# System Calls for Operating System

## Open :-

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
extern int errno;
int main()
{
    int fd = open("file.txt", O_RDONLY | O_CREAT);
    printf("fd = %d\n", fd);
    if (fd == -1)
    {
        printf("Error Number %d\n", errno);
        perror("Program");
    }
    return 0;
}
```

Expt: 1

4/1/20

# FAMILIARIZATION OF SYSTEM CALLS

## Aim

To familiarize & understand the use & functioning of System calls used for Operating System & network programming in Linux.

## Theory

### System Calls for Operating System

(i) **Create:-** Used to create a new empty file

Syntax in C language:

```
int creat(char *filename, mode_t mode)
```

(ii) **Open:-** Used to open the file for reading, writing or both.

Syntax in C language:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char* Path, int flags [, int mode]);
```

(iii) **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.

Syntax in C language:

```
size_t read(int fd, void* buf, size_t cnt);
```

(iv) **Write:-** Writes cnt bytes from buf to the file or socket associated with fd.

Syntax in C language:

```
#include <fcntl.h>
size_t write(int fd, void* buf, size_t cnt);
```

Read :-

```
#include <stdio.h>
#include <fcntl.h>
int main()
{
    int fd, sz;
    char *c = (char *) calloc (100, sizeof (char));
    fd= open ("file.txt", O_RDONLY);
    if (fd<0) { perror ("r1"); exit (1); }
    sz= read (fd, c, 10);
    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);
}
```

(v) **Close**:- Tells the operating system we are done with a file descriptor & closes the file which is pointed to by fd.

Syntax in C language:

```
#include <fcntl.h>
int close (int fd);
```

(vi) **Sleep**:- Sleep for a specified number of seconds.

Syntax in C language:

```
#include <unistd.h>
unsigned int sleep (unsigned int seconds);
```

(vii) **Exit**:- Terminate the calling process

Syntax in C language:

```
#include <unistd.h>
void _exit (int status);
```

(viii) **Unlink**:- Delete a name & possibly the file it refers to.

Syntax in C language:

```
#include <unistd.h>
int unlink (const char *pathname);
```

(ix) **Kill**:- Send signal to a process

Syntax in C language:

```
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig);
```

(x) **Get Pid**:- Get process identification of calling process

Syntax in C language:

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid (void);
```

Close:-

```
#include <stdio.h>
#include <fcntl.h>
int main()
{
    int fd1= open ("file.txt", O_RDONLY);
    if (fd1 < 0)
    {
        perror ("c1");
        exit(1);
    }
    printf ("opened the fd= %d \n", fd1);
    if (close (fd1) < 0)
    {
        perror ("c1");
        exit(1);
    }
    printf (" closed the fd. \n");
}
```

(xi) Get Ppid:- Returns the process ID of the parent of the calling process.

Syntax in C language:

```
#include<sys/types.h>
#include<unistd.h>
pid_t getppid(void);
```

(xii) Get Uid:- Get user identity

Syntax in C language:

```
#include <unistd.h>
#include <sys/types.h>
uid_t getuid(void);
```

(xiii) Fork:- Create a child process.

Syntax in C language:

```
#include <sys/types.h>
#include<unistd.h>
pid_t fork(void);
```

(xiv) Pipe:- Creates pipe, a unidirectional data channel that can be used for interprocess communication.

Syntax in C language:

```
#include <unistd.h>
int pipe(int pipefd[2]);
```

(xv) Fifo:- First-in-first-out special file, named pipe

Syntax in C language:

```
int mkfifo(const char *pathname, mode_t mode);
```

(xvi) Execl:- Replaces the current process image with a new process image.

Syntax in C language:

```
#include <unistd.h>
int execl(const char *path, const char *arg,
  ...);
```

# System Calls for Network Programming

## Socket:-

Server:-

```
#include <unistd.h>
#include <stdio.h>
#include<sys/socket.h>
#include<stdlib.h>
#include<netinet/in.h>
#include<string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
  int server_fd, new_socket, valread;
  struct sockaddr_in address;
  int opt=1;
  int addrlen=sizeof(address);
  char buffer[1024]={0};
  char *hello="Hello from server";
  if((server_fd=socket(AF_INET,SOCK_STREAM,
      0))==0)
  {
    perror("socket failed");
    exit(EXIT_FAILURE);
  }
  if(setsockopt(server_fd,SOL_SOCKET,SO_REUSEADDR
      |SO_REUSEPORT,&opt,sizeof(opt)))
  {
    perror("setsockopt");
    exit(EXIT_FAILURE);
  }
  address.sin_family=AF_INET;
  address.sin_addr.s_addr=INADDR_ANY;
  address.sin_port=htons(PORT);
  if(bind(server_fd,(struct sockaddr *)&address,
                    sizeof(address))<0)
  {
    perror("bind failed");
    exit(EXIT_FAILURE);
  }
```

# System Calls for Network Programming

(i) **Socket:-** Create an endpoint for communication.

Syntax in C language:

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int domain, int type, int protocol);
```

(ii) **Bind:-** Bind a name to a socket

Syntax in C language:

```
#include <sys/types.h>
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr
        *addr, socklen_t addrlen);
```

(iii) **Listen:-** Listen for connections on a socket

Syntax in C language:

```
#include <sys/types.h>
#include <sys/socket.h>
int listen (int sockfd, int backlog);
```

(iv) **Accept:-** Accept a connection on a socket

Syntax in C language:

```
#include <sys/types.h>
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *addr,
        socklen_t *addrlen);
```

(v) **Connect:-** Initiate a connection on a socket

Syntax in C language:

```
#include <sys/types.h>
#include <sys/socket.h>
int connect (int sockfd, const struct
        sockaddr *addr, socklen_t addrLen);
```

```
if (listen(server_fd, 3) < 0)
{
    perror("listen");
    exit(EXIT_FAILURE);
}
if ((new_socket = accept(server_fd, (struct sockaddr
    *)&address, (socklen_t*)&addrlen)) < 0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}
valread = read(new_socket, buffer, 1024);
printf("%s\n", buffer);
send(new_socket, hello, strlen(hello), 0);
printf("Hello message sent\n");
return 0;
}
```

## Result:

Familiarized & understood the use & functioning of System calls used for Operating System & network programming in Linux.