

Bahir Dar university

**Faculty of computing**

Name biruk getachew

ID BDU1307550

Submitted to instructor wendimu

Submission date 22/10/2014 E.C

**Introduction to system call**

**system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system’s kernel.

* **Sum rule about system call**
* system calls cannot have floating-point parameters.
* system call runs on its own stack.
* **some parameters are passed in multiple registers**

**What is “int madvise(void\*adder ,size\_t length,int advice)”?**

The **madvise()** . It is mechanism interface between application and operating system about memory. in which a computer program give information about memory usage to the kernel of the OS. madvise communication of the operating system to the user programs via **API (Application Programming Interface**

It is function advises the kernel that a region of user virtual memory in the range starting at the address specified in addr and with length equal to the value of the len parameter is expected to follow a particular pattern of use. The kernel uses this information to optimize the procedure for manipulating and maintaining the resources associated with the specified range. Use of the **madvise()** function can increase system performance when used by programs that have specific knowledge of their access patterns over memory.

#include <sys/types.h>

#include <sys/mman.h>

int madvise(caddr\_t addr, size\_t len, int advice);

**Why is int madvise(void\*adder ,size\_t length,int advice) implemented?**

* The kernel uses this information to optimize the procedure for manipulating and maintaining the resources associated with the specified range. Use of the **madvise()** function can increase system performance when used by programs that have specific knowledge of their access patterns over memory.
* is system call that give advice about use of memory
* the goal of such advice is to improve system or application performance.
* The kernel uses this information to optimize the procedure for manipulating and maintaining the resources associated with the specified mapping range.

**How is “int madvise(void\*adder ,size\_t length,int advice)”?**

**Step 1)** The processes executed in the user mode till the time a madvice interrupts it.

**Step 2)**After that, the madvice is executed in the kernel-mode on a priority basis.

**Step 3)** Once madvice execution is over, control returns to the user mode.

**Step 4)**The execution of user processes resumed in Kernel mode.

2 describe briefly parameter and flags

Paramaters

madvise does not have direct access to the application's stack, all parameters for madvise must fit in eight registers. These parameter hold the below value.

* addr – address of memory
* length – length of segment
* advice – advice flag

The behavior argument is one the following flags defined in the header :

**MADV\_NORMAL**

* Removes any previous advice and sets the default behavior. By default, the kernel tracks access patterns on data objects and performs I/Os based on process trends (that is, sequential versus random). Sequential trends cause larger "read-ahead" I/Os, while random accesses reduce the amount of I/O to avoid unnecessary I/O.

**MADV\_RANDOM**

* Informs the kernel that any objects mapped in this range will be accessed in a random matter. The kernel will read only the minimal amount of data to satisfy the user fault.

**MADV\_SEQUENTIAL**

* Informs the kernel that any objects mapped in this range will be accessed in a sequential matter. The kernel will perform the maximum read-ahead for every fault. The kernel does not pay attention to access patterns and trends, but instead assumes sequentiality for every access on the object.

**MADV\_DONTNEED**

* Informs the kernel that the specified range is no longer needed by the process. This allows the kernel to release the physical pages associated with an address range back to the system for use by other processes. MADV\_DONTNEED is restricted to object ranges created with calls to mmap() and shmat(). Attempting to use MADV\_DONTNEED on an object that was not created using a call to mmap() or shmat() will result in [EINVAL] being returned to the caller.

**MADV\_WILLNEED**

* Tell the system that a certain address range is definitely needed so the kernel will start reading the specified range into memory. This can benefit programs wanting to minimize the time needed to access memory the first time, as the kernel would need to read in from the file

**MADV\_REMOVE** (Since Linux 2.6.16)

* Free up a given range of pages and its associated backing store. Currently, only shmfs/tmpfs supports this; other file systems return with the error **ENOSYS**.

**MADV\_DONTFORK** (Since Linux 2.6.16)

* Do not make the pages in this range available to the child after a [**fork**](https://linux.die.net/man/2/fork)(2). This is useful to prevent copy-on-write semantics from changing the physical location of a **page**(s) if the parent writes to it after a [**fork**](https://linux.die.net/man/2/fork)(2). (Such page relocations cause problems for hardware that DMAs into the **page**(s).)

