# Lab 4 & 5: System Calls

## Introduction

A system call is just what its name implies—a request for the operating system to do something on behalf of the user's program

- Syscall()
- Sendfile()
- Free()
- Stat()
- Access()
- Chmod()/Fchmod

## Objective

- To be familiar with system calls.
- To be able to use system calls.
- To be able to manipulate data using system calls.
- To be able to access.
- To be able to change mode.

## Concept Map

A system call is just what its name implies—a request for the operating system to do something on behalf of the user's program

## Syscall()

This system call has a collection of system related information. Example of this system call is as following.

```
syscall(SYS_call, arg1, arg2, …);
```

Here in the first argument we are providing the syscall which we specifically want to use and the rest of the arguments can be anything which that syscall requires.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>

int main() {

unsigned cpu, node;
// Get current CPU core and NUMA node via system call
// Note this has no glibc wrapper so we must call it directly
syscall (SYS_getcpu, &cpu, &node, NULL);

    //    Display information f("This program is running on CPU
    print    core %u and NUMA node
%u.\n\n",   cpu, node);
    return   0;
    }
```

## Sendfile()

The sendfile system call provides an efficient mechanism for copying data from one file descriptor to another. The file descriptors may be open to disk files, sockets, or other devices. Typically, to copy from one file descriptor to another, a program allocates a fixed-size buffer, copies some data from one descriptor into the buffer, writes the buffer out to the other descriptor, and repeats until all the data has been copied. This is inefficient in both time and space because it requires additional memory for the buffer and performs an extra copy of the data into that buffer.

## Free()

The **free**() function frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc**(), **calloc**() or **realloc**(). Otherwise, or if *free(ptr)* has already been called before, undefined behavior occurs. If *ptr* is NULL, no operation is performed.

## Stat()

Stat system call is a system call in Linux to check the status of a file such as to check when the file was accessed. The stat() system call actually returns file attributes. The file attributes of an inode are basically returned by Stat() function. An inode contains the metadata of the file.

```
#include<stdio.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>

void sfile(char const filename[]);

int main(){   ssize_t
read;   char* buffer =
0;   size_t buf_size =
0;

  printf("Enter the name of a file to check: \n");
read = getline(&buffer, &buf_size, stdin);

  if (read <=0 ){
    printf("getline failed\n");
exit(1);
  }
  if (buffer[read-1] == '\n'){     buffer[read-
1] = 0;
  }

  int s=open(buffer,O_RDONLY); if(s==-1){

      printf("File doesn't exist\n"); exit(1);

        {
  } el

    sfile      (buffer);
```

```
  }
    freebuffer     );
   retur       0;

}
  id            (  ar const filename[]){

  struct            sfile;

  if(           (   ename,&sfile)==-1){
                f(  rror Occurred\n");
  print

                     data members of stat struct
  print        ("\n ile st_uid %d \n",sfile.st_uid);
  print        ("\n ile st_blksize %ld \n",sfile.st_blksize);
  print        ("\n ile st_gid %d \n",sfile.st_gid);
  print        ("\n ile st_blocks %ld \n",sfile.st_blocks);
  print        ("\n ile st_size %ld \n",sfile.st_size);
  print        ("\n ile st_nlink %u \n",(unsigned int)sfile.st_nlink);
  print        ("\n          File           Permissions User\n");
  print        ((   sfile.       S_IRUSRst_mode &     )?
  print        ((   sfile.       S_IWUSR"r":"-");
  print        ((   sfile.       S_IXUSRst_mode &     )?
  print        ("\n    "w":"-"); st_mode & )? "x":"-");
  print        "     );
  print        ("\n          File           Permissions Group\n");
  print        ((   sfile.       S_IRGRPst_mode &     )?
  print        ((   sfile.       S_IWGRP"r":"-");
  print        ((   sfile.       S_IXGRPst_mode &     )?
  print        ("\n    "w":"-"); st_mode & )? "x":"-");
  print        "     );
  print        ("\n          File           Permissions Other\n");
  print        ((   sfile.       S_IROTHst_mode &     )?
  print        ((   sfile.       S_IWOTH"r":"-");
               ((   sfile.       S_IXOTHst_mode &     )?
       ile     ("\n    "w":"-"); st_mode & )? "x":"-");
                     );


       at



       essing
       f f f
       f f f
       f f f
}      f f f
```

## Access()

*Access()* command is used to check whether the calling program has access to a specified file. It can be used to check whether a file exists or not. The check is done using the calling process's real UID and GID.

