BAHIR DAR UNIVERSITY

FACULTY OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING (SED)

COURSE: Operating System and System Programming

INDIVIDUAL ASSIGNMENT

NAME	ID NUMBER
Dagmawi Tinsae	1306926

SUBMISSION DATE: 26/08/2014 E.C

Instructors Name: Wendemu Baye

Contents

System Call

In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system's kernel. System call provides the services of the operating system to the user programs via Application Program Interface (API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system.

While designing the application program, the programmer has to include desired system calls. This will help the user program get the operating system's services. Thus, the system call creates an interface between the user program and the operating system.

Services Provided by System Calls:

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection
6. Networking, etc.

Types of System Calls: There are 5 different categories of system calls –

1. Process control: end, abort, create, terminate, allocate and free memory.
2. File management: create, open, close, delete, read file etc.
3. Device management
4. Information maintenance
5. Communication

In this Module I try to give some information about

What the system call long keyctl(int cmd, ...) is.

Why we use the system call long keyctl(int cmd, ...).

How the system call long keyctl(int cmd, ...) work.

Brief description about the list of parameters in the system call long keyctl(int cmd, ...) is.

NAME
keyctl - Manipulate the kernel's key management facility

SYNOPSIS
#include <keyutils.h>

long keyctl(int cmd, ...);

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

cmd – command flag modifying syscall behavior.

... – additional arguments per cmd flag.

The operation performed by keyctl() is determined by the value of the operation argument.

Each of these operations is wrapped by the libkeyutils library into individual functions to permit the compiler to check types.

This system call is a nonstandard Linux extension.

keyctl() has a number of functions available:

KEYCTL_GET_KEYRING_ID
Ask for a keyring's ID.

KEY_SPEC_SESSION_KEYRING
This specifies the caller's session-specific keyring.

KEYCTL_JOIN_SESSION_KEYRING
Join or start named session keyring.

KEYCTL_UPDATE
```
#include <keyutils.h>

       long keyctl_update(key_serial_t key, const void *payload,size_t plen);
```
Update a key's data payload.

The arg2 argument (cast to key_serial_t) specifies the ID of the key to be updated. The arg3 argument (cast to void *) points to the new payload and arg4 (cast to size_t) contains the new payload size in bytes.

The caller must have write permission on the key specified and the key type must support updating. A negatively instantiated key can be positively instantiated with this operation. The arg5 argument is ignored.

KEYCTL_REVOKE

```
#include <keyutils.h>

       long keyctl_revoke(key_serial_t key);
```

Revoke the key with the ID provided in arg2 (cast to key_serial_t). The key is scheduled for garbage collection; it will no longer be findable, and will be unavailable for further operations. Further attempts to use the key will fail with the error EKEYREVOKED.

The caller must have write or setattr permission on the key.

The arguments arg3, arg4, and arg5 are ignored.

KEYCTL_CHOWN

```
#include <keyutils.h>

       long keyctl_chown(key_serial_t key, uid_t uid, gid_t gid);
```

Change the ownership (user and group ID) of a key.

The arg2 argument (cast to key_serial_t) contains the key ID. The arg3 argument (cast to uid_t) contains the new user ID (or -1 in case the user ID shouldn't be changed). The arg4 argument (cast to gid_t) contains the new group ID (or -1 in case the group ID shouldn't be changed).

The key must grant the caller setattr permission.

For the UID to be changed, or for the GID to be changed to a group the caller is not a member of, the caller must have the CAP_SYS_ADMIN capability.

If the UID is to be changed, the new user must have sufficient quota to accept the key. The quota deduction will be removed from the old user to the new user should the UID be hanged.

The arg5 argument is ignored. KEYCTL_SETPERM Set perms on a key.

KEYCTL_DESCRIBE

Describe a key.

KEYCTL_CLEAR

```
#include <keyutils.h>

       long keyctl_clear(key_serial_t keyring);
```

Clear the contents of (i.e., unlink all keys from) a keyring. The ID of the key (which must be of keyring type) is provided in arg2 (cast to key_serial_t).

The caller must have write permission on the keyring.

The arguments arg3, arg4, and arg5 are ignored.

KEYCTL_LINK

```
#include <keyutils.h>

       long keyctl_link(key_serial_t key, key_serial_t keyring);
```

Create a link from a keyring to a key.

The key to be linked is specified in arg2 (cast to key_serial_t); the keyring is specified in arg3.

The caller must have link permission on the key being added and write permission on the keyring.

The arguments arg4 and arg5 are ignored.

KEYCTL_UNLINK

```
#include <keyutils.h>

              long keyctl_unlink(key_serial_t key, key_serial_t keyring);
```

Unlink a key from a keyring.

