NAME

stpcpy, strcpy, strcat, stpecpy, strlcpy, strlcat, stpncpy, strncpy, zustr2ustp, zustr2stp, strncat, ustpcpy, ustr2stp – copying strings and character sequences

SYNOPSIS

Strings

```
// Chain-copy a string.
char *stpcpy(char *restrict dst, const char *restrict src);

// Copy/catenate a string.
char *strcpy(char *restrict dst, const char *restrict src);
char *strcat(char *restrict dst, const char *restrict src);

// Chain-copy a string with truncation.
char *stpecpy(char *dst, char end[0], const char *restrict src);

// Copy/catenate a string with truncation.
size_t strlcpy(char dst[restrict .sz], const char *restrict src,
               size_t sz);
size_t strlcat(char dst[restrict .sz], const char *restrict src,
               size_t sz);
```

Null-padded character sequences

```
// Zero a fixed-width buffer, and
// copy a string into a character sequence with truncation.
char *stpncpy(char dst[restrict .sz], const char *restrict src,
              size_t sz);

// Zero a fixed-width buffer, and
// copy a string into a character sequence with truncation.
char *strncpy(char dest[restrict .sz], const char *restrict src,
              size_t sz);

// Chain-copy a null-padded character sequence into a character sequence.
char *zustr2ustp(char *restrict dst, const char src[restrict .sz],
                 size_t sz);

// Chain-copy a null-padded character sequence into a string.
char *zustr2stp(char *restrict dst, const char src[restrict .sz],
                size_t sz);

// Catenate a null-padded character sequence into a string.
char *strncat(char *restrict dst, const char src[restrict .sz],
              size_t sz);
```

Measured character sequences

```
// Chain-copy a measured character sequence.
char *ustpcpy(char *restrict dst, const char src[restrict .len],
              size_t len);

// Chain-copy a measured character sequence into a string.
char *ustr2stp(char *restrict dst, const char src[restrict .len],
               size_t len);
```

DESCRIPTION

Terms (and abbreviations)

string (*str*)
: is a sequence of zero or more non-null characters followed by a null byte.

character sequence
: is a sequence of zero or more non-null characters. A program should never use a character sequence where a string is required. However, with appropriate care, a string can be used in the

place of a character sequence.

null-padded character sequence (*zustr*)
: Character sequences can be contained in fixed-width buffers, which contain padding null bytes after the character sequence, to fill the rest of the buffer without affecting the character sequence; however, those padding null bytes are not part of the character sequence.

measured character sequence (*ustr*)
: Character sequence delimited by its length. It may be a slice of a larger character sequence, or even of a string.

length (*len*)
: is the number of non-null characters in a string or character sequence. It is the return value of *strlen(str)* and of *strnlen(ustr, sz)*.

size (*sz*)
: refers to the entire buffer where the string or character sequence is contained.

end : is the name of a pointer to one past the last element of a buffer. It is equivalent to *&str[sz]*. It is used as a sentinel value, to be able to truncate strings or character sequences instead of overrunning the containing buffer.

copy : This term is used when the writing starts at the first element pointed to by *dst*.

catenate
: This term is used when a function first finds the terminating null byte in *dst*, and then starts writing at that position.

chain : This term is used when it's the programmer who provides a pointer to the terminating null byte in the string *dst* (or one after the last character in a character sequence), and the function starts writing at that location. The function returns a pointer to the new location of the terminating null byte (or one after the last character in a character sequence) after the call, so that the programmer can use it to chain such calls.

Copy, catenate, and chain-copy

Originally, there was a distinction between functions that copy and those that catenate. However, newer functions that copy while allowing chaining cover both use cases with a single API. They are also algorithmically faster, since they don't need to search for the terminating null byte of the existing string. However, functions that catenate have a much simpler use, so if performance is not important, it can make sense to use them for improving readability.

The pointer returned by functions that allow chaining is a byproduct of the copy operation, so it has no performance costs. Functions that return such a pointer, and thus can be chained, have names of the form *__stp__*(), since it's common to name the pointer just *p*.

Chain-copying functions that truncate should accept a pointer to the end of the destination buffer, and have names of the form *__stpe__*(). This allows not having to recalculate the remaining size after each call.

Truncate or not?

The first thing to note is that programmers should be careful with buffers, so they always have the correct size, and truncation is not necessary.

