



BAHIR DAR UNIVERSITY  
FACULTY OF COMPUTING  
DEPARTMENT OF SOFTWARE ENGINEERING  
OPERATING SYSTEM AND SYSTEM PROGRAMMING  
INDIVIDUAL ASSIGNMENT

Title: swapon system call  
Submitted to: Wendimu B.  
Submitted by: Abrham Abayneh  
Id: BDU1311576

---

# System call

## Contents

<b>Introduction.....</b>	<b>1</b>
What is system call?.....	1
How can we write system call?.....	1
How does system call work?.....	1
What services can system call provide? .....	1
<b>What is swapon ? .....</b>	<b>2</b>
What is swap file ? .....	2
What is free command ? .....	2
What is swapon command ? .....	2
<b>Why do we use swapon() ? .....</b>	<b>3</b>
<b>How does this system call work ? .....</b>	<b>3</b>
How does it work ? (step by step ).....	3
<b>List of parameters and flags.....</b>	<b>5</b>
What is const char *path ? .....	5
What is int swapflags ? .....	5
<b>What are flags ?.....</b>	<b>5</b>
SWAP_FLAG_PREFER .....	5
SWAP_FLAG_DISCARD .....	5
Priority.....	5
<b>How can we implement this system call ? .....</b>	<b>6</b>
What kinds of errors might we face ? .....	7

---

# System call

## Introduction

### What is system call?

A system call in computing is the programmatic method by which a computer application asks the kernel of the operating system it is running on for a service. Programs can communicate with the operating system by making a system call. When a computer application requests anything from the kernel of the operating system, it performs a system call. System call uses Application Programming Interfaces to provide operating system services to user programs (API). It offers a point of contact between processes and the operating system so that user-level processes can ask for the operating system's assistance. The only ways to access the kernel system are through system calls. System calls are required for any programs that use resources.

### How can we write system call?

A system call can be written in high-level languages like C or Pascal or in assembly code. If a high-level language is employed, the operating system may directly invoke system calls, which are predefined functions.

### How does system call work?

In our computer, we have user mode and kernel mode. In user mode, work is done on behalf of the user, and in kernel/monitor mode, work is done on behalf of the operating system.

When a process is under execution, process executes under user mode, but when it requires some operating system resources to complete the task, the process makes a system call. Once we call the system to complete the task, it will be executed in kernel mode on a priority basis. On completion of the system call, the control is passed to process in user mode. And the process continues the execution in user mode.

System calls fall into 5 different categories. which is:

1. Process creation
2. File management
3. Device management
4. Information maintenance and
5. Communication.

### What services can system call provide?

- ✓ Process creation and management
- ✓ Main memory management
- ✓ File Access, Directory and File system management
- ✓ Device handling
- ✓ Networking, etc.

We will now discuss on one of the system calls, `swapon()`, in considerable detail.

## System call

### What is swapon ?

#### What is swap file ?

Before we get into the swapon function, let's first define what a swap file is. In general, a swap file is a system file that generates temporary storage space on a hard drive or solid-state drive when the system is running low on memory. The file frees up memory for other programs by swapping a portion of RAM storage from an inactive program.

The operating system can seem to have more RAM than it actually does by using swap files, a sort of virtual memory that is not stored in physical RAM. New files or applications can be "swapped in" to RAM while the least recently used ones in RAM are "swapped out" to the hard drive until they are later needed.

The operating system manages creating and using a swap file as necessary, therefore how swap files are implemented differs. Swap files for Windows differ from those in Linux.

The swap space in Linux comes in two forms. The swap partition is one example; it is a separate area or partition on the drive. The second is the swap file, which can be easily resized and is located between system and data files. There is no need for a separate partition when using a swap file. Instead, the system uses a file that the user creates under the root as the swap space.

One of the two commands, free or swapon, can be used to check the amount of swap space already present on a Linux system.

#### What is free command ?

A summary of RAM utilization, including total, used, free, shared, and accessible memory as well as swap space, is produced by the Linux free command. The command enables an administrator to keep track of resource utilization and decide whether there is adequate space to launch additional programs.

The swap space's size is displayed using the free command. It doesn't say if the space is a swap partition or a swap file, though. The swapon command works better for this.

#### What is swapon command ?

swapon is one of the basic system call that sets the swap area to the file or block device specified by path. The special file parameter specifies the tool or file that is being used.

