



Linux Ubuntu 26.04 Manual Pages on command 'madvise.2'

\$ man madvise.2

madvise(2) System Calls Manual madvise(2)

NAME

madvise - give advice about use of memory

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <sys/mman.h>
```

```
int madvise(size_t size;
```

```
void addr[size], size_t size, int advice);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

madvise():

Since glibc 2.19:

 _DEFAULT_SOURCE

Up to and including glibc 2.19:

 _BSD_SOURCE

DESCRIPTION

The `madvise()` system call is used to give advice or directions to the kernel about the address range beginning at address `addr` and with size `size`. `madvise()` only operates on whole pages, therefore `addr` must be page-aligned. The value of `size` is rounded up to a multiple of page size. In most cases, the goal of such advice is to improve system or application performance.

Initially, the system call supported a set of "conventional" advice values, which are also available on several other implementations. (Note, though, that `madvise()` is not specified

in POSIX.) Subsequently, a number of Linux-specific advice values have been added.

Conventional advice values

The advice values listed below allow an application to tell the kernel how it expects to use some mapped or shared memory areas, so that the kernel can choose appropriate read-ahead and caching techniques. These advice values do not influence the semantics of the application (except in the case of `MADV_DONTNEED`), but may influence its performance. All of the advice values listed here have analogs in the POSIX-specified `posix_madvise(3)` function, and the values have the same meanings, with the exception of `MADV_DONTNEED`.

The advice is indicated in the advice argument, which is one of the following:

`MADV_NORMAL`

No special treatment. This is the default.

`MADV_RANDOM`

Expect page references in random order. (Hence, read ahead may be less useful than normally.)

`MADV_SEQUENTIAL`

Expect page references in sequential order. (Hence, pages in the given range can be aggressively read ahead, and may be freed soon after they are accessed.)

`MADV_WILLNEED`

Expect access in the near future. (Hence, it might be a good idea to read some pages ahead.)

`MADV_DONTNEED`

Do not expect access in the near future. (For the time being, the application is finished with the given range, so the kernel can free resources associated with it.)

After a successful `MADV_DONTNEED` operation, the semantics of memory access in the specified region are changed: subsequent accesses of pages in the range will succeed, but will result in either repopulating the memory contents from the up-to-date contents of the underlying mapped file (for shared file mappings, shared anonymous mappings, and `shm`-based techniques such as System V shared memory segments) or zero-fill-on-demand pages for anonymous private mappings.

Note that, when applied to shared mappings, `MADV_DONTNEED` might not lead to immediate freeing of the pages in the range. The kernel is free to delay freeing the pages until an appropriate moment. The resident set size (RSS) of the calling process will be immediately reduced however.

`MADV_DONTNEED` cannot be applied to locked pages, or `VM_PFNMAP` pages. (Pages marked with the kernel-internal `VM_PFNMAP` flag are special memory areas that are not managed by the virtual memory subsystem. Such pages are typically created by device drivers that map the pages into user space.)

Support for Huge TLB pages was added in Linux 5.18. Addresses within a mapping backed by Huge TLB pages must be aligned to the underlying Huge TLB page size, and the range size is rounded up to a multiple of the underlying Huge TLB page size.

Linux-specific advice values

The following Linux-specific advice values have no counterparts in the POSIX-specified `posix_madvise(3)`, and may or may not have counterparts in the `madvise()` interface available on other implementations. Note that some of these operations change the semantics of memory accesses.

`MADV_REMOVE` (since Linux 2.6.16)

Free up a given range of pages and its associated backing store. This is equivalent to punching a hole in the corresponding range of the backing store (see `fallocate(2)`). Subsequent accesses in the specified address range will see data with a value of zero.

The specified address range must be mapped shared and writable. This flag cannot be applied to locked pages, or `VM_PFNMAP` pages.