In most cases, truncation is not desired, and it is simpler to just do the copy. Simpler code is safer code. Programming against programming mistakes by adding more code just adds more points where mistakes can be made.

Nowadays, compilers can detect most programmer errors with features like compiler warnings, static analyzers, and **_FORTIFY_SOURCE** (see **ftm**(7)). Keeping the code simple helps these overflow-detection features be more precise.

When validating user input, however, it makes sense to truncate. Remember to check the return value of such function calls.

Functions that truncate:

- **stpecpy**(3) is the most efficient string copy function that performs truncation. It only requires to check for truncation once after all chained calls.
- **strlcpy**(3bsd) and **strlcat**(3bsd) are designed to crash if the input string is invalid (doesn't contain a terminating null byte).
- **stpncpy**(3) and **strncpy**(3) also truncate, but they don't write strings, but rather null-padded character sequences.

Null-padded character sequences

For historic reasons, some standard APIs, such as **utmpx**(5), use null-padded character sequences in fixed-width buffers. To interface with them, specialized functions need to be used.

To copy strings into them, use **stpncpy**(3).

To copy from an unterminated string within a fixed-width buffer into a string, ignoring any trailing null bytes in the source fixed-width buffer, you should use **zustr2stp**(3) or **strncat**(3).

To copy from an unterminated string within a fixed-width buffer into a character sequence, ignoring any trailing null bytes in the source fixed-width buffer, you should use **zustr2ustp**(3).

Measured character sequences

The simplest character sequence copying function is **mempcpy**(3). It requires always knowing the length of your character sequences, for which structures can be used. It makes the code much faster, since you always know the length of your character sequences, and can do the minimal copies and length measurements. **mempcpy**(3) copies character sequences, so you need to explicitly set the terminating null byte if you need a string.

However, for keeping type safety, it's good to add a wrapper that uses *char* * instead of *void* *: **ustpcpy**(3).

In programs that make considerable use of strings or character sequences, and need the best performance, using overlapping character sequences can make a big difference. It allows holding subsequences of a larger character sequence. while not duplicating memory nor using time to do a copy.

However, this is delicate, since it requires using character sequences. C library APIs use strings, so programs that use character sequences will have to take care of differentiating strings from character sequences.

To copy a measured character sequence, use **ustpcpy**(3).

To copy a measured character sequence into a string, use **ustr2stp**(3).

Because these functions ask for the length, and a string is by nature composed of a character sequence of the same length plus a terminating null byte, a string is also accepted as input.

String vs character sequence

Some functions only operate on strings. Those require that the input *src* is a string, and guarantee an output string (even when truncation occurs). Functions that catenate also require that *dst* holds a string before the call. List of functions:

- **stpcpy**(3)
- **strcpy**(3), **strcat**(3)
- **stpecpy**(3)
- **strlcpy**(3bsd), **strlcat**(3bsd)

Other functions require an input string, but create a character sequence as output. These functions have confusing names, and have a long history of misuse. List of functions:

- **stpncpy**(3)
- **strncpy**(3)

Other functions operate on an input character sequence, and create an output string. Functions that catenate also require that *dst* holds a string before the call. **strncat**(3) has an even more misleading name than the functions above. List of functions:

- **zustr2stp**(3)
- **strncat**(3)
- **ustr2stp**(3)

Other functions operate on an input character sequence to create an output character sequence. List of functions:

- **ustpcpy**(3)
- **zustr2stp**(3)

Functions

stpcpy(3)
: This function copies the input string into a destination string. The programmer is responsible for allocating a buffer large enough. It returns a pointer suitable for chaining.

strcpy(3)
strcat(3)
: These functions copy and catenate the input string into a destination string. The programmer is responsible for allocating a buffer large enough. The return value is useless.

  stpcpy(3) is a faster alternative to these functions.

stpecpy(3)
: This function copies the input string into a destination string. If the destination buffer, limited by a pointer to its end, isn't large enough to hold the copy, the resulting string is truncated (but it is guaranteed to be null-terminated). It returns a pointer suitable for chaining. Truncation needs to be detected only once after the last chained call.