A paging space is activated by the swapon command. It is used to make the initial paging space available during early system initialization. The swapon -a command is used to make extra devices available during a later stage of system initialization so that paging and swapping activity is distributed over numerous devices. If auto=yes, any devices listed in /etc/swapspaces that aren't specifically excluded from being automatically switched on by their stanza are made available by the swapon -a command. The /etc/rc file that is used for system multiuser initialization typically contains calls to the swapon command.

## System call

### Why do we use swapon() ?

We use a swap file in order to make the computer able to use more memory than is physically installed. In other words, it can run more programs than it could run with just the limited resources of the installed RAM.

Swap files play two important roles in computer processing:

They can handle the extra load of memory consumption. Some applications consume a huge amount of RAM, depriving other background applications that also need memory. Swap files help resolve this situation by paging out idle files to provide extra virtual memory.

Unforeseen circumstances. When a particular program consumes extra memory, or when extra space is needed for device operations, swap files provide some breathing room until the user can come up with a more permanent solution, such as a RAM upgrade.

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

### How does this system call work ?

When the system's memory is low, a swap file generates temporary storage space on a solid-state drive or hard disk. The file frees up memory for other programs by swapping a portion of RAM storage from an inactive program.

The computer can use more RAM than is actually installed by employing a swap file. In other words, it can run more applications than it could with just the installed RAM's constrained resources.

Swap files are a form of virtual memory because they are not kept in actual RAM. A computer's operating system (OS) can seem to have more RAM than it actually does by using a swap file. New files or applications can be "swapped in" to RAM while the least recently used ones in RAM are "swapped out" to the hard drive until they are later needed.

The OS manages creating and using a swap file as necessary, therefore how swap files are implemented differs.

### How does it work ? (step by step )

Step 1: open terminal and swapon –show command to see if there is existing swap file

## System call

```
abrrsh@fedora:~  
[abrrsh@fedora ~]$ swapon --show  
NAME          TYPE          SIZE USED PRIO  
/dev/zram0 partition 2.9G  1M  100  
[abrrsh@fedora ~]$
```

As we can see there is no swap file available currently, so let's create it.

Step 2: type "sudo dd if=/dev/zram0 of=/swapfile bs=1024 count=1048576"

If you don't understand what you have typed, this is the breakdown

- if = the input file (/dev/zram0)
- of = the output file (/swapfile), the name swapfile can be anything you want.
- bs = block size
- Count = amount of blocks to read and write

```
abrrsh@fedora:~  
[abrrsh@fedora ~]$ sudo dd if=/dev/zram0 of=/swapfile bs=1024 count=104857  
104857+0 records in  
104857+0 records out  
107373568 bytes (107 MB, 102 MiB) copied, 1.15806 s, 92.7 MB/s  
[abrrsh@fedora ~]$
```

We can check on our new swap using the command "ls /"

Step 3: Now let us prepare our swap file

On your terminal type "sudo chmod 600 /swapfile" to give permission for mkswap that we will use on the next step.

Once you have done that, we can create our swap file by typing "sudo mkswap /swapfile". But what is mkswap ? well, mkswap is a system call that will prepare our path or partition.

```
abrrsh@fedora:~  
[abrrsh@fedora ~]$ sudo dd if=/dev/zram0 of=/swapfile bs=1024 count=104857  
104857+0 records in  
104857+0 records out  
107373568 bytes (107 MB, 102 MiB) copied, 1.15806 s, 92.7 MB/s  
[abrrsh@fedora ~]$ ls /  
afs boot etc lib lost+found mnt proc run snap swapfile tmp var  
bin dev home lib64 media opt root sbin srv sys usr  
[abrrsh@fedora ~]$ sudo chmod 600 /swapfile  
[sudo] password for abrrsh:  
[abrrsh@fedora ~]$ sudo mkswap /swapfile  
mkswap: /swapfile: warning: wiping old swap signature.  
Setting up swspace version 1, size = 102.4 MiB (107368448 bytes)  
no label, UUID=49d54d2d-918b-46a3-9296-c94f67b98749  
[abrrsh@fedora ~]$
```

Step 4: Activating our swap file

In order to activate our swapfile we will use the swapon command as "sudo swapon /swapfile"

Then if we want to check it, we can go for "swapon -show".

## System call

### List of parameters and flags

In swapon system call we have two parameters, these are `const char *path` and `int swapflags`.