**MADV\_DOFORK** (Since Linux 2.6.16)

* Undo the effect of **MADV\_DONTFORK**, restoring the default behavior, whereby a mapping is inherited across [**fork**](https://linux.die.net/man/2/fork)(2).

**MADV\_HWPOISON** (Since Linux 2.6.32)

* Poison a page and handle it like a hardware memory corruption. This operation is only available 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 only available if the kernel was configured with **CONFIG\_MEMORY\_FAILURE**.

**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 only available 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 only merges private anonymous pages (see [**mmap**](https://linux.die.net/man/2/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/vm/ksm.txt* for more details. The **MADV\_MERGEABLE** and **MADV\_UNMERGEABLE** operations are only available 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\_HUGEPAGE** (since Linux 2.6.38)

* Enables Transparent Huge Pages (THP) for pages in the range specified by *addr* and *length*. Currently, Transparent Huge Pages only work with private anonymous pages (see [**mmap**](https://linux.die.net/man/2/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 2MB mapping that only ever accesses 1 byte will result in 2MB of wired memory instead of one 4KB page). See the Linux kernel source file *Documentation/vm/transhuge.txt* for more details. The **MADV\_HUGEPAGE** and **MADV\_NOHUGEPAGE** operations are only available 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 collapsed into huge pages.

**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**](https://linux.die.net/man/5/core)(5)).

**MADV\_DODUMP** (since Linux 3.4)

Undo the effect of an earlier **MADV\_DONTDUMP**.

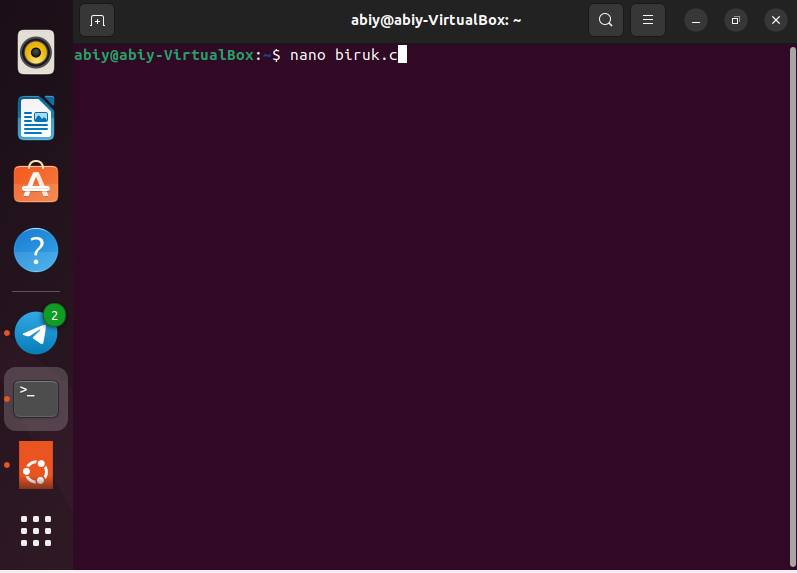
RETURN VALUE

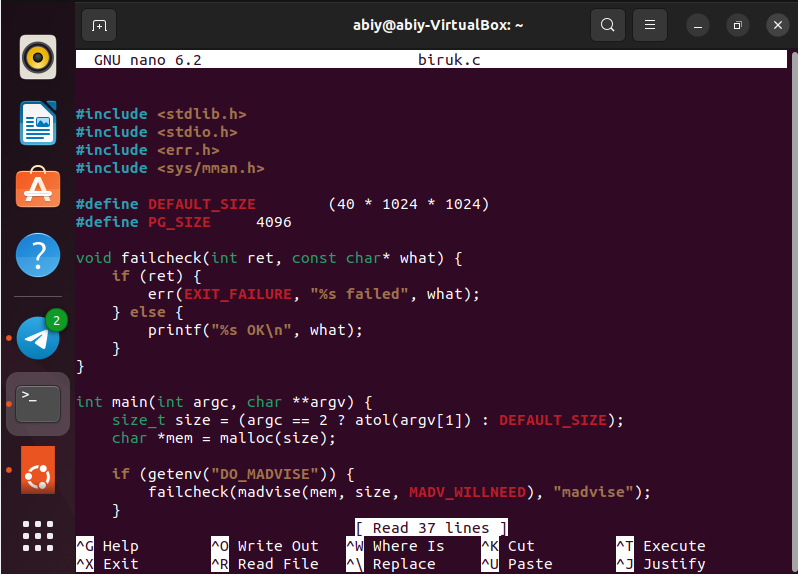
* madvise() returns the following values:
* 0 Successful completion.
* -1 Failure.
* errno is set to indicate the error