  This function is not provided by any library; See EXAMPLES for a reference implementation.

strlcpy(3bsd)
strlcat(3bsd)
: These functions copy and catenate the input string into a destination string. If the destination buffer, limited by its size, isn't large enough to hold the copy, the resulting string is truncated (but it is guaranteed to be null-terminated). They return the length of the total string they tried to create. These functions force a SIGSEGV if the *src* pointer is not a string.

  stpecpy(3) is a simpler alternative to these functions.

stpncpy(3)
: This function copies the input string into a destination null-padded character sequence in a fixed-width buffer. If the destination buffer, limited by its size, isn't large enough to hold the copy, the resulting character sequence is truncated. Since it creates a character sequence, it doesn't need to write a terminating null byte. It's impossible to distinguish truncation by the result of the call, from a character sequence that just fits the destination buffer; truncation should be detected by comparing the length of the input string with the size of the destination buffer.

strncpy(3)
: This function is identical to **stpncpy**(3) except for the useless return value.

  stpncpy(3) is a more useful alternative to this function.

zustr2ustp(3)
: This function copies the input character sequence contained in a null-padded wixed-width buffer, into a destination character sequence. The programmer is responsible for allocating a buffer large enough. It returns a pointer suitable for chaining.

  A truncating version of this function doesn't exist, since the size of the original character sequence is always known, so it wouldn't be very useful.

  This function is not provided by any library; See EXAMPLES for a reference implementation.

zustr2stp(3)
: This function copies the input character sequence contained in a null-padded wixed-width buffer, into a destination string. The programmer is responsible for allocating a buffer large enough. It returns a pointer suitable for chaining.

  A truncating version of this function doesn't exist, since the size of the original character sequence is always known, so it wouldn't be very useful.

  This function is not provided by any library; See EXAMPLES for a reference implementation.

strncat(3)
: Do not confuse this function with **strncpy**(3); they are not related at all.

  This function catenates the input character sequence contained in a null-padded wixed-width buffer, into a destination string. The programmer is responsible for allocating a buffer large enough. The return value is useless.

  zustr2stp(3) is a faster alternative to this function.

ustpcpy(3)
: This function copies the input character sequence, limited by its length, into a destination character sequence. The programmer is responsible for allocating a buffer large enough. It returns a pointer suitable for chaining.

ustr2stp(3)
: This function copies the input character sequence, limited by its length, into a destination string. The programmer is responsible for allocating a buffer large enough. It returns a pointer suitable for chaining.

RETURN VALUE

The following functions return a pointer to the terminating null byte in the destination string.

- **stpcpy**(3)
- **ustr2stp**(3)
- **zustr2stp**(3)

The following function returns a pointer to the terminating null byte in the destination string, except when truncation occurs; if truncation occurs, it returns a pointer to the end of the destination buffer.

- **stpecpy**(3)

The following function returns a pointer to one after the last character in the destination character sequence; if truncation occurs, that pointer is equivalent to a pointer to the end of the destination buffer.

- **stpncpy**(3)

The following functions return a pointer to one after the last character in the destination character sequence.

- **zustr2ustp**(3)
- **ustpcpy**(3)

The following functions return the length of the total string that they tried to create (as if truncation didn't occur).

- **strlcpy**(3bsd), **strlcat**(3bsd)

The following functions return the *dst* pointer, which is useless.

- **strcpy**(3), **strcat**(3)
- **strncpy**(3)
- **strncat**(3)

NOTES

The Linux kernel has an internal function for copying strings, which is similar to **stpecpy**(3), except that it can't be chained:

strscpy(9)

This function copies the input string into a destination string. If the destination buffer, limited by its size, isn't large enough to hold the copy, the resulting string is truncated (but it is guaranteed to be null-terminated). It returns the length of the destination string, or **−E2BIG** on truncation.

stpecpy(3) is a simpler and faster alternative to this function.

CAVEATS

Don't mix chain calls to truncating and non-truncating functions. It is conceptually wrong unless you know that the first part of a copy will always fit. Anyway, the performance difference will probably be negligible, so it will probably be more clear if you use consistent semantics: either truncating or non-truncating. Calling a non-truncating function after a truncating one is necessarily wrong.

BUGS

All catenation functions share the same performance problem: Shlemiel the painter ⟨https://www.joelonsoftware.com/2001/12/11/back−to−basics/⟩.