```
int access(const char *pathname, int mode);
```

Here, the first argument takes the path to the *directory/file* and the second argument takes flags *R_OK, W_OK, X_OK or F_OK*.

- **F_OK flag** : Used to check for existence of file.

- **R_OK flag** : Used to check for read permission bit.
- **W_OK flag** : Used to check for write permission bit.
- **X_OK flag** : Used to check for execute permission bit.

*Note:* If access() cannot access the file, it will return -1 or else it will be 0.

```
#include<errno.h>
#include<sys/types.h>
#include<sys/stat.h>
```

```
#include<fcntl.h> extern int errno;  int main(int
argc, const char *argv[]){           int fd =
access("sample.txt", F_OK);        if(fd == -1){
printf("Error    Number    :    %d\n",    errno);
perror("Error Description:");
    }                               else
printf("No error\n");      return 0;
}
```

## Chmod() / Fchmod()

The chmod() and fchmod() system calls change a files mode bits. (The file mode consists of the file permission bits plus the set-user-ID, set-group-ID, and sticky bits.) These system calls differ only in how the file is specified:

* chmod() changes the mode of the file specified whose pathname is given in pathname, which is dereferenced if it is a symbolic link.

* fchmod() changes the mode of the file referred to by the open file descriptor fd.

### Procedure& Tools

In this section, you will study how to setup and VMware.
Tools

- Download and install Virtual Box
- Download and install Ubuntu

### Walkthrough Task

This section will provide a practice task which you need to finish during the lab. You need to finish the tasks in the required time.

### Task:

Then make a new c type file to code, to do so type the following command:
**gedit code.c**
Now type all the following code into that file.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/file.h>
#include <sys/sendfile.h>
#include <sys/random.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/fcntl.h>

#define BUFFER_SIZE 67108864

int main() {

int fOut, fIn;

printf("\nI/O test with sendfile() and related system calls.\n\n");
// Grab a BUFFER_SIZE buffer.
     // The buffer will have random data in it but we don't care about
     that.
     printf("Allocating 64 MB buffer:                        ");
     char *buffer = (char *) malloc(BUFFER_SIZE);
     printf("DONE\n");

    // Write the buffer to fOut
    printf("Writing data to first buffer:                ");
fOut = open("buffer1", O_RDONLY);     write(fOut, &buffer,
BUFFER_SIZE);     close(fOut);     printf("DONE\n");

    printf("Copying data from first file to second:      ");
fIn = open("buffer1", O_RDONLY);     fOut = open("buffer2",
O_RDONLY);     sendfile(fOut, fIn, 0, BUFFER_SIZE);
close(fIn);     close(fOut);     printf("DONE\n");
      printf("Freeing
     buffer:"); free(buffer);
     printf("DONE\n");

         printf("Deleting
files: ");
unlink("buffer1");
unlink("buffer2");
printf("DONE\n");     return 0;
}
```

To run this .c file, you have to first install GCC library on your system. To install gcc, give the following command on terminal: **$ sudo apt-get install build-essential**

Give the following command on terminal: **gcc code.c -o code**
**./code**

### Practice Tasks

This section will provide more practice exercises which you need to finish during the lab. You need to finish the tasks in the required time. When you finish them, put these tasks in your GitHub Account.

Practice Task 1 [Expected time = 15mins]

Write a .c code to create a text file with name “Lab04_ONE” name. Now write the following sentence in it “We are writing data inside a file with Write command”. Create another file with “LAB04_T” name and now use sendfile command to write the exact same content in the new file and now save this file and display the content of LAB04_TWO on console screen.

Practice Task 2 [Expected time = 15mins]

Write a code to display the stats of file LAB04_TWO created in task 1 by using system call.

## Evaluation Task (Unseen) [Expected time = 30mins]

The lab instructor will give you unseen task depending upon the progress of the class.

## Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 3: Evaluation of the Lab

| Sr. No. | Task No | Description | Marks |
|---|---|---|---|
| 1 | 1 | Problem Modeling | 20 |
| 2 | 2 | Procedures and Tools | 10 |
| 3 | 3 | Practice tasks and Testing | 35 |
| 4 | 4 | Evaluation Tasks (Unseen) | 20 |
| 5 | | Comments | 5 |
| 6 | | Good Programming Practices | 10 |

## Further Reading

This section provides the references to further polish your skills.

Slides

The slides and reading material have already been shared via WhatsApp group.