In the initial implementation, only `tmpfs(5)` supported `MADV_REMOVE`; but since Linux 3.5, any filesystem which supports the `fallocate(2)` `FALLOC_FL_PUNCH_HOLE` mode also supports `MADV_REMOVE`. Filesystems which do not support `MADV_REMOVE` fail with the error `EOPNOTSUPP`.

Support for the Huge TLB filesystem was added in Linux 4.3.

`MADV_DONTFORK` (since Linux 2.6.16)

Do not make the pages in this range available to the child after a `fork(2)`. This is useful to prevent copy-on-write semantics from changing the physical location of a page if the parent writes to it after a `fork(2)`. (Such page relocations cause problems for hardware that DMA's into the page.)

`MADV_DOFORK` (since Linux 2.6.16)

Undo the effect of `MADV_DONTFORK`, restoring the default behavior, whereby a mapping is inherited across `fork(2)`.

`MADV_HWPOISON` (since Linux 2.6.32)

Poison the pages in the range specified by `addr` and `size` and handle subsequent references to those pages like a hardware memory corruption. This operation is available only for privileged (`CAP_SYS_ADMIN`) processes. This operation may result in the calling process receiving a `SIGBUS` and the page being unmapped.

This feature is intended for testing of memory error-handling code; it is available only if the kernel was configured with `CONFIG_MEMORY_FAILURE`.

`MADV_MERGEABLE` (since Linux 2.6.32)

Enable Kernel Samepage Merging (KSM) for the pages in the range specified by `addr` and `size`. The kernel regularly scans those areas of user memory that have been marked as mergeable, looking for pages with identical content. These are replaced by a single write-protected page (which is automatically copied if a process later wants to update the content of the page). KSM merges only private anonymous pages (see `mmap(2)`).

The KSM feature is intended for applications that generate many instances of the same data (e.g., virtualization systems such as KVM). It can consume a lot of processing power; use with care. See the Linux kernel source file `Documentation/ads/min-guide/mm/ksm.rst` for more details.

The `MADV_MERGEABLE` and `MADV_UNMERGEABLE` operations are available only if the kernel was configured with `CONFIG_KSM`.

`MADV_UNMERGEABLE` (since Linux 2.6.32)

Undo the effect of an earlier `MADV_MERGEABLE` operation on the specified address range; KSM unmerges whatever pages it had merged in the address range specified by `addr` and `size`.

`MADV_SOFT_OFFLINE` (since Linux 2.6.33)

Soft offline the pages in the range specified by `addr` and `size`. The memory of each page in the specified range is preserved (i.e., when next accessed, the same content will be visible, but in a new physical page frame), and the original page is offlined (i.e., no longer used, and taken out of normal memory management). The effect of the `MADV_SOFT_OFFLINE` operation is invisible to (i.e., does not change the semantics of) the calling process.

This feature is intended for testing of memory error-handling code; it is available only if the kernel was configured with `CONFIG_MEMORY_FAILURE`.

`MADV_HUGEPAGE` (since Linux 2.6.38)

Enable Transparent Huge Pages (THP) for pages in the range specified by `addr` and `size`. The kernel will regularly scan the areas marked as huge page candidates to replace them with huge pages. The kernel will also allocate huge pages directly when the region is naturally aligned to the huge page size (see `posix_memalign(2)`).

This feature is primarily aimed at applications that use large mappings of data and access large regions of that memory at a time (e.g., virtualization systems such as QEMU). It can very easily waste memory (e.g., a 2 MB mapping that only ever accesses 1 byte will result in 2 MB of wired memory instead of one 4 KB page). See the Linux kernel source file `Documentation/admin-guide/mm/transhuge.rst` for more details.

Most common kernels configurations provide `MADV_HUGEPAGE`-style behavior by default, and thus `MADV_HUGEPAGE` is normally not necessary. It is mostly intended for embedded systems, where `MADV_HUGEPAGE`-style behavior may not be enabled by default in the kernel. On such systems, this flag can be used in order to selectively enable THP.