EXAMPLES

The following are examples of correct use of each of these functions.

stpcpy(3)

```
p = buf;
p = stpcpy(p, "Hello ");
p = stpcpy(p, "world");
p = stpcpy(p, "!");
len = p - buf;
puts(buf);
```

strcpy(3)
strcat(3)

```
strcpy(buf, "Hello ");
strcat(buf, "world");
strcat(buf, "!");
len = strlen(buf);
puts(buf);
```

stpecpy(3)

```
end = buf + sizeof(buf);
p = buf;
p = stpecpy(p, end, "Hello ");
p = stpecpy(p, end, "world");
p = stpecpy(p, end, "!");
if (p == end) {
    p--;
    goto toolong;
}
len = p - buf;
puts(buf);
```

strlcpy(3bsd)
strlcat(3bsd)

```
if (strlcpy(buf, "Hello ", sizeof(buf)) >= sizeof(buf))
    goto toolong;
if (strlcat(buf, "world", sizeof(buf)) >= sizeof(buf))
    goto toolong;
len = strlcat(buf, "!", sizeof(buf));
if (len >= sizeof(buf))
    goto toolong;
puts(buf);
```

strscpy(9)

```
len = strscpy(buf, "Hello world!", sizeof(buf));
if (len == -E2BIG)
    goto toolong;
puts(buf);
```

stpncpy(3)

```
p = stpncpy(buf, "Hello world!", sizeof(buf));
if (sizeof(buf) < strlen("Hello world!"))
    goto toolong;
len = p - buf;
for (size_t i = 0; i < sizeof(buf); i++)
    putchar(buf[i]);
```

strncpy(3)

```
strncpy(buf, "Hello world!", sizeof(buf));
if (sizeof(buf) < strlen("Hello world!"))
    goto toolong;
len = strnlen(buf, sizeof(buf));
for (size_t i = 0; i < sizeof(buf); i++)
    putchar(buf[i]);
```

zustr2ustp(3)

```
p = buf;
p = zustr2ustp(p, "Hello ", 6);
p = zustr2ustp(p, "world", 42);  // Padding null bytes ignored.
p = zustr2ustp(p, "!", 1);
len = p - buf;
printf("%.*s\n", (int) len, buf);
```

zustr2stp(3)

```
p = buf;
p = zustr2stp(p, "Hello ", 6);
p = zustr2stp(p, "world", 42);  // Padding null bytes ignored.
p = zustr2stp(p, "!", 1);
len = p - buf;
puts(buf);
```

strncat(3)

```
buf[0] = '\0';  // There's no 'cpy' function to this 'cat'.
strncat(buf, "Hello ", 6);
strncat(buf, "world", 42);  // Padding null bytes ignored.
strncat(buf, "!", 1);
len = strlen(buf);
puts(buf);
```

ustpcpy(3)

```
p = buf;
p = ustpcpy(p, "Hello ", 6);
p = ustpcpy(p, "world", 5);
p = ustpcpy(p, "!", 1);
len = p - buf;
printf("%.*s\n", (int) len, buf);
```

ustr2stp(3)

```
p = buf;
p = ustr2stp(p, "Hello ", 6);
p = ustr2stp(p, "world", 5);
p = ustr2stp(p, "!", 1);
```

```
    len = p - buf;
    puts(buf);
```

Implementations

Here are reference implementations for functions not provided by libc.

```
/* This code is in the public domain. */

char *
stpecpy(char *dst, char end[0], const char *restrict src)
{
    char *p;

    if (src[strlen(src)] != '\0')
        raise(SIGSEGV);

    if (dst == end)
        return end;

    p = memccpy(dst, src, '\0', end - dst);
    if (p != NULL)
        return p - 1;

    /* truncation detected */
    end[-1] = '\0';
    return end;
}

char *
zustr2ustp(char *restrict dst, const char *restrict src, size_t sz)
{
    return ustpcpy(dst, src, strnlen(src, sz));
}

char *
zustr2stp(char *restrict dst, const char *restrict src, size_t sz)
{
    char  *p;

    p = zustr2ustp(dst, src, sz);
    *p = '\0';

    return p;
}

char *
ustpcpy(char *restrict dst, const char *restrict src, size_t len)
{
    return mempcpy(dst, src, len);
}

char *
ustr2stp(char *restrict dst, const char *restrict src, size_t len)
{
    char  *p;
```

```
    p = ustpcpy(dst, src, len);
    *p = '\0';

    return p;
}
```

SEE ALSO

bzero(3), **memcpy**(3), **memccpy**(3), **mempcpy**(3), **stpcpy**(3), **strlcpy**(3bsd), **strncat**(3), **stpncpy**(3), **string**(3)