Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'stdarg.h.0p' command

$ man stdarg.h.0p

stdarg.h(0P) POSIX Programmer's Manual stdarg.h(0P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stdarg.h ? handle variable argument list

SYNOPSIS

```
#include <stdarg.h>
void va_start(va_list ap, argN);
void va_copy(va_list dest, va_list src);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The <stdarg.h> header shall contain a set of macros which allows porta? ble functions that accept variable argument lists to be written. Func? tions that have variable argument lists (such as printf()) but do not use these macros are inherently non-portable, as different systems use

different argument-passing conventions.

The <stdarg.h> header shall define the va_list type for variables used to traverse the list.

The va_start() macro is invoked to initialize ap to the beginning of the list before any calls to va_arg().

The va_copy() macro initializes dest as a copy of src, as if the va_start() macro had been applied to dest followed by the same sequence of uses of the va_arg() macro as had previously been used to reach the present state of src. Neither the va_copy() nor va_start() macro shall be invoked to reinitialize dest without an intervening invocation of the va_end() macro for the same dest.

The object ap may be passed as an argument to another function; if that function invokes the va_arg() macro with parameter ap, the value of ap in the calling function is unspecified and shall be passed to the va_end() macro prior to any further reference to ap. The parameter argN is the identifier of the rightmost parameter in the variable pa? rameter list in the function definition (the one just before the ...).

If the parameter argN is declared with the register storage class, with a function type or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.

The va_arg() macro shall return the next argument in the list pointed to by ap. Each invocation of va_arg() modifies ap so that the values of successive arguments are returned in turn. The type parameter shall be a type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a '*' to type. If there is no actual next argument, or if type is not compat? ible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined, except for the following cases:

* One type is a signed integer type, the other type is the corre? sponding unsigned integer type, and the value is representable in both types.

* One type is a pointer to void and the other is a pointer to a char?

acter type.

* Both types are pointers.

Different types can be mixed, but it is up to the routine to know what

type of argument is expected.

The va_end() macro is used to clean up; it invalidates ap for use (un?

less va_start() or va_copy() is invoked again).

Each invocation of the va_start() and va_copy() macros shall be matched

by a corresponding invocation of the va_end() macro in the same func?

tion.

Multiple traversals, each bracketed by va_start() ... va_end(), are

possible.

The following sections are informative.

EXAMPLES

This example is a possible implementation of execl():

```
#include <stdarg.h>
#define  MAXARGS     31
/*
 * execl is called by
 * execl(file, arg1, arg2, ..., (char *)(0));
 */
int execl(const char *file, const char *args, ...)
{
    va_list ap;
    char *array[MAXARGS +1];
    int argno = 0;
    va_start(ap, args);
    while (args != 0 && argno < MAXARGS)
    {
        array[argno++] = args;
        args = va_arg(ap, const char *);
    }
    array[argno] = (char *) 0;
```

```
    va_end(ap);
    return execv(file, array);
}
```

APPLICATION USAGE

It is up to the calling routine to communicate to the called routine how many arguments there are, since it is not always possible for the called routine to determine this in any other way. For example, execl() is passed a null pointer to signal the end of the list. The printf() function can tell how many arguments are there by the format argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of POSIX.1?2017, exec, fprintf()

COPYRIGHT