Low-Level File I/O Operations

System calls are essential components of the operating system Linux because they enable user-space programs to perform privileged activities in a controlled and safe manner. File management system calls are used to open, read, write, and close documents, as well as to create, rename, and delete them.

- open(): It is the system call to open a file to perform operations such as read and write
- read(): This system call opens the file in reading mode. We cannot edit the files with this system call. Multiple processes can execute the read() system call on the same file simultaneously.
- write(): It is an internal call that is used to write information stored in a buffer of memory. We cannot edit the files with this system call. Multiple processes can execute the read() system call on the same file simultaneously.
- close(): It is a system call for closing a file determined by a given file descriptor.
- creat(): It creates a new empty file on the disk. It returns the file descriptor assigned to the file on success, whereas it returns -1 upon failure.
- lseek(): It is a system call that repositions the read/write file offset, i.e., it changes the positions of the read/write pointer within the file.

System Call Usage:

- int creat(char *filename, mode_t mode): filename is used to specify the name of the file which you want to create and mode indicates permissions of a new file. It returns the file descriptor used, whereas it returns -1 upon failure.
- int open (const char* Path, int flags [, int mode]): path to the file which you want to use. Flags can take the following values: O_RDONLY: read-only, O_WRONLY: write only, O_RDWR: read and write, O_CREAT: create the file if it does not exist, and O_EXCL: prevent creation if it already exists. It returns the file descriptor used, whereas it returns -1 upon failure.
- int close(int fd): fd represents the file descriptor of the file that we want to close. It returns 0 on success and -1 on failure
- size_t read (int fd, void* buf, size_t cnt): fd is the file descriptor, buf is the buffer to read the data from, and cnt is the requested number of bytes to read. It returns the number of bytes read on success, returns 0 on reaching the end of the file, and returns -1 on error.
- size_t write (int fd, void* buf, size_t cnt): fd is the file descriptor, buf is the buffer to write the data to, and cnt is the requested number of bytes to write. It returns the number of bytes written successfully, returns 0 on reaching the end of the file, and returns -1 on error.

Off_t lseek(int fd, off_t offset, int whence): The first parameter “fd” is a file descriptor. The second is “offset,” which is used to position the pointer. The third parameter, “whence,” is used to specify the position of a file pointer, e.g., beginning, end, and mid.
lseek(fd,5,SEEK_SET): This moves the pointer 5 positions ahead starting from the beginning of the file.

lseek(fd,5,SEEK_CUR): This moves the pointer 5 positions ahead from the current position in the file.
lseek(fd,-5,SEEK_CUR): This moves the pointer 5 positions back from the current position in the file.

- lseek(fd,-5,SEEK_END): This moves the pointer 5 positions back from the end of the file.

Example: C program to show open system call.

```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
  // if the file does not exist in the current directory, then file demo.txt is created.
  int fd = open("demo.txt", O_RDONLY | O_CREAT);
  printf("fd = %d/n", fd);
  if (fd ==-1)
  {
     // print which type of error has occurred
     printf("Error Number % d\n", errno);
     // print program detail"Success or failure"
     perror("Program");
  }
  return 0;
}
```

Sample Output:

fd = 3

Example: Demonstrate the close system call.

```
#include<stdio.h>
#include<fcntl.h>
int main()
{
  // assuming that demo.txt is already created
  int fd1 = open("demo.txt", O_RDONLY, 0);  //open file in read-only mode
  close(fd1); //close system call just takes the file descriptor as parameter
  // assuming test.txt is already created
  int fd2 = open("baz.txt", O_RDONLY, 0);  //open file in read-only mode
  printf("fd2 = % d\n", fd2);
  exit(0);
}
```

Output:

fd2 = 3

Example: C program to illustrate the read system call.

```
#include<stdio.h>
#include <fcntl.h>
```

```
int main()
{
 int fd, sz;
 char *c = (char *) calloc(100, sizeof(char));
 fd = open("demo.txt", O_RDONLY); //open file in read-only mode
 if (fd < 0) { perror("r1"); exit(1); } //check for error in opening
 sz = read(fd, c, 10); //read 10 bytes from the opened file
 printf("called read(% d, c, 10).  returned that"
        " %d bytes were read.\n", fd, sz);
 c[sz] = '\0';
 printf("Those bytes are as follows: % s\n", c);
}
```

Output:

called read(3, c, 10). returned that 10 bytes were read.
Those bytes are as follows: 0 0 0 demo.

Example: C program to illustrate the write system call.

```
#include<stdio.h>
#include <fcntl.h>
main()
{
 int sz;
 int fd = open("demo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644); //open demo.txt
 //in write mode and create the file if it doesn't exist, O_TRUNC will truncate
//the file and 0644 represents permissions
 if (fd < 0) //check for errors
 {
    perror("r1");
    exit(1);
 }
 sz = write(fd, "hello friends\n", strlen("hello friends\n"));
  printf("called write(% d, \"hello friends\\n\", %d)."
    " It returned %d\n", fd, strlen("hello friends\n"), sz);
  close(fd);
}
```

Output:

called write(3, "hello friends\n", 14). It returned 13

Example: C program to illustrate open, write, and close I/O system calls.

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<fcntl.h>
 int main (void)
{
   int fd[2];
```

```
  char buf1[12] ="hello world";
  char buf2[12];
   // assume demo.txt is already created
  fd[0] = open("demo.txt", O_RDWR);
  fd[1] = open("demo.txt", O_RDWR);
  write(fd[0], buf1, strlen(buf1));
  write(1, buf2, read(fd[1], buf2, 12));
  close(fd[0]);
  close(fd[1]);
  return 0;
}
```

Output:

hello world

Example: C program to illustrate create system call.

