stdarg(3) - Linux man page

Name

stdarg, va_start, va_arg, va_end, va_copy - variable argument lists

Synopsis

#include <stdarg.h>

void va_start(va_list *ap*, *last*);
type **va_arg(va_list** *ap*, *type*);
void va_end(va_list *ap*);
void va_copy(va_list *dest*, **va_list** *src*);

Description

A function may be called with a varying number of arguments of varying types. The include file <*stdarg.h*> declares a type *va_list* and defines three macros for stepping through a list of arguments whose number and types are not known to the called function.

The called function must declare an object of type *va_list* which is used by the macros **va_start**(), **va_arg**(), and **va_end**().

va_start()

The **va_start**() macro initializes *ap* for subsequent use by **va_arg**() and **va_end**(), and must be called first.

The argument *last* is the name of the last argument before the variable argument list, that is, the last argument of which the calling function knows the type.

Because the address of this argument may be used in the **va_start**() macro, it should not be declared as a register variable, or as a function or an array type.

va_arg()

The **va_arg**() macro expands to an expression that has the type and value of the next argument in the call. The argument *ap* is the *va_list ap* initialized by **va_start**(). Each call to **va_arg**() modifies *ap* so that the next call returns the next argument. The argument *type* is a type name specified so that the type of a pointer to an object that has the specified type can be obtained simply by adding a * to *type*.

The first use of the **va_arg**() macro after that of the **va_start**() macro returns the argument after *last*. Successive invocations return the values of the remaining arguments.

If there is no next argument, or if *type* is not compatible with the type of the actual

next argument (as promoted according to the default argument promotions), random errors will occur.

If *ap* is passed to a function that uses **va_arg(***ap***,***type***)** then the value of *ap* is undefined after the return of that function.

va_end()

Each invocation of **va_start**() must be matched by a corresponding invocation of **va_end**() in the same function. After the call **va_end(***ap***)** the variable *ap* is undefined. Multiple traversals of the list, each bracketed by **va_start**() and **va_end**() are possible. **va_end**() may be a macro or a function.

va_copy()

The **va_copy**() macro copies the (previously initialized) variable argument list *src* to *dest*. The behavior is as if **va_start**() were applied to *dest* with the same *last* argument, followed by the same number of **va_arg**() invocations that was used to reach the current state of *src*.

An obvious implementation would have a *va_list* be a pointer to the stack frame of the variadic function. In such a setup (by far the most common) there seems nothing against an assignment

```
va_list aq = ap;
```

Unfortunately, there are also systems that make it an array of pointers (of length 1), and there one needs

```
va_list aq;
*aq = *ap;
```

Finally, on systems where arguments are passed in registers, it may be necessary for **va_start**() to allocate memory, store the arguments there, and also an indication of which argument is next, so that **va_arg**() can step through the list. Now **va_end**() can free the allocated memory again. To accommodate this situation, C99 adds a macro **va_copy**(), so that the above assignment can be replaced by

```
va_list aq;
va_copy(aq, ap);
...
va_end(aq);
```

Each invocation of **va_copy**() must be matched by a corresponding invocation of **va_end**() in the same function. Some systems that do not supply **va_copy**() have **__va_copy** instead, since that was the name used in the draft proposal.

Conforming To

The **va_start**(), **va_arg**(), and **va_end**() macros conform to C89. C99 defines the **va_copy**() macro.

Notes

These macros are *not* compatible with the historic macros they replace. A backward-compatible version can be found in the include file *<varargs.h>*.

The historic setup is:

```
    #include <varargs.h>

    void
    foo(va_alist)
        va_dcl
    {
        va_list ap;

       va_start(ap);
        while (...) {
            ...
            x = va_arg(ap, type);
            ...
        }
        va_end(ap);
    }
```

On some systems, *va_end* contains a closing '}' matching a '{' in *va_start*, so that both macros must occur in the same function, and in a way that allows this.

Bugs

Unlike the **varargs** macros, the **stdarg** macros do not permit programmers to code a function with no fixed arguments. This problem generates work mainly when converting **varargs** code to **stdarg** code, but it also creates difficulties for variadic functions that wish to pass all of their arguments on to a function that takes a *va_list* argument, such as **vfprintf**(3).

Example

The function *foo* takes a string of format characters and prints out the argument associated with each format character based on the type.

```
#include <stdio.h>
#include <stdarg.h>

void
foo(char *fmt, ...)
```

```
{
    va_list ap;
    int d;
    char c, *s;

   va_start(ap, fmt);
    while (*fmt)
        switch (*fmt++) {
        case 's':              /* string */
            s = va_arg(ap, char *);
            printf("string %s\n", s);
            break;
        case 'd':              /* int */
            d = va_arg(ap, int);
            printf("int %d\n", d);
            break;
        case 'c':              /* char */
            /* need a cast here since va_arg only
               takes fully promoted types */
            c = (char) va_arg(ap, int);
            printf("char %c\n", c);
            break;
        }
    va_end(ap);
}
```

Referenced By

avcall(3), **pam_verror**(3), **pam_vinfo**(3), **pam_vsyslog**(3), **printf**(3), **scanf**(3), **syslog**(3), **tiffvgetfielddefaulted**(3), **tiffvsetfield**(3)