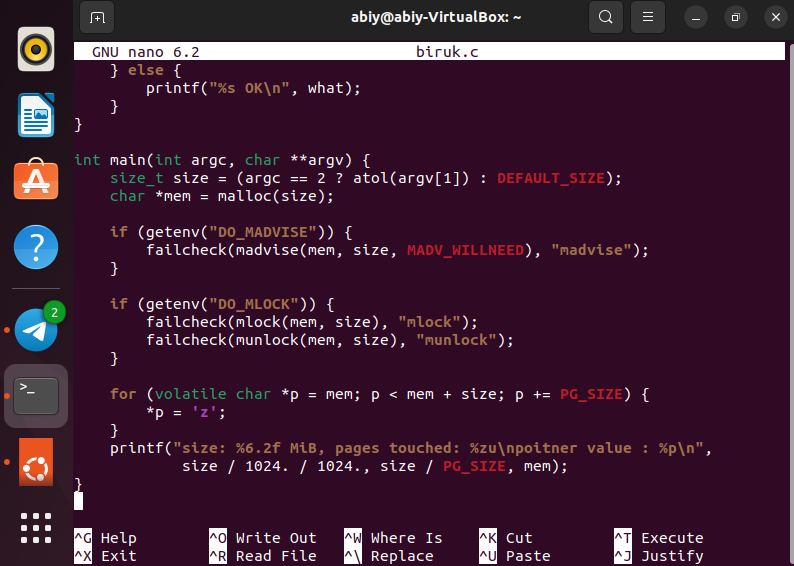
## ERRORS

* **EAGAIN**
* A kernel resource was temporarily unavailable.
* **EBADF**
* The map exists, but the area maps something that isn't a file.
* **EINVAL**
* This error can occur for the following reasons:
* The value *len* is negative.
* *addr* is not page-aligned.
* *advice* is not a valid value
* The application is attempting to release locked or shared pages (with **MADV\_DONTNEED**).
* **MADV\_MERGEABLE** or **MADV\_UNMERGEABLE** was specified in *advice*, but the kernel was not configured with **CONFIG\_KSM**.
* **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.

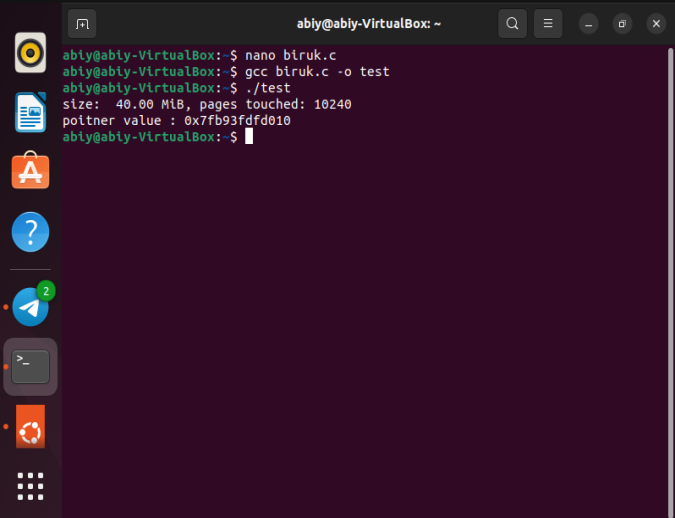
Implemetation of MADV\_WILLNEED







Out put



* Implemention of MADV\_DONTNEED

#include <sys/mman.h>

#include <stdio.h>

#include <stddef.h>

#include <assert.h>

#include <string.h>

int main(int argc, char \*\*argv)

