LAB_01_OS_Assignment

- System calls :- In computing, a 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.

- Services Provided by System Calls :
  - Process creation and management
  - Main memory management
  - File Access, Directory and File system management
  - Device handling(I/O)
  - Protection
  - Networking, etc.

- Types of System Calls : There are 5 different categories of system calls –
  - Process control: end, abort, create, terminate, allocate and free memory.
  - File management: create, open, close, delete, read file etc.
  - Device management
  - Information maintenance
  - Communication

- File Descriptor :
  Read from stdin => read from fd 0 : Whenever we write any character from keyboard, it read from stdin through fd 0 and save to file named /dev/tty.
  Write to stdout => write to fd 1 : Whenever we see any output to the video screen, it's from the file named /dev/tty and written to stdout in screen through fd 1.
  Write to stderr => write to fd 2 : We see any error to the video screen, it is also from that file write to stderr in screen through fd 2.

1. Read :
   From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.
   Syntex:
   size_t read (int fd, void* buff, size_t cnt);

   Returns: How many bytes were actually read
   - return Number of bytes read on success
   - return 0 on reaching end of file
   - return -1 on error
   - return -1 on signal interrupt

   Important points
   - buf needs to point to a valid memory location with length not smaller than the specified size because of overflow.

- fd should be a valid file descriptor returned from open() to perform read operation because if fd is NULL then read should generate error.
- cnt is the requested number of bytes read, while the return value is the actual number of bytes read. Also, some times read system call should read less bytes than cnt.

2. Open:
   Used to Open the file for reading, writing or both.
   Syntex:
   #include<sys/types.h>
   #include<sys/stat.h>
   #inlcude<fcntl.h>
   Open(const char* path, int flags, [, int mode]);
   - Path : path to file which you want to use
     - use absolute path begin with "/", when you are not work in same directory of file.
     - Use relative path which is only file name with extension, when you are work in same directory of file.
   - flags : How you like to use
     - O_RDONLY: read only, O_WRONLY: write only, O_RDWR: read and write, O_CREAT: create file if it doesn't exist, O_EXCL: prevent creation if it already exists

3. Write:
   Writes cnt bytes from buf to the file or socket associated with fd. cnt should not be greater than INT_MAX (defined in the limits.h header file). If cnt is zero, write() simply returns 0 without attempting any other action.
   Syntex :
   #include<fcntl.h>
   Size_t write (int fd, void * buff, size_t cnt);

   Returns: How many bytes were actually written
   - return Number of bytes written on success
   - return 0 on reaching end of file
   - return -1 on error
   - return -1 on signal interrupt