The caller must have write permission on the keyring from which the key is being removed. If the last link to a key is removed, then that key will be scheduled for destruction.

The arguments arg4 and arg5 are ignored.

KEYCTL_SEARCH

```
#include <keyutils.h>

       long keyctl_search(key_serial_t keyring, const char *type,
       const char *description, key_serial_t destination);
```

Search for a key in a keyring tree, returning its ID and optionally linking it to a specified keyring.

The tree to be searched is specified by passing the ID of the head keyring in arg2 (cast to key_serial_t). The search is performed breadth-first and recursively.

The arg3 and arg4 arguments specify the key to be searched for: arg3 (cast as char *) contains the key type (a null-terminated character string up to 32 bytes in size, including the terminating null byte), and arg4 (cast as char *) contains the description of the key (a null-terminated character string up to 4096 bytes in size, including the terminating null byte).

The source keyring must grant search permission to the caller. When performing the recursive search, only keyrings that grant the caller search permission will be searched. Only keys with for which the caller has search permission can be found.

If the key is found, its ID is returned as the function result.

If the key is found and arg5 (cast to key_serial_t) is nonzero, then, subject to the same constraints and rules as KEYCTL_LINK, the key is linked into the keyring whose ID is specified in arg5. If the destination keyring specified in arg5 already contains a link to a key that has the same type and description, then that link will be displaced by a link to the key found by this operation.

Instead of valid existing keyring IDs, the source (arg2) and destination (arg5) keyrings can be one of the special keyring IDs listed under KEYCTL_GET_KEYRING_ID.

KEYCTL_READ

```
#include <keyutils.h>

        long keyctl_read(key_serial_t key, char *buffer,
        size_tbuflen);
```

Read a key or keyring's contents.

The ID of the key to be instantiated is provided in arg2 (cast to key_serial_t).

The key payload is specified in the buffer pointed to by arg3 (cast to void *); the size of that buffer is specified in arg4 (cast to size_t).

The payload may be a NULL pointer and the buffer size may be 0 if this is supported by the key type (e.g., it is a keyring).

The operation may be fail if the payload data is in the wrong format or is otherwise invalid.

If arg5 (cast to key_serial_t) is nonzero, then, subject to the same constraints and rules as KEYCTL_LINK, the instantiated key is linked into the keyring whose ID specified in arg5.

The caller must have the appropriate authorization key, and once the uninstantiated key has been instantiated, the authorization key is revoked. In other words, this operation is available only from a request-key-style program.

KEYCTL_INSTANTIATE

```
#include <keyutils.h>

       long keyctl_instantiate(key_serial_t key, const void *payload,
       size_t plen, key_serial_t keyring);
```

Instantiate an instantiated key with a specified payload.

The ID of the key to be instantiated is provided in arg2 (cast to key_serial_t).

The key payload is specified in the buffer pointed to by arg3 (cast to void *); the size of that buffer is specified in arg4 (cast to size_t).

The payload may be a NULL pointer and the buffer size may be 0 if this is supported by the key type (e.g., it is a keyring).

The operation may be fail if the payload data is in the wrong format or is otherwise invalid.

The caller must have the appropriate authorization key, and once the uninstantiated key has been instantiated, the authorization key is revoked. In other words, this operation is available only from a request-key-style program.

KEYCTL_NEGATE

```
#include <keyutils.h>

 long keyctl_negate(key_serial_t key, unsigned timeout,key_serial_t keyring);
```

Negatively instantiate an uninstantiated key.

This operation is equivalent to the call:

keyctl(KEYCTL_REJECT, arg2, arg3, ENOKEY, arg4);

The arg5 argument is ignored. KEYCTL_SET_REQKEY_KEYRING

Set default request-key keyring.

KEYCTL_SET_TIMEOUT

```
#include <keyutils.h>

       long keyctl_set_timeout(key_serial_t key, unsigned timeout);
```

Set a timeout on a key.

The ID of the key is specified in arg2 (cast to key_serial_t). The timeout value, in seconds from the current time, is specified in arg3 (cast to unsigned int). The timeout is measured against the realtime clock.

Specifying the timeout value as 0 clears any existing timeout on the key.

The caller must either have the setattr permission on the key or hold an instantiation authorization token for the key.

The key and any links to the key will be automatically garbage collected after the timeout expires. Subsequent attempts to access the key will then fail with the error KEYEXPIRED.

This operation cannot be used to set timeouts on revoked, expired, or negatively instantiated keys.

The arguments arg4 and arg5 are ignored.

