madvise(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | VERSIONS |
CONFORMING TO | NOTES | SEE ALSO | COLOPHON

[Search online pages]

```
MADVISE(2)              Linux Programmer's Manual              MADVISE(2)
```

NAME   top

```
       madvise - give advice about use of memory
```

SYNOPSIS   top

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

       int madvise(void *addr, size_t length, 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   top

```
       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 length bytes 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
        shmem-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, Huge TLB
        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.)

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 byte range of the backing store (see
fallocate(2)). Subsequent accesses in the specified
address range will see bytes containing zero.

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

In the initial implementation, only tmpfs(5) was supported
MADV_REMOVE; but since Linux 3.5, any filesystem which
supports the fallocate(2) **FALLOC_FL_PUNCH_HOLE** mode also
supports **MADV_REMOVE**. Hugetlbfs fails with the error
EINVAL and other filesystems fail with the error
EOPNOTSUPP.

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 DMAs 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 *length*
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 *length*.   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/admin-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
>   *Length*.

MADV_SOFT_OFFLINE (since Linux 2.6.33)
>   Soft offline the pages in the range specified by *addr* and
>   *Length*.   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 *Length*. Currently, Transparent Huge
>   Pages work only with private anonymous pages (see
>   mmap(2)). 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.

The **MADV_HUGEPAGE** and **MADV_NOHUGEPAGE** operations are
available only if the kernel was configured with
CONFIG_TRANSPARENT_HUGEPAGE.

MADV_NOHUGEPAGE (since Linux 2.6.38)
        Ensures that memory in the address range specified by *addr*
        and *length* will not be backed by transparent hugepages.

MADV_DONTDUMP (since Linux 3.4)
        Exclude from a core dump those pages in the range
        specified by *addr* and *length*. 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 *len*. 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)). In Linux before version
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_KEEPONFORK (since Linux 4.14)
Undo the effect of an earlier **MADV_WIPEONFORK**.

RETURN VALUE   top

On success, **madvise**() returns zero. On error, it returns -1 and
errno is set appropriately.

ERRORS   top

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.

EINVAL *addr* is not page-aligned or *length* is negative.

EINVAL *advice* is not a valid.

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.

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

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

VERSIONS   top

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

CONFORMING TO   top

madvise() is not specified by any standards. 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.

NOTES   top

Linux notes
The Linux implementation requires that the address *addr* be page-
aligned, and allows *length* 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).

SEE ALSO   top

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

COLOPHON   top

This page is part of release 5.10 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/.

Linux   2020-04-11   MADVISE(2)

Pages that refer to this page: fork(2), getrlimit(2), ioctl_userfaultfd(2), mincore(2), open(2), prctl(2),
readahead(2), syscalls(2), userfaultfd(2), malloc_trim(3), posix_madvise(3), core(5), proc(5),
tmpfs(5), capabilities(7)

Copyright and license for this manual page

HTML rendering created 2020-12-21 by Michael Kerrisk, author of *The Linux
Programming Interface*, maintainer of the Linux *man-pages* project.

For details of in-depth **Linux/UNIX system programming training courses** that I
teach, look here.

Hosting by jambit GmbH.