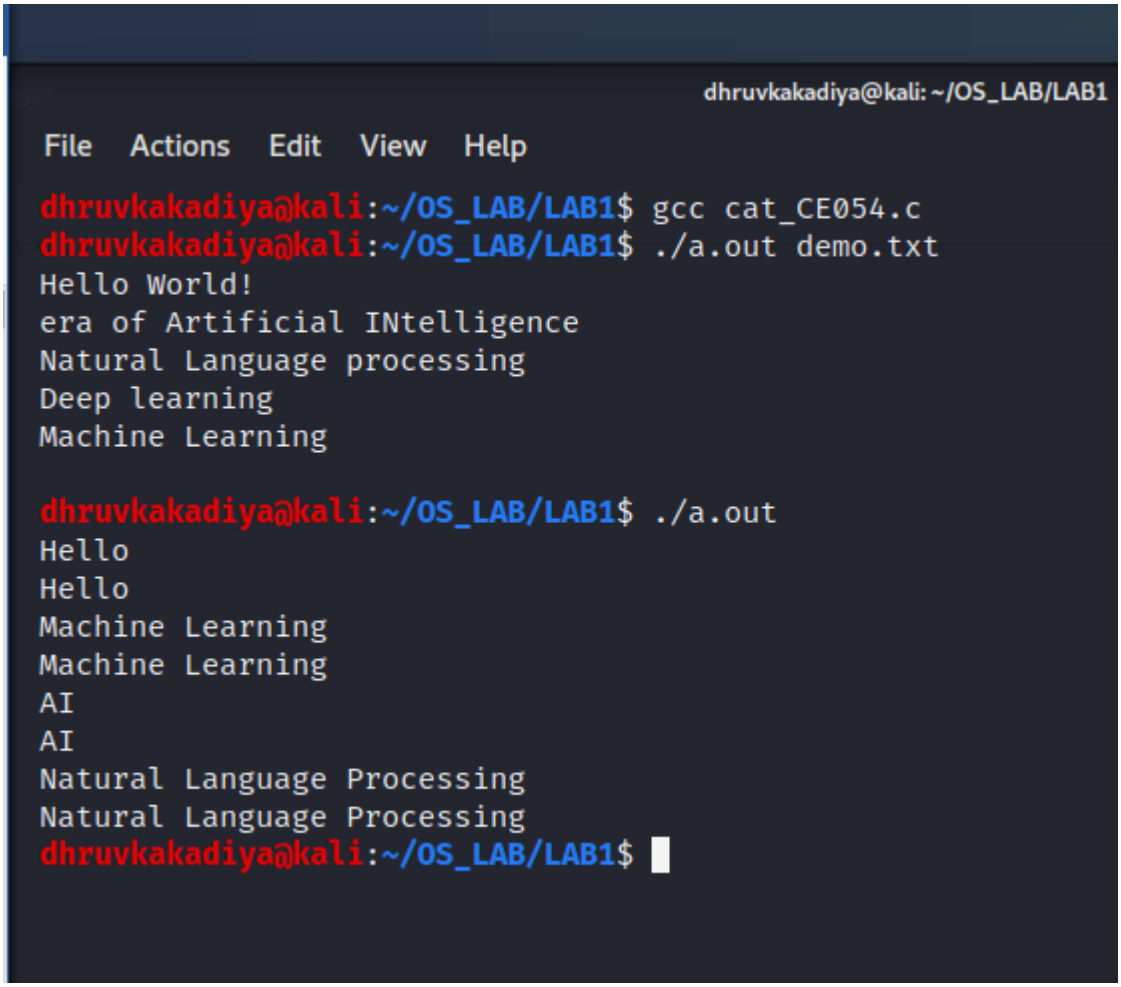
   Important points
   - The file needs to be opened for write operations
   - buf needs to be at least as long as specified by cnt because if buf size less than the cnt then buf will lead to the overflow condition.

4. Close:
   Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.
   Syntex:
   #include<fcntl.h>
   Int close(int fd);
   Return
   0 on success.
   -1 on error.
   How it works in the OS
   Destroy file table entry referenced by element fd of file descriptor table
   As long as no other process is pointing to it!

CODE:

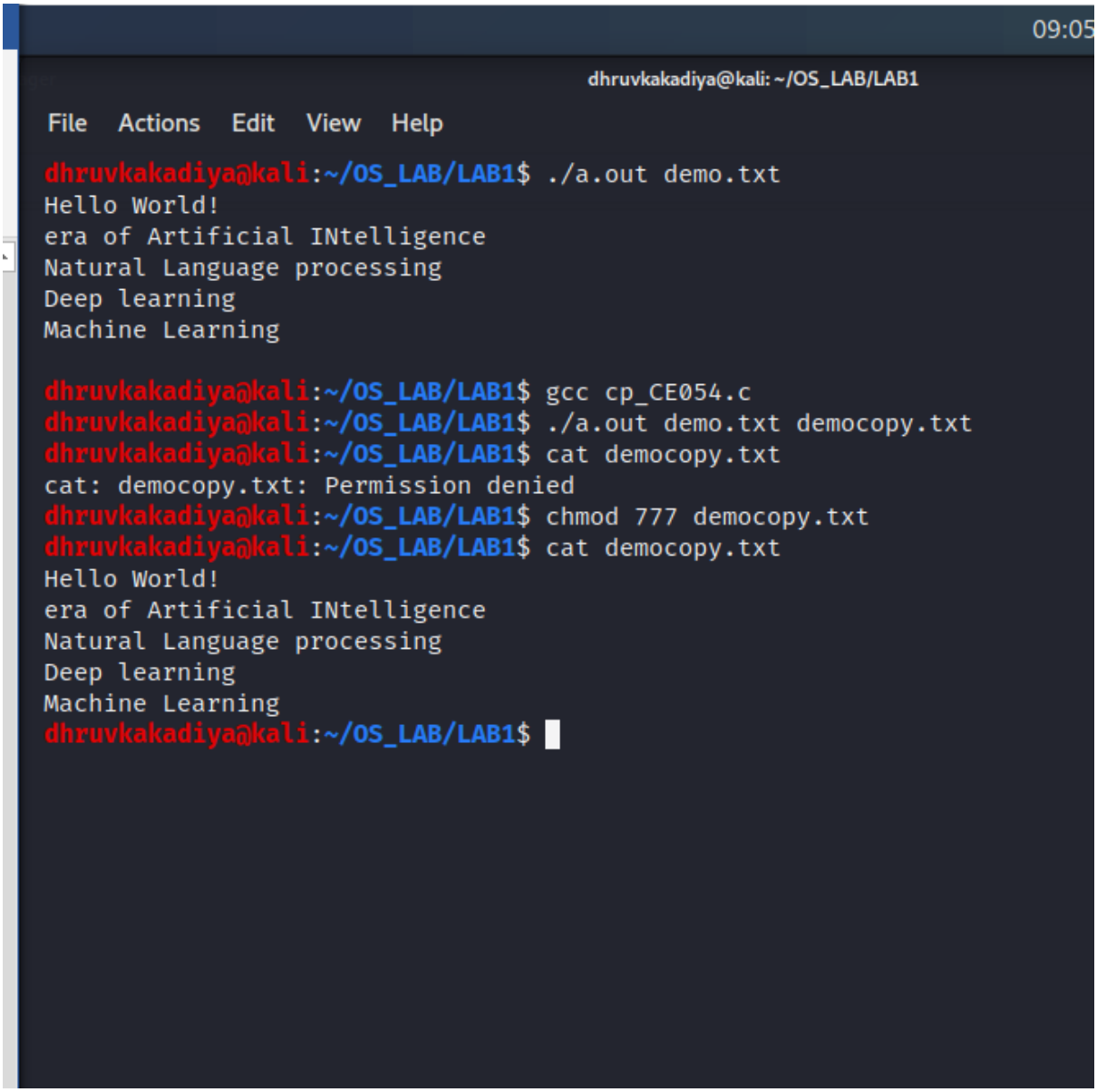
1. Implement basic “cat” command using system calls.

```
// Implementation of CAT command
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
void main(int argc, char *argv[])
{

        char buffer[10000];
        int fd, n;
        // implementation of simple cat command without any cmd line argument

        if(argc == 1)
        {
        while (n = read(0, buffer, sizeof(buffer)))
        {
                        write(1, buffer, n);
        }
        }
        // implementation of cat command with cmd line argument

        if(argc >= 2)
        {
        for(int i = 1 ; i < argc ; i++)
                {
                        fd = open(argv[i], O_RDONLY);
                        n = read(fd, buffer, 10000);
                        write(1, buffer, n);
                        write(1, "\n", 1);
        }
        }
}
```



2. Implement basic “cp” command using system calls.

```
// Implementatio of CP command
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main(int argc,char* argv[])
{
        char buffer[10000];
        // open src file
        int src = open(argv[1], O_RDONLY);
        // open destination file
        int dest = open(argv[2],O_WRONLY | O_CREAT);
        int n = read(src, buffer, sizeof(buffer));
        // write operation on destination file
        write(dest, buffer, n);

        // closing files
        close(src);
        close(dest);
}
```


09:05
dhruvkakadiya@kali: ~/OS_LAB/LAB1
File Actions Edit View Help
dhruvkakadiya@kali:~/OS_LAB/LAB1$ ./a.out demo.txt
Hello World!
era of Artificial INtelligence
Natural Language processing
Deep learning
Machine Learning
dhruvkakadiya@kali:~/OS_LAB/LAB1$ gcc cp_CE054.c
dhruvkakadiya@kali:~/OS_LAB/LAB1$ ./a.out demo.txt democopy.txt
dhruvkakadiya@kali:~/OS_LAB/LAB1$ cat democopy.txt
cat: democopy.txt: Permission denied
dhruvkakadiya@kali:~/OS_LAB/LAB1$ chmod 777 democopy.txt
dhruvkakadiya@kali:~/OS_LAB/LAB1$ cat democopy.txt
Hello World!
era of Artificial INtelligence
Natural Language processing
Deep learning
Machine Learning
dhruvkakadiya@kali:~/OS_LAB/LAB1$