KEY_SPEC_REQKEY_AUTH_KEY

This specifies the authorization key created by request_key and passed to the process it spawns to generate a key. This key is available only in a request-key-style program that was passed an authorization key by the kernel and ceases to be available once the requested key has been instantiated.

KEYCTL_ASSUME_AUTHORITY

```
#include <keyutils.h>

        long keyctl_assume_authority(key_serial_t key);
```

Assume (or divest) the authority for the calling thread to instantiate a key.

The arg2 argument (cast to key_serial_t) specifies either a nonzero key ID to assume authority, or the value 0 to divest authority.

If arg2 is nonzero, then it specifies the ID of an instantiated key for which authority is to be assumed. That key can then be instantiated using one of KEYCTL_INSTANTIATE, KEYCTL_INSTANTIATE_IOV, KEYCTL_REJECT, or KEYCTL_NEGATE. Once the key has been instantiated, the thread is automatically divested of authority to instantiate the key.

Authority over a key can be assumed only if the calling thread has present in its

keyrings the authorization key that is associated with the specified key. (In other words, the KEYCTL_ASSUME_AUTHORITY operation is available only from a request-key-style program.) The caller must have search permission on the authorization key.

If the specified key has a matching authorization key, then the ID of that key is returned. The authorization key can be read (KEYCTL_READ) to obtain the callout information passed to request_key.

If the ID given in arg2 is 0, then the currently assumed authority is cleared (divested), and the value 0 is returned. The arguments arg3, arg4, and arg5 are ignored.

KEYCTL_GET_SECURITY

```
#include <keyutils.h>

long keyctl_get_security(key_serial_t key, char *buffer,size_t buflen);

long keyctl_get_security_alloc(key_serial_t key, char **_buffer);
```

Get the LSM (Linux Security Module) security label of the specified key.

The ID of the key whose security label is to be fetched is specified in arg2. The security label (terminated by a null byte) will be placed in the buffer pointed to by arg3 argument (cast to char *); the size of the buffer must be provided in arg4 (cast to size_t).

If arg3 is specified as NULL or the buffer size specified in arg4 is too small, the full size of the security label string (including the terminating null byte) is returned as the function result, and nothing is copied to the buffer.

The caller must have view permission on the specified key.

The returned security label string will be rendered in a form appropriate to the LSM in force. For example, with SELinux, it may look like:
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023

If no LSM is currently in force, then an empty string is placed in the buffer.

The arg5 argument is ignored.

KEYCTL_SESSION_TO_PARENT

```
#include <keyutils.h>

long keyctl_session_to_parent();
```

Replace the session keyring to which the parent of the calling process subscribes with the session keyring of the calling process.

The keyring will be replaced in the parent process at the point where the parent next transitions from kernel space to user space.

The keyring must exist and must grant the caller link permission. The parent process must be single-threaded and have the same effective ownership as this process and must not be set-user-ID or set-group-ID. The UID of the parent process's existing session keyring (f it has one), as well as the UID of the caller's session keyring much match the caller's effective UID.

The fact that it is the parent process that is affected by this operation allows a program such as the shell to start a child process that uses this operation to change the shell's session keyring.

The arguments arg2, arg3, arg4, and arg5 are ignored.

KEYCTL_REJECT

```
#include <keyutils.h>

long keyctl_reject(key_serial_t key, unsigned timeout, unsigned error,
key_serial_t keyring);
```

Mark a key as negatively instantiated and set an expiration timer on the key. This operation provides a superset of the functionality of the earlier KEYCTL_NEGATE operation.

The ID of the key that is to be negatively instantiated is specified in arg2. The arg3 (cast to unsigned int) argument specifies the lifetime of the key, in seconds. The arg4 argument (cast to unsigned int) specifies the error to be returned when a search hits this key; typically, this is one of EKEYREJECTED, EKEYREVOKED, or EKEYEXPIRED.

If arg5 (cast to key_serial_t) is nonzero, then, subject to the same constraints and rules as KEYCTL_LINK, the negatively instantiated key is linked into the keyring whose ID is specified in arg5.

The caller must have the appropriate authorization key. In other words, this operation is available only from a request-key-style program.

The caller must have the appropriate authorization key, and once the uninstantiated key has been instantiated, the authorization key is revoked. In other words, this operation is available only from a request-key-style program.

KEYCTL_INSTANTIATE_IOV

```
#include <keyutils.h>

      long keyctl_instantiate_iov(key_serial_t key, const struct iovec
*payload_iov, unsigned ioc, key_serial_t keyring);
```

Instantiate an uninstantiated key with a payload specified via a vector of buffers.