Whenever `MADV_HUGEPAGE` is used, it should always be in regions of memory with an access pattern that the developer knows in advance won't risk to increase the memory footprint of the application when transparent hugepages are enabled.

Since Linux 5.4, automatic scan of eligible areas and replacement by huge pages works with private anonymous pages (see `mmap(2)`), `shmem` pages, and file-backed pages. For all memory types, memory may only be replaced by huge pages on hugepage-aligned boundaries. For file-mapped memory (including `tmpfs` (see `tmpfs(2)`)) the mapping must also be naturally hugepage-aligned within the file. Additionally, for file-backed, non-`tmpfs` memory, the file must not be open for write and the mapping must be executable.

The VMA must not be marked `VM_NOHUGEPAGE`, `VM_HUGETLB`, `VM_IO`, `VM_DONTEXPAND`, `VM_MIXEDMAP`, or `VM_PFNMAP`, nor can it be stack memory or backed by a DAX-enabled device (unless the DAX device is hot-plugged as System RAM). The process can have `PR_SET_THP_DISABLE` set only if `PR_THP_DISABLE_EXCEPT_ADVISED` is set (see `prctl(2)`).

The `MADV_HUGEPAGE`, `MADV_NOHUGEPAGE`, and `MADV_COLLAPSE` operations are available only if the kernel was configured with `CONFIG_TRANSPARENT_HUGEPAGE` and file/`shmem` memory is only supported if the kernel was configured with `CONFIG_READ_ONLY_THP_FOR_FS`.

`MADV_NOHUGEPAGE` (since Linux 2.6.38)

Ensures that memory in the address range specified by `addr` and `size` will not be backed by transparent hugepages.

MADV_COLLAPSE (since Linux 6.1)

Perform a best-effort synchronous collapse of the native pages mapped by the memory range into Transparent Huge Pages (THPs). MADV_COLLAPSE operates on the current state of memory of the calling process and makes no persistent changes or guarantees on how pages will be mapped, constructed, or faulted in the future.

MADV_COLLAPSE supports private anonymous pages (see `mmap(2)`), `shmem` pages, and file-backed pages. See MADV_HUGEPAGE for general information on memory requirements for THP. If the range provided spans multiple VMAs, the semantics of the collapse over each VMA is independent from the others. If collapse of a given huge page-aligned/sized region fails, the operation may continue to attempt collapsing the remainder of the specified memory. MADV_COLLAPSE will automatically clamp the provided range to be hugepage-aligned.

All non-resident pages covered by the range will first be swapped/faulted-in, before being copied onto a freshly allocated hugepage. If the native pages compose the same PTE-mapped hugepage, and are suitably aligned, allocation of a new hugepage may be elided and collapse may happen in-place. Unmapped pages will have their data directly initialized to 0 in the new hugepage. However, for every eligible hugepage-aligned/sized region to be collapsed, at least one page must currently be backed by physical memory.

MADV_COLLAPSE is independent of any `sysfs` (see `sysfs(5)`) setting under `/sys/kernel/mm/transparent_hugepage`, both in terms of determining THP eligibility, and allocation semantics. See Linux kernel source file `Documentation/admin-guide/mm/transparent_hugepage.rst` for more information. MADV_COLLAPSE also ignores `huge= tmpfs` mount when operating on `tmpfs` files. Allocation for the new hugepage may enter direct reclaim and/or compaction, regardless of VMA flags (though `VM_NOHUGEPAGE` is still respected). When the system has multiple NUMA nodes, the hugepage will be allocated from the node providing the most native pages.

If all hugepage-sized/aligned regions covered by the provided range were either successfully collapsed, or were already PMD-mapped THPs, this operation will be deemed successful. Note that this doesn't guarantee anything about other possible mappings of the memory. In the event multiple hugepage-aligned/sized areas fail to collapse, only the most-recently failed code will be set in `errno`.

