



# OPERATING SYSTEM AND SYSTEM PROGRAMMING INDIVIDUAL ASSIGNMENT

---

TOPIC: SYSTEM CALL

PREPARED BY: *WINTANA GEBREHIWOT*

ID NUMBER: *1306934*

SUBMITTED TO: *MR. WENDIMU B*

E.C.

## TABLE OF CONTENTS

1.1 The definition, why and how the system call <i>key_serial_t</i> <i>add_key(const char *type, const char *description, const void</i> <i>*payload, size_t plen, key_serial_t keyring)</i> works-----	-----2
1.1.2 What are keyrings-----	-----2
1.1.2 Key types-----	-----2
1.1.3 The add_key syscall-----	-----3
1.2 List of parameters and flags and their definition-----	---4
1.3 CODE IMPLEMENTATION OF KEY_SPEC_SESSION_KEYRING-----	-----6
Reference-----	-----8

## 1.1 The definition, why and how the system call *key\_serial\_t add\_key(const char \*type, const char \*description, const void \*payload, size\_t plen, key\_serial\_t keyring)* works

### 1.1.1 What are keyrings

keyrings in kernel are key management and retention facility. The Linux key-management facility is primarily a way for various kernel components to retain or cache security data authentication keys, encryption keys, and other data in the kernel.

System call interfaces are provided so that user-space program can manage those objects and also use the facility for their own purposes. A library and some user-space utilities are provided to allow access to the facility.

### 1.1.2 Key types

The kernel provides several basic types of key

"Keyring": Keyrings are special keys which store a set of links to other keys (including other keyrings), analogous to a directory holding links to files. The main purpose of a keyring is to prevent other keys from being garbage collected because nothing refers to them.

Keyrings with descriptions (names) that begin with a period ('.') are reserved to the implementation.

"User": This is a general-purpose key type. The key is kept entirely within kernel memory. The payload may be read and updated by user-space applications. The payload for keys of this type is a blob of arbitrary data of up to 32,767 bytes.

The description may be any valid string, though it is preferred that it start with a colon-delimited prefix representing the service to which the key is of interest (for instance "afs:mykey").

"Ligon" (since Linux 3.3): This key type is essentially the same as "user", but it does not provide reading (i.e., the `keyctl(2)` `KEYCTL_READ` operation), meaning that the key payload is

never visible from user space. This is suitable for storing username password pairs that should not be readable from user space. The description of a "logon" key must start with a non- empty colon-delimited prefix whose purpose is to identify the service to which the key belongs. (Note that this differs from keys of the "user" type, where the inclusion of a prefix is recommended but is not enforced.)

"Big\_key" (since Linux 3.13): This key type is similar to the "user" key type, but it may hold a payload of up to 1 MB in size. This key type is useful for purposes such as holding Kerberos ticket caches.

The payload data may be stored in a tmpfs filesystem rather than in kernel memory, if the data size exceeds the overhead of storing the data in the filesystem. (Storing the data in a filesystem requires filesystem structures to be allocated in the kernel. The size of these structures determines the size threshold above which the tmpfs storage method is used.) Since Linux 4.8, the payload data is encrypted when stored in tmpfs, thereby preventing it from being written unencrypted into swap space.

### 1.1.3 The `add_key` syscall

**NAME** `add_key` - add a key to the kernel's key management facility

**SYNOPSIS** `#include <keyutils.h>`

**DESCRIPTION** `key_serial_t add_key(const char *type, const char *description, const void *payload, size_t plen, key_serial_t keyring);`

`Add_key()` creates or updates a key of the given type and description, instantiates it with the payload of length `plen`, attaches it to the nominated keyring, and returns the key's serial number. The key may be rejected if the provided data is in the wrong format or it is invalid in some other way. If the destination keyring already contains a key that matches the specified type and description, then, if the key type supports it, that key will be updated rather than a new key being created; if not, a new key (with a different ID) will be created and it will displace the link to the extant key from the keyring.

The destination keyring serial number may be that of a valid keyring for which the caller has write permission. Alternatively, it may be one of the following special keyring IDs.

`KEY_SPEC_THREAD_KEYRING`

`KEY_SPEC_PROCESS_KEYRING`

`KEY_SPEC_SESSION_KEYRING`

`KEY_SPEC_USER_KEYRING`

`KEY_SPEC_USER_SESSION_KEYRING`

**RETURN VALUE** On success, `add_key()` returns the serial number of the key it created or

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

## ERRORS

EACCES The keyring wasn't available for modification by the user.

