strncpy(3) - Linux man page

Name

strcpy, strncpy - copy a string

Synopsis

```
#include <string.h>

char *strcpy(char *dest, const char *src);

char *strncpy(char *dest, const char *src, size_t n);
```

Description

The **strcpy**() function copies the string pointed to by *src*, including the terminating null byte ('\0'), to the buffer pointed to by *dest*. The strings may not overlap, and the destination string *dest* must be large enough to receive the copy. *Beware of buffer overruns!* (See BUGS.)

The **strncpy**() function is similar, except that at most *n* bytes of *src* are copied. **Warning**: If there is no null byte among the first *n* bytes of *src*, the string placed in *dest* will not be null-terminated.

If the length of *src* is less than *n*, **strncpy**() writes additional null bytes to *dest* to ensure that a total of *n* bytes are written.

A simple implementation of **strncpy**() might be:

```
char *
strncpy(char *dest, const char *src, size_t n)
{
    size_t i;

   for (i = 0; i < n && src[i] != '\0'; i++)
        dest[i] = src[i];
    for ( ; i < n; i++)
        dest[i] = '\0';

   return dest;
}
```

Return Value

The **strcpy**() and **strncpy**() functions return a pointer to the destination string *dest*.

Conforming To

SVr4, 4.3BSD, C89, C99.

Notes

Some programmers consider **strncpy**() to be inefficient and error prone. If the programmer knows (i.e., includes code to test!) that the size of *dest* is greater than the length of *src*, then **strcpy**() can be used.

One valid (and intended) use of **strncpy**() is to copy a C string to a fixed-length buffer while ensuring both that the buffer is not overflowed and that unused bytes in the target buffer are zeroed out (perhaps to prevent information leaks if the buffer is to be written to media or transmitted to another process via an interprocess communication technique).

If there is no terminating null byte in the first *n* bytes of *src*, **strncpy**() produces an unterminated string in *dest*. You can force termination using something like the following:

```
strncpy(buf, str, n);
if (n > 0)
    buf[n - 1]= '\0';
```

(Of course, the above technique ignores the fact that information contained in *src* is lost in the copying to *dest*.)

Some systems (the BSDs, Solaris, and others) provide the following function:

size_t strlcpy(char *dest, const char *src, size_t size);

This function is similar to **strncpy**(), but it copies at most *size-1* bytes to *dest*, always adds a terminating null byte, and does not pad the target with (further) null bytes. This function fixes some of the problems of **strcpy**() and **strncpy**(), but the caller must still handle the possibility of data loss if *size* is too small. The return value of the function is the length of *src*, which allows truncation to be easily detected: if the return value is greater than or equal to *size*, truncation occurred. If loss of data matters, the caller *must* either check the arguments before the call, or test the function return value. **strlcpy**() is not present in glibc and is not standardized by POSIX, but is available on Linux via the *libbsd* library.

Bugs

If the destination string of a **strcpy**() is not large enough, then anything might happen. Overflowing fixed-length string buffers is a favorite cracker technique for taking complete control of the machine. Any time a program reads or copies data into a buffer, the program first needs to check that there's enough space. This may be unnecessary if you can show that overflow is impossible, but be careful: programs can get changed over time, in ways that may make the impossible possible.

See Also

bcopy(3), **memccpy**(3), **memcpy**(3), **memmove**(3), **stpcpy**(3), **stpncpy**(3), **strdup**(3), **string**(3), **wcscpy**(3), **wcsncpy**(3)

Referenced By

db2x_manxml(1), **feature_test_macros**(7), **fmt_strn**(3), **strcat**(3), **strl**(3), **strlcpy**(3), **strxfrm**(3)