# System-call tracing with strace


Veda Solutions

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

System calls are kernel interfaces through which applications enter kernel mode to run kernel services. Since system calls are in kernel address space applications cannot directly call or step into a system call routine. To facilitate applications invoke system calls Glibc provides applications special functions called API's(Application programming interfaces). API's can be understood as user mode functions that contain assembly instructions to invoke system calls. API functions are made available to application programmers through shared/ static libraries. Gnu Glibc distribution includes various api routines that applications can link.

read (2), write(2), open(2), Ioctl(2), mmap(2), fork(2) are few examples of api's. API's invoke system calls using a special interrupt with the number of system call and its parameters stored in CPU registers. Since an Api is bound to kernel specific system calls, an application program code calling API will end up being operating system specific, and is not portable. Programmers use varieties to techniques to achieve code portability across operating systems, one of the widely used method is using standard library functions provided by the compiler distributions like GCC. Standard C / C++ libraries provide ready to link functions to perform operations like file I/O, dynamic memory allocations and so on. Applications can be written to use these functions instead of API's to achieve code portability.

Strace is an open source tool that can trace an application execution and show up list of API's being invoked at runtime. This information helps us discover which API's are being invoked with what arguments and return status of the API. Since API's invoke system calls we can conclude on what system calls succeeded and which ones failed.

Strace is very handy to find bugs that may cause exceptions during system call execution, or detect signal events that are delivered to the app.

System administrators, diagnosticians and trouble-shooters will find it invaluable for solving problems with programs for which the source is not readily available since they do not need to be recompiled in order to trace them. Students, hackers and the overly-curious will find that a great deal can be learned about a system and its system calls by tracing even ordinary programs, And programmers will find that since system calls and signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions.

Let's trace few sample applications to understand how to use strace

```
root@techveda:~# vim test.c
/* strace test code
 * Author: techveda.org
 */

# include<stdio.h>

int main()
```

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

```
{
	int a;
	a++;
	return 0;
}
root@techveda:~# gcc test.c -o test
root@techveda:~# strace  ./test
execve("./test", ["./test"], [/* 41 vars */]) = 0
brk(0)                                  = 0x906e000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7726000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63579, ...}) = 0
mmap2(NULL, 63579, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7716000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220o\1\0004\0\0\0"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442372, ...}) = 0
mmap2(NULL, 1448456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb75b4000
mmap2(0xb7710000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15c) = 0xb7710000
mmap2(0xb7713000, 10760, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7713000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb75b3000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb75b38d0, limit:1048575, seg_32bit:1,
contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7710000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb7745000, 4096, PROT_READ)   = 0
munmap(0xb7716000, 63579)               = 0
exit_group(0)                           = ?
```

Above trace dump shows API's invoked during application initialization and system calls to set up process address space components and load time libraries, arguments being passed to each apis and their return status are also being shown up.  System calls invoked by applications code would be shown up after all startup calls, since our test application did not invoke any API's above dump ends with description of startup calls.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

Let's modify our application to include a c standard library function call

```
/* strace test code
 * Author: techveda.org
 */

# include <stdio.h>
int main()
{
	printf("hello strace\n");
	return 0;
}
root@techveda:~# gcc test.c  -o test
root@techveda:~# strace ./test
execve("./test", ["./test"], [/* 41 vars */]) = 0
brk(0)                                  = 0x9e9f000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7742000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63579, ...}) = 0
mmap2(NULL, 63579, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7732000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220o\1\0004\0\0\0"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442372, ...}) = 0
mmap2(NULL, 1448456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb75d0000
mmap2(0xb772c000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15c) = 0xb772c000
mmap2(0xb772f000, 10760, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb772f000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb75cf000
set_thread_area({entry_number:-1 -&amp;gt; 6, base_addr:0xb75cf8d0, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb772c000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb7761000, 4096, PROT_READ)   = 0
munmap(0xb7732000, 63579)               = 0
```

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

```
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7741000
write(1, "hello strace\n", 13hello strace
)       = 13
exit_group(0)                   = ?
```

Printf function has invoked write API to direct string constant to terminal device, we can see write being called with stdout as first argument.

```
/* Strace test code
 * Author: techveda.org
 */

# include <stdio.h>
# include <malloc.h>
# include <string.h>
# include <stdlib.h>

int main()
{
      void *ptr;
      printf("hello strace\n");
      ptr = (void *)malloc( 2048);
      if( ptr == NULL){
            perror("malloc");
            exit(1);
      }
      memset( ptr, 0, 2048);
      free( ptr);
      return 0;
}
root@techveda:~# gcc test2.c -o test2
root@techveda:~# strace ./test2
execve("./test2", ["./test2"], [/* 41 vars */]) = 0
brk(0)                          = 0x89c2000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb777e000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63579, ...}) = 0
```

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

```
mmap2(NULL, 63579, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb776e000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220o\1\0004\0\0\0"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442372, ...}) = 0
mmap2(NULL, 1448456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb760c000
mmap2(0xb7768000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15c) = 0xb7768000
mmap2(0xb776b000, 10760, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb776b000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb760b000
set_thread_area({entry_number:-1 -&gt; 6, base_addr:0xb760b8d0, limit:1048575, seg_32bit:1,
contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7768000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb779d000, 4096, PROT_READ)   = 0
munmap(0xb776e000, 63579)               = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb777d000
write(1, "hello strace\n", 13hello strace
)       = 13
brk(0)                                  = 0x89c2000
brk(0x89e3000)                          = 0x89e3000
exit_group(0)                           = ?
```

Dynamic memory allocation request of malloc was processed by brk system call; brk alters the program break point. Increasing the program break has the effect of allocating memory to the process; decreasing the break deallocates memory. The first brk call with 0 argument returns current program break point and value read is 0x89c2000, second brk call is being invoked with new break point as an argument, the value **0x89e3000** indicates new break point, and return value of this call confirms that it succeeded in altering program break point to specified location.