{

    void \*addr = mmap(NULL, 1<<16, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0);

    if (addr == MAP\_FAILED) {

        perror("mmap");

        return 1;

    }

    memset(addr, 'A', 1<<16);

    if (!madvise(addr, 1<<16, MADV\_DONTNEED)) {

        puts("MADV\_DONTNEED does not return error. Check memory.");

        for (int i = 0; i < 1<<16; ++i) {

            assert(((unsigned char \*)addr)[i] == 0);

        }

    } else {

        perror("madvise");

    }

    if (munmap(addr, 1<<16)) {

        perror("munmap");

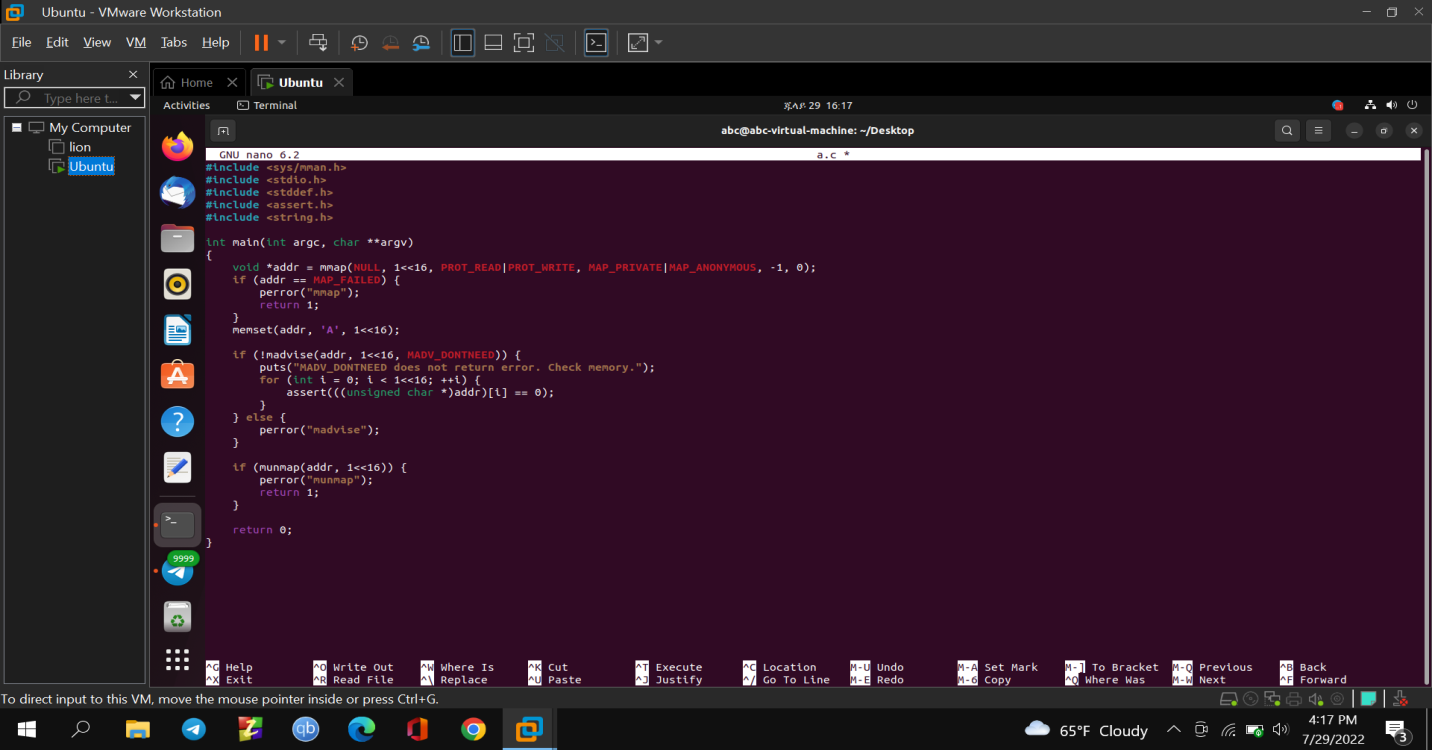
        return 1;

    }

    return 0;

}

Implementation



Out put