```
#include <stdio.h>
#include <sys/types.h>       /* defines types used by sys/stat.h */
#include <sys/stat.h>        /* defines S_IREAD & S_IWRITE       */
int main()
   {
     int fd;
          //create data.txt, S_IREAD is used for read permission bit for the owner
    //of the file, S_IWRITE is used for write permission bit for the owner of
    //the file.
     fd = creat("data.txt", S_IREAD | S_IWRITE);
     if (fd == -1)//check for error in creation
       printf("Error in opening data.txt\n");
     else
       {
       printf("data.txt opened for read/write access\n");
       printf("data.txt is currently empty\n");
       }
     close(fd);//close the file
     exit (0);
   }
```

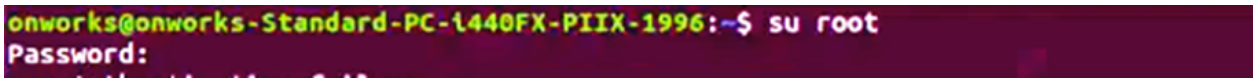
Example: C program to illustrate lseek system call.

```
#include <stdio.h>
#include <fcntl.h>
int main()
{
   int fd;
   long position;
   //open data.txt file in read-only mode
   fd = open("data.txt", O_RDONLY);
   if ( fd != -1)
```

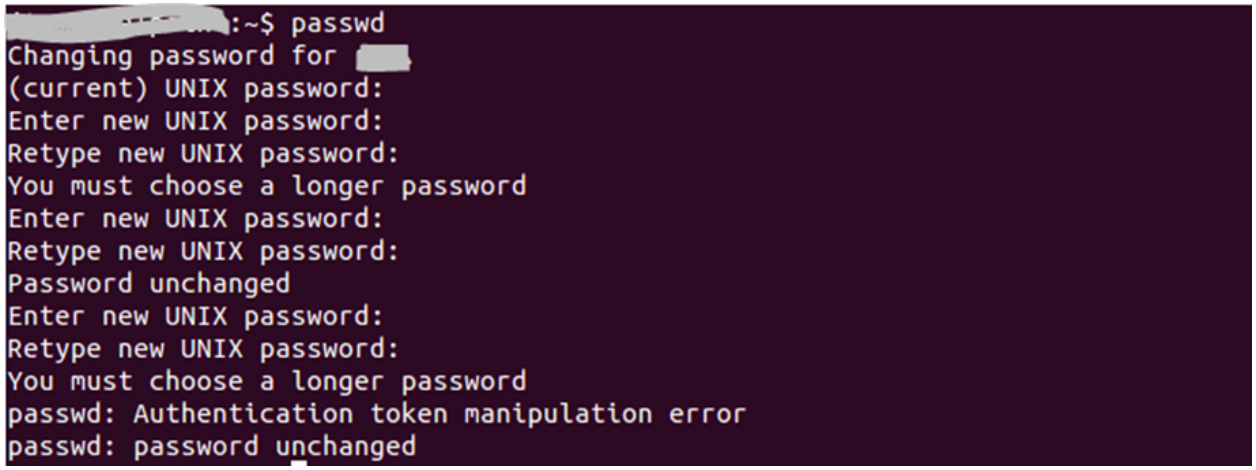
```
    {
      position = lseek(fd, 0L, 2);  /* seek 0 bytes from end-of-file */
      if (position != -1)
           printf("The length of data.txt is %ld bytes.\n", position);
      else
           perror("lseek error");
        }
   else
     printf("can't open data.txt\n");
     close(fd);
}
```

System Administration: Advance Commands

su (also known as a substitute or switch user): you can accomplish commands with the privileges of another user by default root. During a current login session, su command is the easiest way to switch to the administrative account.
When a user runs the su command, they are prompted for the password of the user account they are trying to switch to. Once the password is entered, the user is logged in as the target user and can execute commands with that user's privileges.
Syntax: su [OPTIONS] [USER [ARGUMENT...]]
Example: Switching to root user.



The **passwd** command changes passwords for user accounts. A normal user may only change the password for their own account, whereas the superuser can change the password for any account.
Syntax: passwd [options] [username]
Example: Changing password for user1.



The **adduser** command in Linux creates a new user or group. By applying various options, adduser allows customizing settings in the user provisioning process. The command is a high-level interface for useradd and features interactive prompts when creating new accounts.
Syntax: sudo adduser <options> <username>

```
[redacted]:~$ sudo adduser test
Adding user `test' ...
Adding new group `test' (1001) ...
Adding new user `test' (1001) with group `test' ...
The home directory `/home/test' already exists.  Not copying from `/etc/skel'.
adduser: Warning: The home directory `/home/test' does not belong to the user you are
currently creating.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for test
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
```

userdel command in the Linux system is used to delete a user account and related files. This command basically modifies the system account files, deleting all the entries that refer to the given username.

Syntax: userdel [options] [username]

```
[redacted]:~$ ls /home/
algoscale  kafka  neuser  newuser2  rahul
[redacted]:~$ sudo userdel -r newuser2
userdel: newuser2 mail spool (/var/mail/newuser2) not found
[redacted]:~$ ls /home/
algoscale  kafka  neuser  rahul
```

groupadd command is used to create a new group. Using groups, we can group together a number of users and set privileges and permissions for the entire group.

Syntax: groupadd [option] group_name

```
[redacted]:~$ sudo groupadd student
[redacted]:~$
[redacted]:~$ grep student /etc/group
student:x:1004:
[redacted]:~$
```

groupdel command is used to delete an existing group. It will delete all entries that refer to the group and modify the system account files. It is handled by superuser or root user.

Syntax: groupdel [options] [GROUP]

```
[redacted]:~$ sudo groupdel -f group1
[sudo] password for kash:
groupdel: group 'group1' does not exist
[redacted]:~$
```