We have to pass the parameters in order to activate our swap file.

#### What is `const char *path` ?

This is the path or directory with the data type pointer of characters that hold the path that our swap file is located. So, we need to pass this when calling the swapon function.

#### What is `int swapflags` ?

This parameter with the data type integer needs to be passed in order to give higher priority than the default. We will try to see what these flags are in the next section.

### What are flags ?

We have two flags in swapon system call. These are `SWAP_FLAG_PREFER` and `SWAP_FLAG_DISCARD`.

#### `SWAP_FLAG_PREFER`

the new swap region will be given a higher priority than the default if flag is set in the swapon() swapflags parameter. Within swapflags, the priority is represented as follows:

*`(prio << SWAP_FLAG_PRIO_SHIFT) & SWAP_FLAG_PRIO_MASK`*

#### `SWAP_FLAG_DISCARD`

If the swap device supports the discard or trim operation and the flag is given in the swapon() swapflags parameter, freed swap pages will be discarded before being reused. (Some Solid-State Devices may have improved performance with this, but frequently not.)

#### Priority

There is a high or low priority assigned to each swap region. The priority is low by default. Newer locations are even less important than older places inside the low-priority areas.

With swapflags, all priorities are set to high priority levels that are higher than the default. Any nonnegative value that the caller chooses is allowed for them. Priority increases with larger numbers.

Swap pages are distributed among sections according to importance, starting with the greatest priority. Before employing a lower-priority area for areas with varying priorities, the higher-priority area is first used.

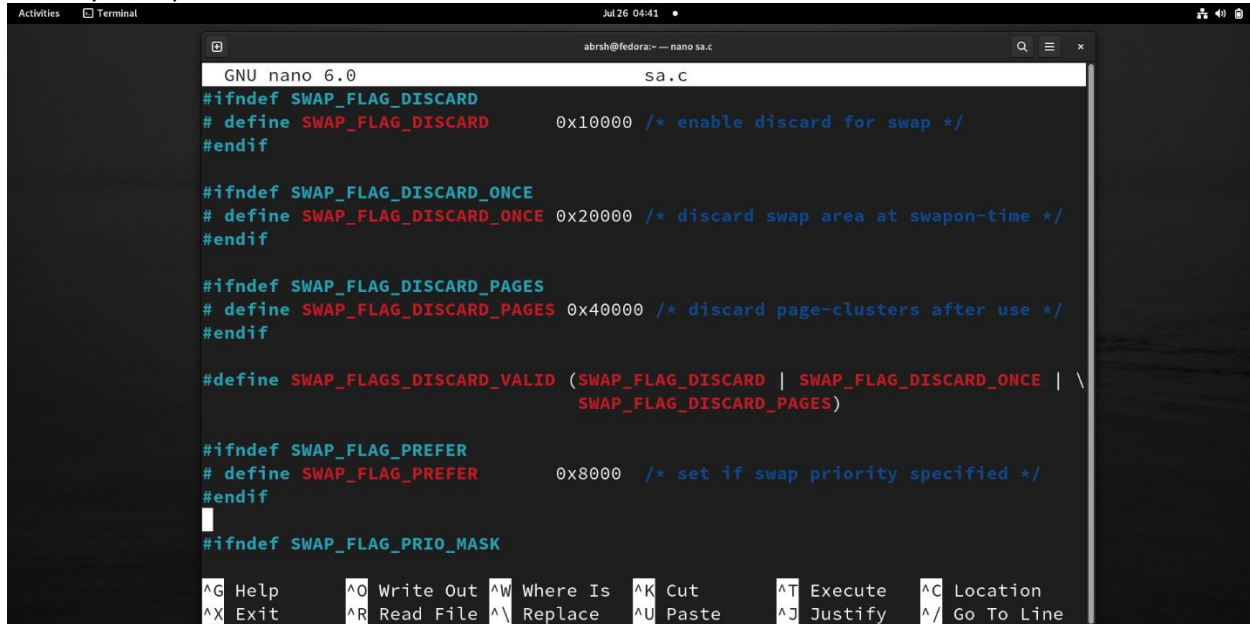
Pages are divided evenly across two or more sections if they have the same priority and it is the highest priority that is available.

## System call

### How can we implement this system call ?

These functions are Linux-specific and should not be used in programs intended to be portable. The source code is available in the internet by c language.

