanying package provides the <keyutils.h> header file.) However, rather than using this system call directly, you proba? bly want to use the various library functions mentioned in the descriptions of individual operations above.

STANDARDS

Linux.

HISTORY

Linux 2.6.10.

EXAMPLES

The program below provide subset of the functionality of the request-key(8) program provided by the keyutils package. For informational purposes, the program records various informa? tion in a log file.

As described in request_key(2), the request-key(8) program is invoked with command-line ar? guments that describe a key that is to be instantiated. The example program fetches and logs these arguments. The program assumes authority to instantiate the requested key, and then instantiates that key.

The following shell session demonstrates the use of this program. In the session, we com? pile the program and then use it to temporarily replace the standard request-key(8) program. (Note that temporarily disabling the standard request-key(8) program may not be safe on some systems.) While our example program is installed, we use the example program shown in re? quest_key(2) to request a key.

```
$ cc -o key_instantiate key_instantiate.c -lkeyutils;
$ sudo mv /sbin/request-key /sbin/request-key.backup;
$ sudo cp key_instantiate /sbin/request-key;
$ ./t_request_key user mykey somepayloaddata;
Key ID is 20d035bf
```

$ sudo mv /sbin/request-key.backup /sbin/request-key;

Looking at the log file created by this program, we can see the command-line arguments sup?
plied to our example program:

```
$ cat /tmp/key_instantiate.log;
Time: Mon Nov  7 13:06:47 2016
Command line arguments:
  argv[0]:            /sbin/request-key
  operation:          create
  key_to_instantiate: 20d035bf
  UID:                1000
  GID:                1000
  thread_keyring:     0
  process_keyring:    0
  session_keyring:    256e6a6
Key description:      user;1000;1000;3f010000;mykey
Auth key payload:     somepayloaddata
Destination keyring:  256e6a6
Auth key description: .request_key_auth;1000;1000;0b010000;20d035bf
```

The last few lines of the above output show that the example program was able to fetch:

? the description of the key to be instantiated, which included the name of the key (mykey);

? the payload of the authorization key, which consisted of the data (somepayloaddata) passed to request_key(2);

? the destination keyring that was specified in the call to request_key(2); and

? the description of the authorization key, where we can see that the name of the autho? rization key matches the ID of the key that is to be instantiated (20d035bf).

The example program in request_key(2) specified the destination keyring as KEY_SPEC_SES? SION_KEYRING. By examining the contents of /proc/keys, we can see that this was translated to the ID of the destination keyring (0256e6a6) shown in the log output above; we can also see the newly created key with the name mykey and ID 20d035bf.

```
$ cat /proc/keys | egrep 'mykey|256e6a6';
0256e6a6 I--Q---  194 perm 3f030000  1000  1000 keyring  _ses: 3
20d035bf I--Q---    1 perm 3f010000  1000  1000 user     mykey: 16
```