This operation is the same as KEYCTL_INSTANTIATE, but the payload data is specified as an array of iovec structures:

```
struct iovec {
   void  *iov_base;    /* Starting address of buffer */
   size_t iov_len;     /* Size of buffer (in bytes) */
};
```

The pointer to the payload vector is specified in arg3 (cast as const struct iovec *). The number of items in the vector is specified in arg4 (cast as unsigned int).

The arg2 (key ID) and arg5 (keyring ID) are interpreted as for KEYCTL_INSTANTIATE.

KEYCTL_INVALIDATE

```
#include <keyutils.h>

        long keyctl_invalidate(key_serial_t key);
```

Mark a key as invalid.

The ID of the key to be invalidated is specified in arg2 (cast to key_serial_t).

To invalidate a key, the caller must have search permission on the key.

This operation marks the key as invalid and schedules immediate garbage collection.

The garbage collector removes the invalidated key from all keyrings and deletes the key when its reference count reaches zero. After this operation, the key will be ignored by all searches, even if it is not yet deleted.

Keys that are marked invalid become invisible to normal key operations immediately, though they are still visible in /proc/keys (marked with an 'i' flag) until they are actually removed.

The arguments arg3, arg4, and arg5 are ignored.

KEYCTL_GET_PERSISTENT

```
#include <keyutils.h>

        long keyctl_get_persistent(uid_t uid, key_serial_t keyring);
```

The user ID is specified in arg2 (cast to uid_t). If the value -1 is specified, the caller's real user ID is used. The ID of the destination keyring is specified in arg3 (cast to key_serial_t).

The caller must have the CAP_SETUID capability in its user namespace in order to fetch the persistent keyring for a user ID that does not match either the real or effective user ID of the caller.

If the call is successful, a link to the persistent keyring is added to the keyring whose ID was specified in arg3.

The caller must have write permission on the keyring.

The persistent keyring will be created by the kernel if it does not yet exist.

When the timeout be reached, the persistent keyring will be removed and everything it pins can then be garbage collected.

The arguments arg4 and arg5 are ignored.

KEYCTL_DH_COMPUTE

```
#include <keyutils.h>

        long keyctl_dh_compute(key_serial_t private, key_serial_t prime,
        key_serial_t base, char *buffer, size_t buflen);
```

Compute a Diffie-Hellman shared secret or public key, optionally applying key derivation function (KDF) to the result.

The arg2 argument is a pointer to a set of parameters containing serial numbers for three "user" keys used in the Diffie-Hellman calculation, packaged in a structure of the following form:

```
struct keyctl_dh_params {
    int32_t private; /* The local private key */
    int32_t prime; /* The prime, known to both parties */
    int32_t base;  /* The base integer: either a shared
                      generator or the remote public key */
};
```

Each of the three keys specified in this structure must grant the caller read permission. The payloads of these keys are used to calculate the Diffie-Hellman result as:

```
base ^ private mod prime
```

If the base is the shared generator, the result is the local public key. If the base is the remote public key, the result is the shared secret.

The arg3 argument (cast to char *) points to a buffer where the result of the calculation is placed. The size of that buffer is specified in arg4 (cast to size_t).

The buffer must be large enough to accommodate the output data, otherwise an error is returned. If arg4 is specified zero, in which case the buffer is not used and the operation returns the minimum required buffer size (i.e., the length of the prime).

If the arg5 argument is NULL, then the DH result itself is returned. Otherwise it is a pointer to a structure which specifies parameters of the KDF operation to be applied:

```
struct keyctl_kdf_params {
    char *hashname;
    char *otherinfo;
    __u32 otherinfolen;
    __u32 __spare[8];
};
```

Error

I can’t implement the system call the first reason was the header file **<keyutils.h>** was not know by the compiler to overcome this problem I install the package for this header file by the command **`sudo apt-get install libkeyutils-dev`** but still after this the code doesn’t run correctly I try to get other codes and also try to debug the codes I have already get but that was beyond my knowledge I try my best to do the assignment for 4 days this was my final output.

Conclusion On my work

In this module I try to give more emphasis on the system call long keyctl(int cmd, ...); including its sub uses and sub system calls. I try to include the most important uses of this system call it was a great time for to work on this project I was having a lot of doubt and misunderstanding about system calls by doing this project I get a clear idea on system calls in general and especially this specific system calls it was an individual assignment but we help each other on some difficulties that is how team work starts. I try to search as many codes as possible for over 4 days but the codes doesn’t work even though they didn’t work in the process I get more information and knowledge about operating systems in general and specifically about system calls. This project was the toughest project I have ever done thanks for giving me this hard assignment so that I can challenge my abilities to the limit.