MADV_DONTDUMP (since Linux 3.4)

Exclude from a core dump those pages in the range specified by `addr` and `size`. This is useful in applications that have large areas of memory that are known not to be useful in a core dump. The effect of `MADV_DONTDUMP` takes precedence over the bit mask that is set via the `/proc/pid/coredump_filter` file (see `core(5)`).

`MADV_DODUMP` (since Linux 3.4)

Undo the effect of an earlier `MADV_DONTDUMP`.

`MADV_FREE` (since Linux 4.5)

The application no longer requires the pages in the range specified by `addr` and `size`. The kernel can thus free these pages, but the freeing could be delayed until memory pressure occurs. For each of the pages that has been marked to be freed but has not yet been freed, the free operation will be canceled if the caller writes into the page. After a successful `MADV_FREE` operation, any stale data (i.e., dirty, unwritten pages) will be lost when the kernel frees the pages. However, subsequent writes to pages in the range will succeed and then kernel cannot free those dirtied pages, so that the caller can always see just written data. If there is no subsequent write, the kernel can free the pages at any time. Once pages in the range have been freed, the caller will see zero-fill-on-demand pages upon subsequent page references.

The `MADV_FREE` operation can be applied only to private anonymous pages (see `mmap(2)`).

Before Linux 4.12, when freeing pages on a swapless system, the pages in the given range are freed instantly, regardless of memory pressure.

`MADV_WIPEONFORK` (since Linux 4.14)

Present the child process with zero-filled memory in this range after a `fork(2)`.

This is useful in forking servers in order to ensure that sensitive per-process data (for example, PRNG seeds, cryptographic secrets, and so on) is not handed to child processes.

The `MADV_WIPEONFORK` operation can be applied only to private anonymous pages (see `mmap(2)`).

Within the child created by `fork(2)`, the `MADV_WIPEONFORK` setting remains in place on the specified address range. This setting is cleared during `execve(2)`.

`MADV_KEEPPONFORK` (since Linux 4.14)

Undo the effect of an earlier `MADV_WIPEONFORK`.

`MADV_COLD` (since Linux 5.4)

Deactivate a given range of pages. This will make the pages a more probable reclaim

target should there be a memory pressure. This is a nondestructive operation. The advice might be ignored for some pages in the range when it is not applicable.

`MADV_PAGEOUT` (since Linux 5.4)

Reclaim a given range of pages. This is done to free up memory occupied by these pages. If a page is anonymous, it will be swapped out. If a page is file-backed and dirty, it will be written back to the backing storage. The advice might be ignored for some pages in the range when it is not applicable.

`MADV_POPULATE_READ` (since Linux 5.14)

"Populate (prefault) page tables readable, faulting in all pages in the range just as if manually reading from each page; however, avoid the actual memory access that would have been performed after handling the fault.

In contrast to `MAP_POPULATE`, `MADV_POPULATE_READ` does not hide errors, can be applied to (parts of) existing mappings and will always populate (prefault) page tables readable. One example use case is prefaulting a file mapping, reading all file content from disk; however, pages won't be dirtied and consequently won't have to be written back to disk when evicting the pages from memory.

Depending on the underlying mapping, map the shared zeropage, preallocate memory or read the underlying file; files with holes might or might not preallocate blocks. If populating fails, a `SIGBUS` signal is not generated; instead, an error is returned.

If `MADV_POPULATE_READ` succeeds, all page tables have been populated (prefaulted) readable once. If `MADV_POPULATE_READ` fails, some page tables might have been populated.

`MADV_POPULATE_READ` cannot be applied to mappings without read permissions and special mappings, for example, mappings marked with kernel-internal flags such as `VM_PFNMAP` or `VM_IO`, or secret memory regions created using `memfd_secret(2)`.

Note that with `MADV_POPULATE_READ`, the process can be killed at any moment when the system runs out of memory.

`MADV_POPULATE_WRITE` (since Linux 5.14)

Populate (prefault) page tables writable, faulting in all pages in the range just as if manually writing to each each page; however, avoid the actual memory access that would have been performed after handling the fault.