EDQUOT The key quota for this user would be exceeded by creating this key or linking it to the keyring. EFAULT One or more of type, description, and payload points outside process's accessible address space.

EINVAL The size of the string (including the terminating null byte) specified in type or description exceeded the limit (32 bytes and 4096 bytes respectively).

EINVAL The payload data was invalid.

EINVAL type was "logon" and the description was not qualified with a prefix string of the form "service:".

EKEYEXPIRED the keyring has expired.

EKEYREVOKED the keyring has been revoked.

ENOKEY The keyring doesn't exist.

ENOMEM Insufficient memory to create a key.

EPERM The type started with a period ('.'). Key types that begin with a period are reserved to the implementation.

EPERM type was "keyring" and the description started with a period ('.'). Keyrings with descriptions (names) that begin with a period are reserved to the implementation. This system call first appeared in Linux 2.6.10. it is a nonstandard Linux extension.

## 1.2 List of parameters and flags and their definition

A key has the following attributes:

Serial number (ID): This is a unique integer handle by which a key is referred to in system calls. The serial number is sometimes synonymously referred as the key ID. Programmatically, key serial numbers are represented using the type `key_serial_t`.

Type: A key's type defines what sort of data can be held in the key, how the proposed content of the key will be parsed, and how the payload will be used. There are a number of general-purpose types available, plus some specialist types defined by specific kernel components.

Description (name): The key description is a printable string that is used as the search term for the key (in conjunction with the key type) as well as a display name. During searches, the description may be partially matched or exactly matched.

**Payload (data):** The payload is the actual content of a key. This is usually set when a key is created, but it is possible for the kernel to upcall to user space to finish the instantiation of a key if that key wasn't already known to the kernel when it was requested.

A key's payload can be read and updated if the key type supports it and if suitable permission is granted to the caller.

**Access rights:** Much as files do, each key has an owning user ID, an owning group ID, and a security label. Each key also has a set of permissions, though there are more than for a normal UNIX file, and there is an additional category— possessor— beyond the usual user, group, and other.

Note that keys are quota controlled, since they require unswappable kernel memory. The owning user ID specifies whose quota is to be debited.

**Expiration time:** Each key can have an expiration time set. When that time is reached, the key is marked as being expired and accesses to it fail with the error EKEYEXPIRED. If not deleted, updated, or replaced, then, after a set amount of time, an expired key is automatically removed (garbage collected) along with all links to it, and attempts to access the key fail with the error ENOKEY.

**Reference count:** Each key has a reference count. Keys are referenced by keyrings, by currently active users, and by a process's credentials. When the reference count reaches zero, the key is scheduled for garbage collection.

`key_serial_t add_key(const char *type, const char *description, const void *payload, size_t plen, key_serial_t keyring)`

Type – pointer to string with type of key.

Description – pointer to string with description of key

Payload – key to add

Plen – length of key

Keyring – serial number of keyring or special flag

**Keyrings:** As previously mentioned, keyrings are a special type of key that contain links to other keys (which may include other keyrings). Keys may be linked to by multiple keyrings. Keyrings may be considered as analogous to UNIX directories where each directory contains a set of hard links to files. Various operations (system calls) may be applied only to keyrings:

**Adding** A key may be added to a keyring by system calls that create keys. This prevents the new key from being immediately deleted when the system call releases its last reference to the key.

**Linking:** A link may be added to a keyring pointing to a key that is already known, provided this does not create a self-referential cycle.

Unlinking: A link may be removed from a keyring. When the last link to a key is removed, that key will be scheduled for deletion by the garbage collector.

Clearing: All the links may be removed from a keyring.

Searching: A keyring may be considered the root of a tree or subtree in which keyrings form the branches and non-keyrings the leaves. This tree may be searched for a key matching a particular type and description.

