kernel could only handle unicode, but the directory name / file name could be any language (means any encoding code). So kernel need NLS modules to support "Native Language Support".

For example, the following output is from my Linux box

```
walterzh@walterzh-Precision-T1650:~$ tree /lib/modules/`uname -r`/kernel/fs/nls
/lib/modules/4.4.1-040401-generic/kernel/fs/nls
├── mac-celtic.ko
├── mac-centeuro.ko
├── mac-croatian.ko
├── mac-cyrillic.ko
├── mac-gaelic.ko
├── mac-greek.ko
├── mac-iceland.ko
├── mac-inuit.ko
├── mac-romanian.ko
├── mac-roman.ko
├── mac-turkish.ko
├── nls_ascii.ko
├── nls_cp1250.ko
├── nls_cp1251.ko
├── nls_cp1255.ko
├── nls_cp737.ko
├── nls_cp775.ko
├── nls_cp850.ko
├── nls_cp852.ko
├── nls_cp855.ko
├── nls_cp857.ko
├── nls_cp860.ko
├── nls_cp861.ko
├── nls_cp862.ko
├── nls_cp863.ko
├── nls_cp864.ko
├── nls_cp865.ko
├── nls_cp866.ko
├── nls_cp869.ko
├── nls_cp874.ko
├── nls_cp932.ko
├── nls_cp936.ko
├── nls_cp949.ko
├── nls_cp950.ko
├── nls_euc-jp.ko
├── nls_iso8859-13.ko
├── nls_iso8859-14.ko
├── nls_iso8859-15.ko
├── nls_iso8859-1.ko
├── nls_iso8859-2.ko
├── nls_iso8859-3.ko
├── nls_iso8859-4.ko
├── nls_iso8859-5.ko
├── nls_iso8859-6.ko
├── nls_iso8859-7.ko
├── nls_iso8859-9.ko
├── nls_koi8-r.ko
├── nls_koi8-ru.ko
├── nls_koi8-u.ko
└── nls_utf8.ko
```

每个nls module做4件事情(其实真正有意义的是2件)

```c
struct nls_table {
    const char *charset;
    const char *alias;
    int (*uni2char) (wchar_t uni, unsigned char *out, int boundlen);
    int (*char2uni) (const unsigned char *rawstring, int boundlen,
                wchar_t *uni);
    const unsigned char *charset2lower;
    const unsigned char *charset2upper;
    struct module *owner;
    struct nls_table *next;
};
```

这4件事情对应到struct nls_table的4个callback functions

callback	description
uni2char	unicode转char
char2uni	char转unicode
charset2lower	转小写
charset2upper	转大写

这里 `char` 的概念不是c语言中的char，而是只某种编码某个特定character。比如 `周龙` ，这里是2个characters，对应到gb2312 encoding code是0xD6DC, 0xC1FA (GB2312是双字节编码)
`周` 的unicode code point is 0x5468
`龙` 的unicode code point is 0x9f99

`char2uni` callback负责把GB2312 encoding char转换成unicode，而uni2char callback则反之。

char2uni('周') ==> 0x5468
uni2char('龙') ==> 0xC1FA

汉字	GB2312 encoding code	unicode code point
周	0xD6DC	0x5468
龙	0xC1FA	0x9f99

至于charset2lower and charset2upper 2 callbacks则对绝大部分编码都没有意义(比如汉字有大小写吗?或许把繁体汉字可以认为是大写，而简体汉字认为是小写)

上面 `/lib/modules/4.4.1-040401-generic/kernel/fs/nls` 目录下的各个nls module就是完成这个工作的！

exmaple code in ntfs file system driver

in fs/ntfs/namei.c

```
static struct dentry *ntfs_lookup(struct inode *dir_ino, struct dentry *dent,
        unsigned int flags)
{
    ntfs_volume *vol = NTFS_SB(dir_ino->i_sb);
    struct inode *dent_inode;
    ntfschar *uname;
    ntfs_name *name = NULL;
    MFT_REF mref;
    unsigned long dent_ino;
    int uname_len;

    ntfs_debug("Looking up %s in directory inode 0x%lx.",
            dent->d_name.name, dir_ino->i_ino);
    /* Convert the name of the dentry to Unicode. */
    uname_len = ntfs_nlstoucs(vol, dent->d_name.name, dent->d_name.len,     ①
            &uname);
    if (uname_len < 0) {
        if (uname_len != -ENAMETOOLONG)
            ntfs_error(vol->sb, "Failed to convert name to "
                    "Unicode.");
        return ERR_PTR(uname_len);
    }
    mref = ntfs_lookup_inode_by_name(NTFS_I(dir_ino), uname, uname_len,     ②
            &name);
    kmem_cache_free(ntfs_name_cache, uname);

......
```