Program source

```
/* key_instantiate.c */

#include <errno.h>

#include <keyutils.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

#ifndef KEY_SPEC_REQUESTOR_KEYRING

#define KEY_SPEC_REQUESTOR_KEYRING      (-8)

#endif

int

main(int argc, char *argv[])

{

    int           akp_size;       /* Size of auth_key_payload */

    int           auth_key;

    char          dbuf[256];

    char          auth_key_payload[256];

    char          *operation;

    FILE          *fp;

    gid_t         gid;

    uid_t         uid;

    time_t        t;

    key_serial_t  key_to_instantiate, dest_keyring;

    key_serial_t  thread_keyring, process_keyring, session_keyring;

    if (argc != 8) {

        fprintf(stderr, "Usage: %s op key uid gid thread_keyring "

                        "process_keyring session_keyring\n", argv[0]);

        exit(EXIT_FAILURE);

    }

    fp = fopen("/tmp/key_instantiate.log", "w");
```

```
if (fp == NULL)
    exit(EXIT_FAILURE);
setbuf(fp, NULL);
t = time(NULL);
fprintf(fp, "Time: %s\n", ctime(&t));
/*
 * The kernel passes a fixed set of arguments to the program
 * that it execs; fetch them.
 */
operation = argv[1];
key_to_instantiate = atoi(argv[2]);
uid = atoi(argv[3]);
gid = atoi(argv[4]);
thread_keyring = atoi(argv[5]);
process_keyring = atoi(argv[6]);
session_keyring = atoi(argv[7]);
fprintf(fp, "Command line arguments:\n");
fprintf(fp, "  argv[0]:            %s\n", argv[0]);
fprintf(fp, "  operation:          %s\n", operation);
fprintf(fp, "  key_to_instantiate: %jx\n",
        (uintmax_t) key_to_instantiate);
fprintf(fp, "  UID:                %jd\n", (intmax_t) uid);
fprintf(fp, "  GID:                %jd\n", (intmax_t) gid);
fprintf(fp, "  thread_keyring:     %jx\n",
        (uintmax_t) thread_keyring);
fprintf(fp, "  process_keyring:    %jx\n",
        (uintmax_t) process_keyring);
fprintf(fp, "  session_keyring:    %jx\n",
        (uintmax_t) session_keyring);
fprintf(fp, "\n");
/*
 * Assume the authority to instantiate the key named in argv[2].
 */
```

```
if (keyctl(KEYCTL_ASSUME_AUTHORITY, key_to_instantiate) == -1) {
    fprintf(fp, "KEYCTL_ASSUME_AUTHORITY failed: %s\n",
            strerror(errno));
    exit(EXIT_FAILURE);
}
/*
 * Fetch the description of the key that is to be instantiated.
 */
if (keyctl(KEYCTL_DESCRIBE, key_to_instantiate,
           dbuf, sizeof(dbuf)) == -1) {
    fprintf(fp, "KEYCTL_DESCRIBE failed: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}
fprintf(fp, "Key description:      %s\n", dbuf);
/*
 * Fetch the payload of the authorization key, which is
 * actually the callout data given to request_key().
 */
akp_size = keyctl(KEYCTL_READ, KEY_SPEC_REQKEY_AUTH_KEY,
                  auth_key_payload, sizeof(auth_key_payload));
if (akp_size == -1) {
    fprintf(fp, "KEYCTL_READ failed: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}
auth_key_payload[akp_size] = '\0';
fprintf(fp, "Auth key payload:     %s\n", auth_key_payload);
/*
 * For interest, get the ID of the authorization key and
 * display it.
 */
auth_key = keyctl(KEYCTL_GET_KEYRING_ID,
                  KEY_SPEC_REQKEY_AUTH_KEY);
if (auth_key == -1) {
```

```
    fprintf(fp, "KEYCTL_GET_KEYRING_ID failed: %s\n",
            strerror(errno));
    exit(EXIT_FAILURE);
}
fprintf(fp, "Auth key ID:          %jx\n", (uintmax_t) auth_key);
/*
 * Fetch key ID for the request_key(2) destination keyring.
 */
dest_keyring = keyctl(KEYCTL_GET_KEYRING_ID,
                      KEY_SPEC_REQUESTOR_KEYRING);
if (dest_keyring == -1) {
    fprintf(fp, "KEYCTL_GET_KEYRING_ID failed: %s\n",
            strerror(errno));
    exit(EXIT_FAILURE);
}
fprintf(fp, "Destination keyring:  %jx\n", (uintmax_t) dest_keyring);
/*
 * Fetch the description of the authorization key.  This
 * allows us to see the key type, UID, GID, permissions,
 * and description (name) of the key.  Among other things,
 * we will see that the name of the key is a hexadecimal
 * string representing the ID of the key to be instantiated.
 */
if (keyctl(KEYCTL_DESCRIBE, KEY_SPEC_REQKEY_AUTH_KEY,
           dbuf, sizeof(dbuf)) == -1)
{
    fprintf(fp, "KEYCTL_DESCRIBE failed: %s\n", strerror(errno));
    exit(EXIT_FAILURE);
}
fprintf(fp, "Auth key description: %s\n", dbuf);
/*
 * Instantiate the key using the callout data that was supplied
 * in the payload of the authorization key.
```

```
         */
        if (keyctl(KEYCTL_INSTANTIATE, key_to_instantiate,
                   auth_key_payload, akp_size + 1, dest_keyring) == -1)
        {
            fprintf(fp, "KEYCTL_INSTANTIATE failed: %s\n",
                    strerror(errno));
            exit(EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

keyctl(1), add_key(2), request_key(2), keyctl(3), recursive_key_scan(3), recursive_session_key_scan(3), capabilities(7), credentials(7), keyrings(7), keyutils(7), persistent-keyring(7), process-keyring(7), session-keyring(7), thread-keyring(7), user-keyring(7), user_namespaces(7), user-session-keyring(7), request-key(8)

The kernel source files under Documentation/security/keys/.