KEY\_SPEC\_THREAD\_KEYRING

This specifies the caller's thread-specific keyring

KEY\_SPEC\_PROCESS\_KEYRING

This specifies the caller's process-specific keyring

KEY\_SPEC\_SESSION\_KEYRING

This specifies the caller's session-specific keyring

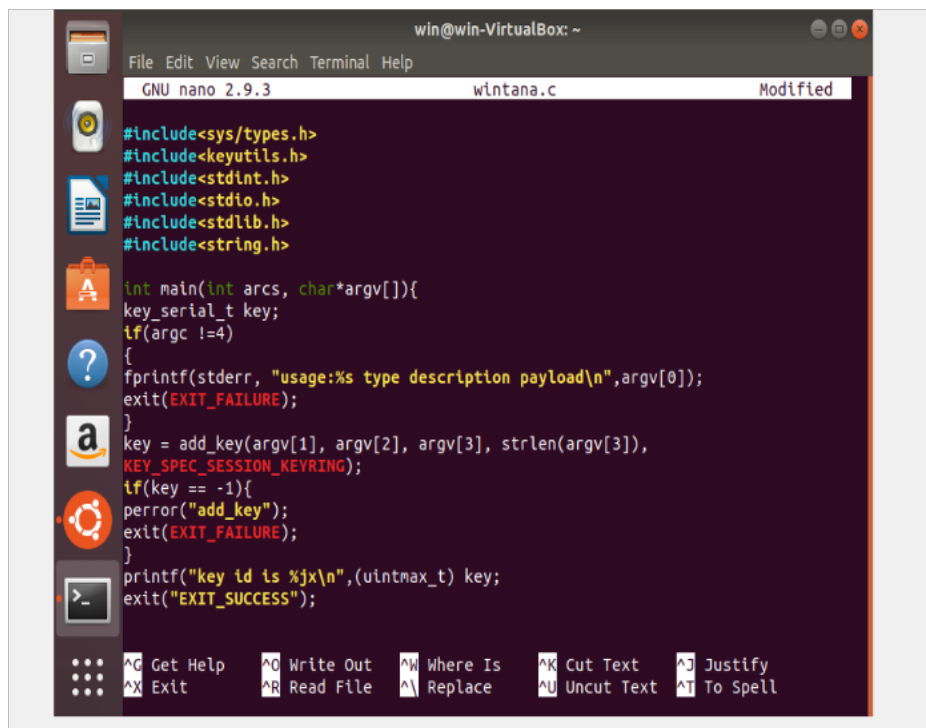
KEY\_SPEC\_USER\_KEYRING

This specifies the caller's UID-specific keyring

KEY\_SPEC\_USER\_SESSION\_KEYRING

This specifies the caller's UID-session keyring

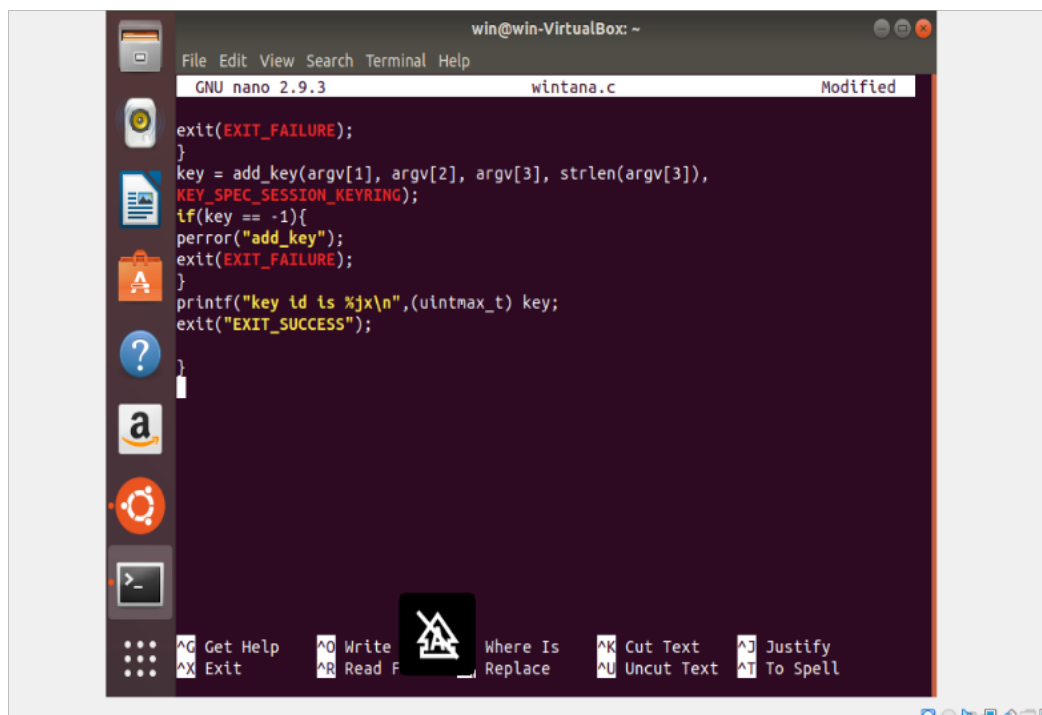
## **1.2 CODE IMPLEMENTATION OF KEY\_SPEC\_SESSION\_KEYRING**



```
win@win-VirtualBox: ~
File Edit View Search Terminal Help
GNU nano 2.9.3 wintana.c Modified

#include<sys/types.h>
#include<keyutils.h>
#include<stdint.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

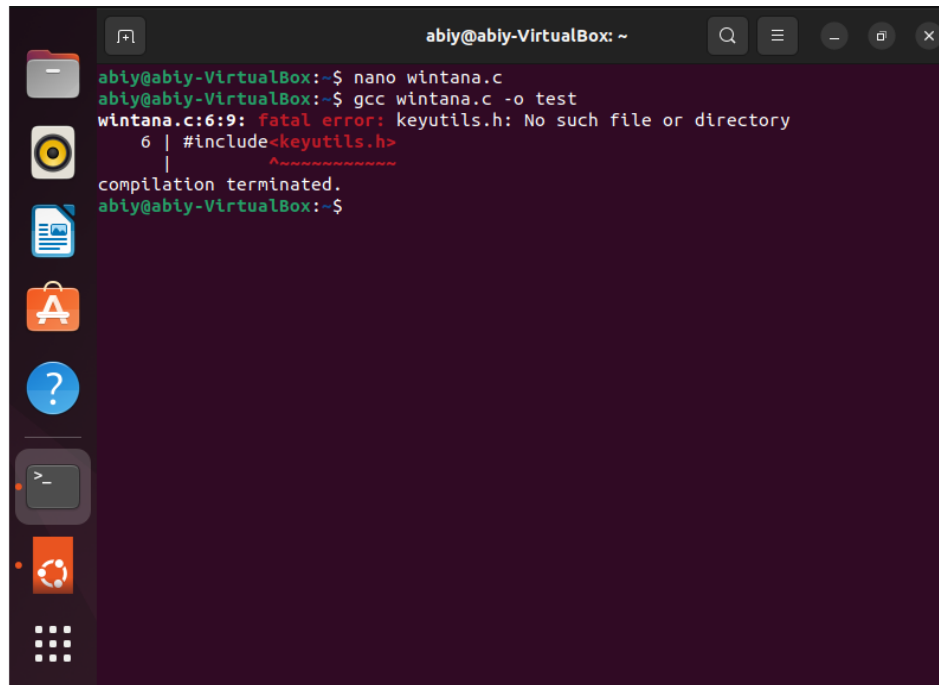
int main(int argc, char*argv[]){
    key_serial_t key;
    if(argc !=4)
    {
        fprintf(stderr, "usage:%s type description payload\n",argv[0]);
        exit(EXIT_FAILURE);
    }
    key = add_key(argv[1], argv[2], argv[3], strlen(argv[3]),
    KEY_SPEC_SESSION_KEYRING);
    if(key == -1){
        perror("add_key");
        exit(EXIT_FAILURE);
    }
    printf("key id is %jx\n",(uintmax_t) key);