In contrast to `MAP_POPULATE`, `MADV_POPULATE_WRITE` does not hide errors, can be applied to (parts of) existing mappings and will always populate (prefault) page tables

writable. One example use case is preallocating memory, breaking any CoW (Copy on Write).

Depending on the underlying mapping, preallocate memory or read the underlying file; files with holes will preallocate blocks. If populating fails, a SIGBUS signal is not generated; instead, an error is returned.

If MADV_POPULATE_WRITE succeeds, all page tables have been populated (prefaulted) writable once. If MADV_POPULATE_WRITE fails, some page tables might have been populated.

MADV_POPULATE_WRITE cannot be applied to mappings without write permissions and special mappings, for example, mappings marked with kernel-internal flags such as VM_PFNMAP or VM_IO, or secret memory regions created using memfd_secret(2).

Note that with MADV_POPULATE_WRITE, the process can be killed at any moment when the system runs out of memory.

MADV_GUARD_INSTALL (since Linux 6.13)

Install a lightweight guard region into the range specified by addr and size, causing any read or write in the range to result in a SIGSEGV signal being raised.

If the region maps memory pages those mappings will be replaced as part of the operation, though if MADV_GUARD_INSTALL is applied to regions containing pre-existing lightweight guard regions, they are left in place.

Prior to Linux 6.15, this operation was supported only for writable anonymous private mappings. Since Linux 6.15, both anonymous and file-backed mappings are supported, including read-only mappings.

The mapping must not be mlock'd, map hugetlb ranges, nor contain special mappings. For example, mappings marked with kernel-internal flags such as VM_PFNMAP or VM_IO, or secret memory regions created using memfd_secret(2).

An EINVAL error is returned if it is attempted on any other kind of mapping.

This operation is more efficient than mapping a new region of memory PROT_NONE, as it does not require the establishment of new mappings. Instead, regions of an existing mapping simply have their page tables manipulated to establish the desired behavior.

No additional memory is used.

Lightweight guard regions remain on fork (except for any parts which have had MADV_WIPEONFORK applied to them), and are not removed by MADV_DONTNEED, MADV_FREE, MADV_PAGEOUT, or MADV_COLD.

Attempting to `mlock(2)` lightweight guard regions will fail, as will `MADV_POPU?`

`LATE_READ` or `MADV_POPULATE_WRITE`.

If the mapping has its attributes changed, or is split or partially unmapped, any ex?

isting guard regions remain in place (except if they are unmapped).

If a mapping is moved using `mremap(2)`, lightweight guard regions are moved with it.

Lightweight guard regions are removed when unmapped, on process teardown, or when the

`MADV_GUARD_REMOVE` operation is applied to them.

`MADV_GUARD_REMOVE` (since Linux 6.13)

Remove any lightweight guard regions which exist in the range specified by `addr` and `size`.

All mappings in the range other than lightweight guard regions are left in place.

The operation is supported on those mappings permitted by `MADV_GUARD_INSTALL` in addition to `mlock()`'d mappings, returning an `EINVAL` error otherwise.

When lightweight guard regions are removed, they act as empty regions of the containing mapping. Therefore, anonymous private mappings become zero-fill-on-demand pages, and file-backed mappings are repopulated with the memory contents from the up-to-date contents of the underlying mapped file.

If any transparent huge pages are encountered in the operation, they are left in place.

RETURN VALUE

On success, `madvise()` returns zero. On error, it returns -1 and `errno` is set to indicate the error.

ERRORS

`EACCES` advice is `MADV_REMOVE`, but the specified address range is not a shared writable mapping.

`EAGAIN` A kernel resource was temporarily unavailable.

`EBADF` The map exists, but the area maps something that isn't a file.

`EBUSY` (for `MADV_COLLAPSE`) Could not charge hugepage to cgroup: cgroup limit exceeded.

`EBUSY` (for `MADV_SOFT_OFFLINE`) Any pages within the specified address range could not be offline'd. This might occur if the page is currently in use or locked.