Let's try to implement it.

A screenshot of a terminal window with the nano text editor open. The editor is editing a file named 'sa.c'. The code visible includes several preprocessor directives for swap flags: SWAP\_FLAG\_DISCARD, SWAP\_FLAG\_DISCARD\_ONCE, SWAP\_FLAG\_DISCARD\_PAGES, SWAP\_FLAGS\_DISCARD\_VALID, SWAP\_FLAG\_PREFER, and SWAP\_FLAG\_PRIO\_MASK. The code is color-coded with red for keywords like #define and #endif, and blue for comments. The bottom of the screen shows the nano editor's command palette with options like Help, Write Out, Where Is, Cut, Execute, Location, Exit, Read File, Replace, Paste, Justify, and Go To Line.

```
GNU nano 6.0 sa.c
#ifndef SWAP_FLAG_DISCARD
# define SWAP_FLAG_DISCARD 0x10000 /* enable discard for swap */
#endif

#ifndef SWAP_FLAG_DISCARD_ONCE
# define SWAP_FLAG_DISCARD_ONCE 0x20000 /* discard swap area at swapon-time */
#endif

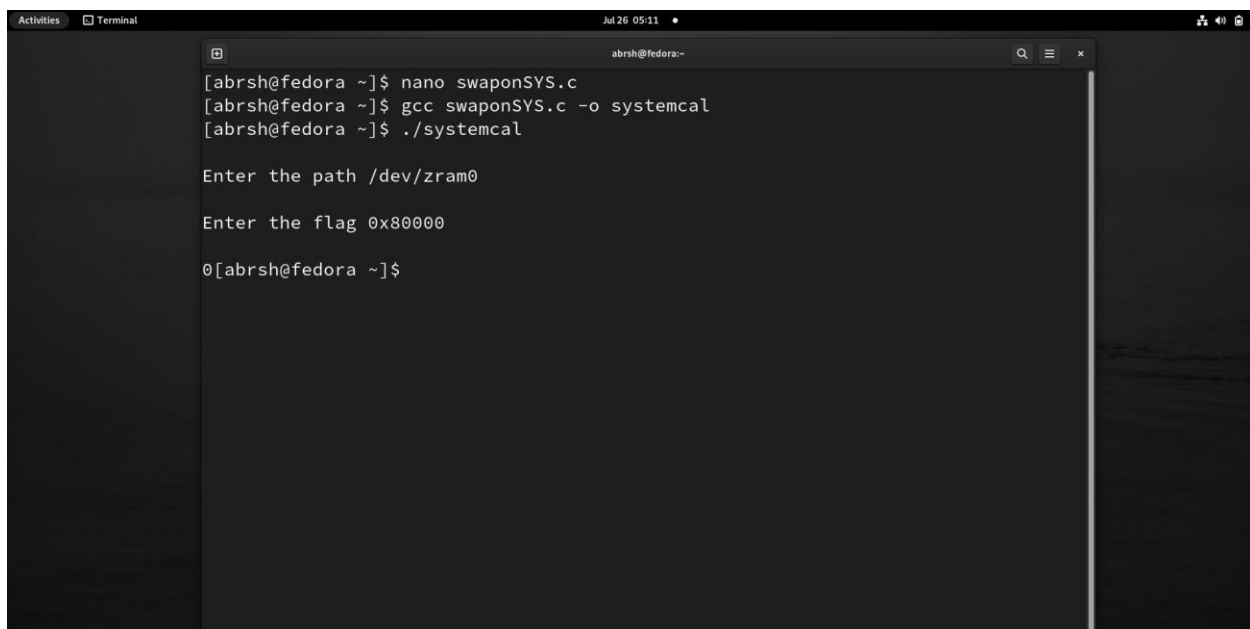
#ifndef SWAP_FLAG_DISCARD_PAGES
# define SWAP_FLAG_DISCARD_PAGES 0x40000 /* discard page-clusters after use */
#endif

#define SWAP_FLAGS_DISCARD_VALID (SWAP_FLAG_DISCARD | SWAP_FLAG_DISCARD_ONCE | \
SWAP_FLAG_DISCARD_PAGES)

#ifndef SWAP_FLAG_PREFER
# define SWAP_FLAG_PREFER 0x80000 /* set if swap priority specified */
#endif

#ifndef SWAP_FLAG_PRIO_MASK
```

The figure above shows a little of the c code. It returns 0, on success and -1 on error.

