

NAME

strncpy, **strncat** - size-bounded string copying and concatenation

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <string.h>
```

size_t

```
strncpy(char * restrict dst, const char * restrict src, size_t dstsize);
```

size_t

```
strncat(char * restrict dst, const char * restrict src, size_t dstsize);
```

DESCRIPTION

The **strncpy()** and **strncat()** functions copy and concatenate strings with the same input parameters and output result as `snprintf(3)`. They are designed to be safer, more consistent, and less error prone replacements for the easily misused functions `strncpy(3)` and `strncat(3)`.

strncpy() and **strncat()** take the full size of the destination buffer and guarantee NUL-termination if there is room. Note that room for the NUL should be included in *dstsize*.

strncpy() copies up to *dstsize* - 1 characters from the string *src* to *dst*, NUL-terminating the result if *dstsize* is not 0.

strncat() appends string *src* to the end of *dst*. It will append at most *dstsize* - `strlen(dst)` - 1 characters. It will then NUL-terminate, unless *dstsize* is 0 or the original *dst* string was longer than *dstsize* (in practice this should not happen as it means that either *dstsize* is incorrect or that *dst* is not a proper string).

If the *src* and *dst* strings overlap, the behavior is undefined.

RETURN VALUES

Besides quibbles over the return type (*size_t* versus *int*) and signal handler safety (`snprintf(3)` is not entirely safe on some systems), the following two are equivalent:

```
n = strncpy(dst, src, len);
n = snprintf(dst, len, "%s", src);
```

Like `snprintf(3)`, the **strncpy()** and **strncat()** functions return the total length of the string they tried to

create. For **strncpy()** that means the length of *src*. For **strncat()** that means the initial length of *dst* plus the length of *src*.

If the return value is $\geq \text{dstsize}$, the output string has been truncated. It is the caller's responsibility to handle this.

EXAMPLES

The following code fragment illustrates the simple case:

```
char *s, *p, buf[BUFSIZ];

...

(void)strncpy(buf, s, sizeof(buf));
(void)strncat(buf, p, sizeof(buf));
```

To detect truncation, perhaps while building a pathname, something like the following might be used:

```
char *dir, *file, pname[MAXPATHLEN];

...

if (strncpy(pname, dir, sizeof(pname)) >= sizeof(pname))
    goto toolong;
if (strncat(pname, file, sizeof(pname)) >= sizeof(pname))
    goto toolong;
```

Since it is known how many characters were copied the first time, things can be sped up a bit by using a copy instead of an append:

```
char *dir, *file, pname[MAXPATHLEN];
size_t n;

...

n = strncpy(pname, dir, sizeof(pname));
if (n >= sizeof(pname))
    goto toolong;
if (strncpy(pname + n, file, sizeof(pname) - n) >= sizeof(pname) - n)
    goto toolong;
```

However, one may question the validity of such optimizations, as they defeat the whole purpose of **strncpy()** and **strcat()**. As a matter of fact, the first version of this manual page got it wrong.

SEE ALSO

snprintf(3), strncat(3), strncpy(3), wcsncpy(3)

Todd C. Miller and Theo de Raadt, "strncpy and strcat -- Consistent, Safe, String Copy and Concatenation", *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, USENIX Association,

http://www.usenix.org/publications/library/proceedings/usenix99/full_papers/millert/millert.pdf, June 6-11, 1999.

HISTORY

The **strncpy()** and **strcat()** functions first appeared in OpenBSD 2.4, and FreeBSD 3.3.