ntfs_lookup()寻找parameter dir_ino代表的path

①

dent->d_name.name

dent->d_name.len

ntfs文件系统上的路径名和长度

路径名的编码完全是用户定的，比如使用中文路径名，可能就是GB2312 (cp936)

ntfs_nlstoucs()负责把路径名转换成unicode，存放在uname中，uname_len是转换成unicode后的长度。

②

ntfs driver在处理中使用的都是unicode string.

in fs/ntfs/unistr.c

```c
/**
 * ntfs_nlstoucs - convert NLS string to little endian Unicode string
 * @vol:    ntfs volume which we are working with
 * @ins:    input NLS string buffer
 * @ins_len:    length of input string in bytes
 * @outs:   on return contains the allocated output Unicode string buffer
 *
 * Convert the input string @ins, which is in whatever format the loaded NLS
 * map dictates, into a little endian, 2-byte Unicode string.
 *
 * This function allocates the string and the caller is responsible for
 * calling kmem_cache_free(ntfs_name_cache, *@outs); when finished with it.
 *
 * On success the function returns the number of Unicode characters written to
 * the output string *@outs (>= 0), not counting the terminating Unicode NULL
 * character. *@outs is set to the allocated output string buffer.
 *
 * On error, a negative number corresponding to the error code is returned. In
 * that case the output string is not allocated. Both *@outs and *@outs_len
 * are then undefined.
 *
 * This might look a bit odd due to fast path optimization...
 */
int ntfs_nlstoucs(const ntfs_volume *vol, const char *ins,
        const int ins_len, ntfschar **outs)
{
    struct nls_table *nls = vol->nls_map;    ③
    ntfschar *ucs;
    wchar_t wc;
    int i, o, wc_len;

    /* We do not trust outside sources. */
    if (likely(ins)) {
        ucs = kmem_cache_alloc(ntfs_name_cache, GFP_NOFS);
        if (likely(ucs)) {
            for (i = o = 0; i < ins_len; i += wc_len) {
                wc_len = nls->char2uni(ins + i, ins_len - i,         ④
                        &wc);
                if (likely(wc_len >= 0 &&
                        o < NTFS_MAX_NAME_LEN)) {
                    if (likely(wc)) {
                        ucs[o++] = cpu_to_le16(wc);
                        continue;
                    } /* else if (!wc) */
                    break;
                } /* else if (wc_len < 0 ||
                        o >= NTFS_MAX_NAME_LEN) */
                goto name_err;
            }
            ucs[o] = 0;
```

```
                *outs = ucs;
                return o;
            } /* else if (!ucs) */
            ntfs_error(vol->sb, "Failed to allocate buffer for converted "
                    "name from ntfs_name_cache.");
            return -ENOMEM;
        } /* else if (!ins) */
        ntfs_error(vol->sb, "Received NULL pointer.");
        return -EINVAL;
name_err:
        kmem_cache_free(ntfs_name_cache, ucs);
        if (wc_len < 0) {
            ntfs_error(vol->sb, "Name using character set %s contains "
                    "characters that cannot be converted to "
                    "Unicode.", nls->charset);
            i = -EILSEQ;
        } else /* if (o >= NTFS_MAX_NAME_LEN) */ {
            ntfs_error(vol->sb, "Name is too long (maximum length for a "
                    "name on NTFS is %d Unicode characters.",
                    NTFS_MAX_NAME_LEN);
            i = -ENAMETOOLONG;
        }
        return i;
}
```

③

```
/*
 * The NTFS in memory super block structure.
 */
typedef struct {
......
    struct nls_table *nls_map;
} ntfs_volume;
```

in Documentation/filesystems/ntfs.txt

```
nls=name        Character set to use when returning file names.
            Unlike VFAT, NTFS suppresses names that contain
            unconvertible characters.  Note that most character
            sets contain insufficient characters to represent all
            possible Unicode characters that can exist on NTFS.
            To be sure you are not missing any files, you are
            advised to use nls=utf8 which is capable of
            representing all Unicode characters.
```

即如果你的ntfs file system有GB2312 encoding的path，那么在载入ntfs driver时应该指定如下module parameter

> nls="cp936"

or

> nls="gb2312"