    exit("EXIT_SUCCESS");
}
```



```
win@win-VirtualBox: ~
File Edit View Search Terminal Help
GNU nano 2.9.3 wintana.c Modified

exit(EXIT_FAILURE);
}
key = add_key(argv[1], argv[2], argv[3], strlen(argv[3]),
KEY_SPEC_SESSION_KEYRING);
if(key == -1){
    perror("add_key");
    exit(EXIT_FAILURE);
}
printf("key id is %jx\n",(uintmax_t) key);
exit("EXIT_SUCCESS");
}
```





The image shows a terminal window titled 'abiy@abiy-VirtualBox: ~'. The user has executed the following commands:

```
abiy@abiy-VirtualBox:~$ nano wintana.c
abiy@abiy-VirtualBox:~$ gcc wintana.c -o test
wintana.c:6:9: fatal error: keyutils.h: No such file or directory
    6 | #include<keyutils.h>
      | 
compilation terminated.
abiy@abiy-VirtualBox:~$
```

The error message indicates that the compiler cannot find the 'keyutils.h' header file. The terminal window has a dark purple background and a sidebar on the left with various application icons.

## REFERENCE

[https://linuxhint.com/list\\_of\\_linux\\_syscalls/](https://linuxhint.com/list_of_linux_syscalls/)

[https://man7.org/linux/man-pages/man2/add\\_key.2.html](https://man7.org/linux/man-pages/man2/add_key.2.html)

<https://man7.org/linux/man-pages/man7/keeyrings.7.html>