Also upon close observation of the dump we can find that applications heap allocation request was for only 2048(2k) bytes, but the dump shows around 135k block is allocated **(0x89e3000 - 0x89c2000 = 0x21000[135168 bytes]).** This is an optimization of glibc implementation of malloc. Further details show Applications *free* call to release memory did not alter program

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

break point, so we could conclude that freeing dynamic memory allocated using malloc doesn't release allocated heap back to system.

```
/* Strace test code
 * Author: techveda.org
 */

#include<stdio.h>
#include<string.h>
int main()
{
	char ch[100];
	FILE *fp;
	fp = fopen("gnu","r");
	fread(&ch,100,1,fp);
	fclose(fp);
}
root@techveda:~# gcc test.c -o test
root@techveda:~# strace  ./test
execve("./test", ["./test"], [/* 41 vars */]) = 0
brk(0)                                  = 0x831a000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb76ed000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63579, ...}) = 0
mmap2(NULL, 63579, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb76dd000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220o\1\0004\0\0\0"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442372, ...}) = 0
mmap2(NULL, 1448456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb757b000
mmap2(0xb76d7000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15c) = 0xb76d7000
mmap2(0xb76da000, 10760, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb76da000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb757a000
```

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

```
set_thread_area({entry_number:-1 -&amp;gt; 6, base_addr:0xb757a8d0, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb76d7000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb770c000, 4096, PROT_READ)   = 0
munmap(0xb76dd000, 63579)            = 0
brk(0)                               = 0x831a000
brk(0x833b000)                       = 0x833b000
open("gnu", O_RDONLY)                = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=44541, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb76ec000
read(3, "\n\nThe GNU Manifesto\n\nCopyright ("..., 4096) = 4096
close(3)                             = 0
munmap(0xb76ec000, 4096)             = 0
exit_group(0)                        = ?
```

*fopen* used to open file has used *open* api to intitate file open, *open* returned with file descriptor no 3. Our next operation in the program was *fread* to read 100 bytes of file data. Dump shows clearly how the operation was carried out. First *fstat64* call was use to get file stats including its total size, then 4k (4096) buffer block is allocated using anonymous memory map. *mmap* returned start address of the region (**0xb76ec000),** subsequently read system call was initiated to read 4096 bytes of file data from file descriptor 3 into memory mapped buffer. This again clearly shows optimization of file read operations, tough applications requested to read 100 bytes, full page was read. Such optimization may consume additional memory , but minimizes system calls, since 4096 data is pre-fetched into application memory map, subsequent *fread* call does not require system call(if req is less than 4096 bytes).

We will insert one more *fread* operation into source application to confirm this optimization

```
#include<stdio.h>
#include<string.h>
int main()
{
      char ch[100];
      FILE *fp;
      fp = fopen("gnu","r");
      fread(&ch,100,1,fp);
      fread(&ch,100,1,fp);
      fclose(fp);
}
root@techveda:~# gcc test.c -o test
root@techveda:~# strace  ./test
execve("./test", ["./test"], [/* 41 vars */]) = 0
```

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

```
brk(0)                          = 0x9474000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb77ab000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=63579, ...}) = 0
mmap2(NULL, 63579, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb779b000
close(3)                        = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220o\1\0004\0\0\0"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1442372, ...}) = 0
mmap2(NULL, 1448456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb7639000
mmap2(0xb7795000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x15c) = 0xb7795000
mmap2(0xb7798000, 10760, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7798000
close(3)                        = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7638000
set_thread_area({entry_number:-1 -&gt; 6, base_addr:0xb76388d0, limit:1048575, seg_32bit:1,
contents:0, read_exec_only:0, limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7795000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb77ca000, 4096, PROT_READ)   = 0
munmap(0xb779b000, 63579)           = 0
brk(0)                          = 0x9474000
brk(0x9495000)                  = 0x9495000
open("gnu", O_RDONLY)               = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=44541, ...}) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb77aa000
read(3, "\n\nThe GNU Manifesto\n\nCopyright ("..., 4096) = 4096
close(3)                        = 0
munmap(0xb77aa000, 4096)            = 0
exit_group(0)                   = ?
```

Above log confirms that each call to *fread*, need not necessarily invoke system calls and shows how library routines are written to perform various optimizations. Call *fclose* resulted in *close* system call and then anonymous map used to hold file data was *unmapped* since application has closed file descriptor.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License

While running strace on large executables it may generate huge trace, we can instruct strace to redirect trace data into a file like this

#strace -o outfile ./a.out

To attach and trace a running program use

#strace –p <pid>

If you use the **-t** option, then **strace** will prefix each line of the trace with the time of day. We can even specify the system call functions to trace using the **-e** option. For example, to trace only open() and close() function system calls use the following command

#strace -o outfile  -e trace=open,close ./a.out

Veda Solutions

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License