

NAME

posix_memalign, aligned_alloc, memalign, valloc, pvalloc - allocate aligned memory

SYNOPSIS

```
#include <stdlib.h>
```

```
int posix_memalign(void **memptr, size_t alignment, size_t size);
```

```
void *aligned_alloc(size_t alignment, size_t size);
```

```
void *valloc(size_t size);
```

```
#include <malloc.h>
```

```
void *memalign(size_t alignment, size_t size);
```

```
void *pvalloc(size_t size);
```

Feature Test Macro Requirements for glibc (see [feature_test_macros\(7\)](#)):

```
posix_memalign(): _POSIX_C_SOURCE >= 200112L
```

```
aligned_alloc(): _ISOC11_SOURCE
```

```
valloc():
```

Since glibc 2.12:

```
(_XOPEN_SOURCE >= 500) && !(_POSIX_C_SOURCE >= 200112L)
```

```
|| /* Glibc since 2.19: */ _DEFAULT_SOURCE
```

```
|| /* Glibc versions <= 2.19: */ _SVID_SOURCE || _BSD_SOURCE
```

Before glibc 2.12:

```
_BSD_SOURCE || _XOPEN_SOURCE >= 500
```

(The (nonstandard) header file `<malloc.h>` also exposes the declaration of `valloc()`; no feature test macros are required.)

DESCRIPTION

The function `posix_memalign()` allocates *size* bytes and places the address of the allocated memory in **memptr*. The address of the allocated memory will be a multiple of *alignment*, which must be a power of two and a multiple of `sizeof(void *)`. If *size* is 0, then the value placed in **memptr* is either NULL, or a unique pointer value that can later be successfully passed to [free\(3\)](#).

The obsolete function `memalign()` allocates *size* bytes and returns a pointer to the allocated memory. The memory address will be a multiple of *alignment*, which must be a power of two.

The function `aligned_alloc()` is the same as `memalign()`, except for the added restriction that *size* should be a multiple of *alignment*.

The obsolete function `valloc()` allocates *size* bytes and returns a pointer to the allocated memory. The memory address will be a multiple of the page size. It is equivalent to `memalign(sysconf(_SC_PAGESIZE), size)`.

The obsolete function `pvalloc()` is similar to `valloc()`, but rounds the size of the allocation up to the next multiple of the system page size.

For all of these functions, the memory is not zeroed.

RETURN VALUE

`aligned_alloc()`, `memalign()`, `valloc()`, and `pvalloc()` return a pointer to the allocated memory, or NULL if the request fails.

`posix_memalign()` returns zero on success, or one of the error values listed in the next section on failure. The value of `errno` is not set. On Linux (and other systems), `posix_memalign()` does not modify *memptr* on failure. A requirement standardizing this behavior was added in POSIX.1-2016.

ERRORS

EINVAL

The *alignment* argument was not a power of two, or was not a multiple of *sizeof(void *)*.

ENOMEM

There was insufficient memory to fulfill the allocation request.

VERSIONS

The functions **memalign()**, **valloc()**, and **pvalloc()** have been available in all Linux libc libraries.

The function **aligned_alloc()** was added to glibc in version 2.16.

The function **posix_memalign()** is available since glibc 2.1.91.

ATTRIBUTES

For an explanation of the terms used in this section, see [attributes\(7\)](#).

Interface	Attribute	Value
aligned_alloc() , memalign() , posix_memalign()	Thread safety	MT-Safe
valloc() , pvalloc()	Thread safety	MT-Unsafe init

CONFORMING TO

The function **valloc()** appeared in 3.0BSD. It is documented as being obsolete in 4.3BSD, and as legacy in SUSv2. It does not appear in POSIX.1.

The function **pvalloc()** is a GNU extension.

The function **memalign()** appears in SunOS 4.1.3 but not in 4.4BSD.

The function **posix_memalign()** comes from POSIX.1d and is specified in POSIX.1-2001 and POSIX.1-2008.

The function **aligned_alloc()** is specified in the C11 standard.

Headers

Everybody agrees that **posix_memalign()** is declared in `<stdlib.h>`.

On some systems **memalign()** is declared in `<stdlib.h>` instead of `<malloc.h>`.

According to SUSv2, **valloc()** is declared in `<stdlib.h>`. Libc4,5 and glibc declare it in `<malloc.h>`, and also in `<stdlib.h>` if suitable feature test macros are defined (see above).

NOTES

On many systems there are alignment restrictions, for example, on buffers used for direct block device I/O. POSIX specifies the `pathconf(path, PC_REC_XFER_ALIGN)` call that tells what alignment is needed. Now one can use **posix_memalign()** to satisfy this requirement.

posix_memalign() verifies that *alignment* matches the requirements detailed above. **memalign()** may not check that the *alignment* argument is correct.

POSIX requires that memory obtained from **posix_memalign()** can be freed using [free\(3\)](#). Some systems provide no way to reclaim memory allocated with **memalign()** or **valloc()** (because one can pass to [free\(3\)](#) only a pointer obtained from [malloc\(3\)](#), while, for example, **memalign()** would call [malloc\(3\)](#) and then align the obtained value). The glibc implementation allows memory obtained from any of these functions to be reclaimed with [free\(3\)](#).

The glibc [malloc\(3\)](#) always returns 8-byte aligned memory addresses, so these functions are needed only if you require larger alignment values.

SEE ALSO

[brk\(2\)](#), [getpagesize\(2\)](#), [free\(3\)](#), [malloc\(3\)](#)

COLOPHON

This page is part of release 4.10 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.