NAME

k_hasafs, **k_hasafs_recheck**, **k_pioctl**, **k_unlog**, **k_setpag**, **k_afs_cell_of_file**, **kafs_set_verbose**, **kafs_settoken_rxkad**, **kafs_settoken**, **krb_afslog**, **krb_afslog_uid**, **kafs_settoken5**, **krb5_afslog**, **krb5_afslog_uid** - AFS library

LIBRARY

AFS cache manager access library (libkafs5, -lkafs5)

SYNOPSIS

#include <kafs.h>

int
k_afs_cell_of_file(*const char *path, char *cell, int len*);

int
k_hasafs(*void*);

int
k_hasafs_recheck(*void*);

int
k_pioctl(*char *a_path, int o_opcode, struct ViceIoctl *a_paramsP, int a_followSymlinks*);

int
k_setpag(*void*);

int
k_unlog(*void*);

void
kafs_set_verbose(*void (*func)(void *, const char *, int), void **);

int
kafs_settoken_rxkad(*const char *cell, struct ClearToken *token, void *ticket, size_t ticket_len*);

int
kafs_settoken(*const char *cell, uid_t uid, CREDENTIALS *c*);

krb_afslog(*char *cell, char *realm*);

int
krb_afslog_uid(*char *cell, char *realm, uid_t uid*);

krb5_error_code
krb5_afslog_uid(*krb5_context context, krb5_ccache id, const char *cell, krb5_const_realm realm, uid_t uid*);

int
kafs_settoken5(*const char *cell, uid_t uid, krb5_creds *c*);

krb5_error_code
krb5_afslog(*krb5_context context, krb5_ccache id, const char *cell, krb5_const_realm realm*);

DESCRIPTION

k_hasafs() initializes some library internal structures, and tests for the presence of AFS in the kernel, none of the other functions should be called before **k_hasafs**() is called, or if it fails.

k_hasafs_recheck() forces a recheck if a AFS client has started since last time **k_hasafs**() or **k_hasafs_recheck**() was called.

kafs_set_verbose() set a log function that will be called each time the kafs library does something important so that the application using libkafs can output verbose logging. Calling the function *kafs_set_verbose* with the function argument set to NULL will stop libkafs from calling the logging function (if set).

kafs_settoken_rxkad() set rxkad with the *token* and *ticket* (that have the length *ticket_len*) for a given *cell*.

kafs_settoken() and **kafs_settoken5**() work the same way as **kafs_settoken_rxkad**() but internally converts the Kerberos 4 or 5 credential to a afs cleartoken and ticket.

krb_afslog(), and **krb_afslog_uid**() obtains new tokens (and possibly tickets) for the specified *cell* and *realm*. If *cell* is NULL, the local cell is used. If *realm* is NULL, the function tries to guess what realm to use. Unless you have some good knowledge of what cell or realm to use, you should pass NULL. **krb_afslog**() will use the real user-id for the ViceId field in the token, **krb_afslog_uid**() will use *uid*.

krb5_afslog(), and **krb5_afslog_uid**() are the Kerberos 5 equivalents of **krb_afslog**(), and **krb_afslog_uid**().

krb5_afslog(), **kafs_settoken5**() can be configured to behave differently via a **krb5_appdefault** option

afs-use-524 in *krb5.conf*. Possible values for afs-use-524 are:

yes use the 524 server in the realm to convert the ticket

no use the Kerberos 5 ticket directly, can be used with if the afs cell support 2b token.

local, 2b
convert the Kerberos 5 credential to a 2b token locally (the same work as a 2b 524 server should have done).

Example:

```
[appdefaults]
        SU.SE = { afs-use-524 = local }
        PDC.KTH.SE = { afs-use-524 = yes }
        afs-use-524 = yes
```

libkafs will use the libkafs as application name when running the **krb5_appdefault** function call.

The (uppercased) cell name is used as the realm to the **krb5_appdefault function.**

k_afs_cell_of_file() will in *cell* return the cell of a specified file, no more than *len* characters is put in *cell*.

k_pioctl() does a **pioctl**() system call with the specified arguments. This function is equivalent to **lpioctl**().

k_setpag() initializes a new PAG.

k_unlog() removes destroys all tokens in the current PAG.

RETURN VALUES

k_hasafs() returns 1 if AFS is present in the kernel, 0 otherwise. **krb_afslog**() and **krb_afslog_uid**() returns 0 on success, or a Kerberos error number on failure. **k_afs_cell_of_file**(), **k_pioctl**(), **k_setpag**(), and **k_unlog**() all return the value of the underlaying system call, 0 on success.

ENVIRONMENT

The following environment variable affect the mode of operation of **kafs**:

AFS_SYSCALL Normally, **kafs** will try to figure out the correct system call(s) that are used by AFS by

itself. If it does not manage to do that, or does it incorrectly, you can set this variable to the system call number or list of system call numbers that should be used.

EXAMPLES

The following code from **login** will obtain a new PAG and tokens for the local cell and the cell of the users home directory.

```
if (k_hasafs()) {
        char cell[64];
        k_setpag();
        if(k_afs_cell_of_file(pwd->pw_dir, cell, sizeof(cell)) == 0)
                krb_afslog(cell, NULL);
        krb_afslog(NULL, NULL);
}
```

ERRORS

If any of these functions (apart from **k_hasafs**()) is called without AFS being present in the kernel, the process will usually (depending on the operating system) receive a SIGSYS signal.

SEE ALSO

krb5_appdefault(3), krb5.conf(5)

Transarc Corporation, "File Server/Cache Manager Interface", *AFS-3 Programmer's Reference*, 1991.

FILES

libkafs will search for *ThisCell and TheseCells* in the following locations: */usr/vice/etc*, */etc/openafs*, */var/db/openafs/etc*, */usr/arla/etc*, */etc/arla*, and */etc/afs*

BUGS

AFS_SYSCALL has no effect under AIX.