这样ntfs driver会载入nls_cp936.ko来处理字符转换，同时上面struct ntfs_volume中的nls_map指向如下structure (in fs/nls/nls_cp936.c)

```
static struct nls_table table = {
    .charset    = "cp936",
    .alias      = "gb2312",
    .uni2char   = uni2char,
    .char2uni   = char2uni,
    .charset2lower  = charset2lower,
    .charset2upper  = charset2upper,
};
```

④
调用char2uni callback做GB2312到unicode的转换。

---

kernel处理完了unicode string，如果需要存盘，那么需要把unicode转换成文件系统原来所使用的encoding.

ntfs driver中ntfs_ucstonls()就干这事。

```c
int ntfs_ucstonls(const ntfs_volume *vol, const ntfschar *ins,
        const int ins_len, unsigned char **outs, int outs_len)
{
    struct nls_table *nls = vol->nls_map;
    unsigned char *ns;
    int i, o, ns_len, wc;

    /* We don't trust outside sources. */
    if (ins) {
        ns = *outs;
        ns_len = outs_len;
        if (ns && !ns_len) {
            wc = -ENAMETOOLONG;
            goto conversion_err;
        }
        if (!ns) {
            ns_len = ins_len * NLS_MAX_CHARSET_SIZE;
            ns = kmalloc(ns_len + 1, GFP_NOFS);
            if (!ns)
                goto mem_err_out;
        }
        for (i = o = 0; i < ins_len; i++) {
retry:          wc = nls->uni2char(le16_to_cpu(ins[i]), ns + o,     ⑤
                    ns_len - o);
            if (wc > 0) {
                o += wc;
                continue;
            } else if (!wc)
                break;
            else if (wc == -ENAMETOOLONG && ns != *outs) {
                unsigned char *tc;
                /* Grow in multiples of 64 bytes. */
                tc = kmalloc((ns_len + 64) &
                        ~63, GFP_NOFS);
                if (tc) {
                    memcpy(tc, ns, ns_len);
                    ns_len = ((ns_len + 64) & ~63) - 1;
                    kfree(ns);
                    ns = tc;
                    goto retry;
                } /* No memory so goto conversion_error; */
            } /* wc < 0, real error. */
            goto conversion_err;
        }
        ns[o] = 0;
        *outs = ns;
        return o;
    } /* else (!ins) */
    ntfs_error(vol->sb, "Received NULL pointer.");
    return -EINVAL;
conversion_err:
    ntfs_error(vol->sb, "Unicode name contains characters that cannot be "
            "converted to character set %s.  You might want to "
            "try to use the mount option nls=utf8.", nls->charset);
    if (ns != *outs)
        kfree(ns);
    if (wc != -ENAMETOOLONG)
        wc = -EILSEQ;
    return wc;
mem_err_out:
    ntfs_error(vol->sb, "Failed to allocate name!");
    return -ENOMEM;
}
```

⑤

把unicode转化回原来的编码。

in fs/ntfs/dir.c

```
static inline int ntfs_filldir(ntfs_volume *vol,
        ntfs_inode *ndir, struct page *ia_page, INDEX_ENTRY *ie,
        u8 *name, struct dir_context *actor)
{
    unsigned long mref;
    int name_len;
    unsigned dt_type;
    FILE_NAME_TYPE_FLAGS name_type;

    name_type = ie->key.file_name.file_name_type;
    if (name_type == FILE_NAME_DOS) {
        ntfs_debug("Skipping DOS name space entry.");
        return 0;
    }
    if (MREF_LE(ie->data.dir.indexed_file) == FILE_root) {
        ntfs_debug("Skipping root directory self reference entry.");
        return 0;
    }
    if (MREF_LE(ie->data.dir.indexed_file) < FILE_first_user &&
            !NVolShowSystemFiles(vol)) {
        ntfs_debug("Skipping system file.");
        return 0;
    }
    name_len = ntfs_ucstonls(vol, (ntfschar*)&ie->key.file_name.file_name,
⑥
            ie->key.file_name.file_name_length, &name,
            NTFS_MAX_NAME_LEN * NLS_MAX_CHARSET_SIZE + 1);
    if (name_len <= 0) {
        ntfs_warning(vol->sb, "Skipping unrepresentable inode 0x%llx.",
                (long long)MREF_LE(ie->data.dir.indexed_file));
        return 0;
    }

......
```

ntfs_filldir()就是写ntfs文件系统目录的

⑥
把unicode转换成原有编码。