`EFAULT` advice is `MADV_POPULATE_READ` or `MADV_POPULATE_WRITE`, and populating (prefaulting) page tables failed because a `SIGBUS` would have been generated on actual memory access and the reason is not a HW poisoned page (HW poisoned pages can, for example, be created).

ated using the MADV_HWPOISON flag described elsewhere in this page).

EINVAL addr is not page-aligned or size is negative.

EINVAL advice is not a valid.

EINVAL advice is MADV_COLD or MADV_PAGEOUT and the specified address range includes locked, Huge TLB pages, or VM_PFNMAP pages.

EINVAL advice is MADV_DONTNEED or MADV_REMOVE and the specified address range includes locked, Huge TLB pages, or VM_PFNMAP pages.

EINVAL advice is MADV_MERGEABLE or MADV_UNMERGEABLE, but the kernel was not configured with CONFIG_KSM.

EINVAL advice is MADV_FREE or MADV_WIPEONFORK but the specified address range includes file, Huge TLB, MAP_SHARED, or VM_PFNMAP ranges.

EINVAL advice is MADV_POPULATE_READ or MADV_POPULATE_WRITE, but the specified address range includes ranges with insufficient permissions or special mappings, for example, mappings marked with kernel-internal flags such as VM_IO or VM_PFNMAP, or secret memory regions created using memfd_secret(2).

EINVAL advice is MADV_GUARD_INSTALL or MADV_GUARD_REMOVE, but the specified address range contains an unsupported mapping.

EIO (for MADV_WILLNEED) Paging in this area would exceed the process's maximum resident set size.

ENOMEM (for MADV_WILLNEED) Not enough memory: paging in failed.

ENOMEM (for MADV_COLLAPSE) Not enough memory: could not allocate hugepage.

ENOMEM Addresses in the specified range are not currently mapped, or are outside the address space of the process.

ENOMEM advice is MADV_POPULATE_READ or MADV_POPULATE_WRITE, and populating (prefaulting) page tables failed because there was not enough memory.

EPERM advice is MADV_HWPOISON, but the caller does not have the CAP_SYS_ADMIN capability.

EHWPOISON

advice is MADV_POPULATE_READ or MADV_POPULATE_WRITE, and populating (prefaulting) page tables failed because a HW poisoned page (HW poisoned pages can, for example, be created using the MADV_HWPOISON flag described elsewhere in this page) was encountered.

VERSIONS

Versions of this system call, implementing a wide variety of advice values, exist on many

other implementations. Other implementations typically implement at least the flags listed above under Conventional advice flags, albeit with some variation in semantics.

POSIX.1-2001 describes `posix_madvise(3)` with constants `POSIX_MADV_NORMAL`, `POSIX_MADV_RANDOM`, `POSIX_MADV_SEQUENTIAL`, `POSIX_MADV_WILLNEED`, and `POSIX_MADV_DONTNEED`, and so on, with behavior close to the similarly named flags listed above.

Linux

The Linux implementation requires that the address `addr` be page-aligned, and allows `size` to be zero. If there are some parts of the specified address range that are not mapped, the Linux version of `madvise()` ignores them and applies the call to the rest (but returns `ENOMEM` from the system call, as it should).

`madvise(0, 0, advice)` will return zero iff `advice` is supported by the kernel and can be relied on to probe for support.

STANDARDS

None.

HISTORY

First appeared in 4.4BSD.

Since Linux 3.18, support for this system call is optional, depending on the setting of the `CONFIG_ADVICE_SYSCALLS` configuration option.

SEE ALSO

`getrlimit(2)`, `memfd_secret(2)`, `mincore(2)`, `mmap(2)`, `mprotect(2)`, `msync(2)`, `munmap(2)`, `prctl(2)`, `process_madvise(2)`, `posix_madvise(3)`, `core(5)`

Linux man-pages 6.17

2026-02-08

`madvise(2)`