A screenshot of a terminal window showing the compilation and execution of a C program. The user runs 'nano swaponSYS.c' to edit the file, then 'gcc swaponSYS.c -o systemcal' to compile it, and finally './systemcal' to run it. The program prompts the user to enter a path and a flag. The user enters '/dev/zram0' and '0x80000', and the program returns '0'.

```
[abrsh@fedora ~]$ nano swaponSYS.c
[abrsh@fedora ~]$ gcc swaponSYS.c -o systemcal
[abrsh@fedora ~]$ ./systemcal

Enter the path /dev/zram0

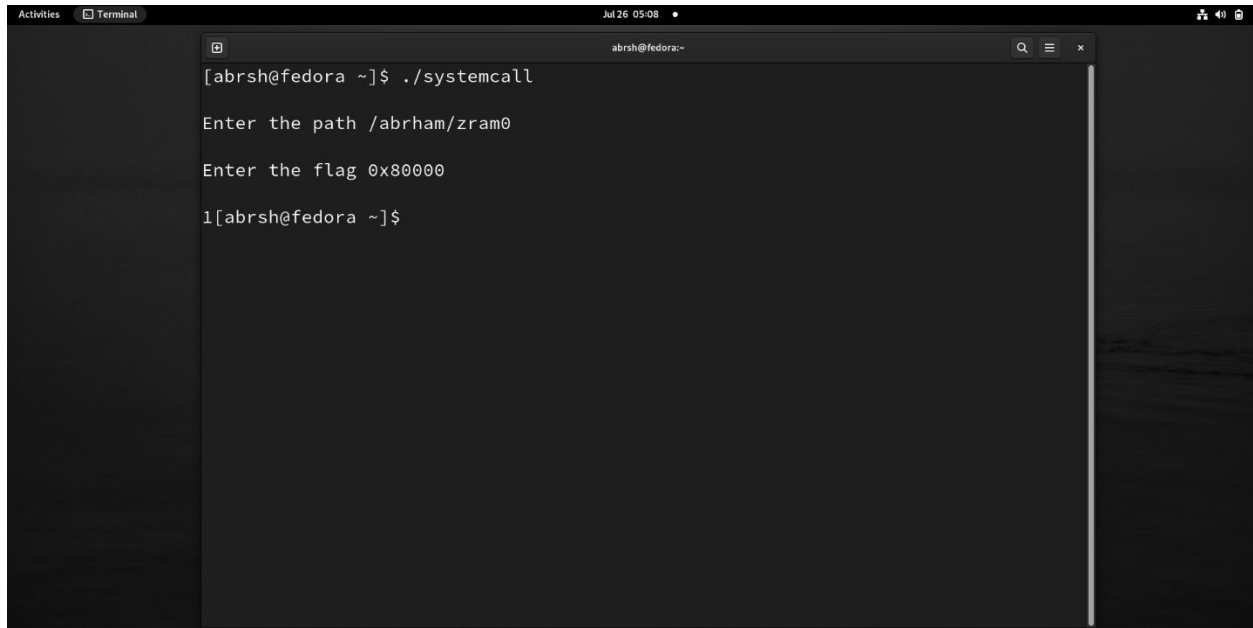
Enter the flag 0x80000

0[abrsh@fedora ~]$
```

Let's try to make it an invalid path.



## System call



```
Activities Terminal Jul 26 05:08 aabrsh@fedora:~  
[aabrsh@fedora ~]$ ./systemcall  
Enter the path /aabrham/zram0  
Enter the flag 0x80000  
1[aabrsh@fedora ~]$
```

### What kinds of errors might we face ?

EBUSY (For swapon()): when a swap area is already in use on the supplied path.

EINVAL: The file path is valid but does not point to a block device or a conventional file;

EINVAL (swapon()): The provided path is either on an in-memory filesystem like tmpfs or does not have a valid swap signature (5).

EINVAL (since Linux 3.4) (swapon()): The swapflags file had an incorrect flag value.

EINVAL (swapon()): route is not a swap area at this time.

ENFILE: The maximum number of open files on the system has been reached.

ENOENT: The file path is invalid.

ENOMEM: The system doesn't have enough RAM to begin swapping.

EPERM: The CAP SYS ADMIN capability is inaccessible to the caller.

The steps to implement this system call is already mentioned in Section 4.