Computer Science & Engineering Department
I. I. T. Kharagpur

Operating System: CS33007
3rd Year CSE: 5th Semester (*Autumn 2006 - 2007*)
Lecture III (Linux System Calls II)

Goutam Biswas *Date:* 1st-7th August, 2006

1 File Access Calls

1. In Unix like systems I/O devices are viewed as files (abstraction). Both files and devices can be accessed using system calls related to file I/O.

2. Whenever a process is created, three standard files are opened for it. They are **stdin**, **stdout** and **stderr**.

3. Three data structures are of interest for open files of a process on Linux (Unix like system). These are file descriptor table (one per process), file table (global for the whole system), in-core inode[1] table (global table, one entry per file).

4. Open files are accessed in the process using the index of the file descriptor table. The indices for the standard files are 0 (stdin), 1 (stdout) and 2 (stderr).

5. A new entry is made, in the lowest available slot of the file descriptor table, when a file is opened. A new entry is also made in the file table. If there is already an entry in the inode table for the file, its usage count is increased; otherwise a new entry is made and loaded with data from the disk inode of the file.

6. The file descriptor table of the parent process is copied to the child process when it is created.

7. **open()** - opens the file specified by the path and returns the file descriptor, the index of the lowest available slot in the file descriptor table.

```
/***************************************
 *  open() system call
 ***************************************/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

[1]Inode or index node is the object by which a file is identified in a Unix like system. The inode for a file resides on the disk. When a file is opened, it is copied to the memory. The memory copy of the inode has other fields than the disk copy e.g. usage count etc.

```
int main(){
        int fd ;

        printf("stdin: %d\n", STDIN_FILENO) ;
        printf("stdout: %d\n", STDOUT_FILENO) ;
        printf("stderr: %d\n", STDERR_FILENO) ;

        fd = open("./outFile", O_WRONLY | O_CREAT, 0666) ;
        printf("./outFile: %d\n", fd) ;

        return 0 ;
}
```

8. Code with the software interrupt.

```
/*************************************
 *  open() with software interrupt
 *************************************/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
        int fd ;

        printf("stdin: %d\n", STDIN_FILENO) ;
        printf("stdout: %d\n", STDOUT_FILENO) ;
        printf("stderr: %d\n", STDERR_FILENO) ;

        __asm__ __volatile (
                        "movl $3, %%eax\n\t"
                        "int $0x80\n\t"
                        :"=a" (fd)
                        :"b" ("./outFile"), "c" (O_WRONLY|O_CREAT),
                                "d" (0666)
                        )
        // fd = open("./outFile", O_WRONLY | O_CREAT, 0666) ;
        printf("./outFile: %d\n", fd) ;

        return 0 ;
}
```

9. **close()** - closes an open file descriptor. After close the descriptor does not refer to any file and can be reused.

```
/********************************
```

```
 * close()
 *********************************/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main(){
          int fd ;

          close(STDIN_FILENO) ;  // Closes stdin
          perror("close") ;      // System error message
          fd = open("./inFile", O_RDONLY, 0) ; // Opens inFile in
          printf("./inFile: %d\n", fd) ;       // the slot of stdin

          return 0 ;
}
```

10. **read()** - reads the number of bytes specified (maximum) from an open file specified by the file descriptor.

```
/***************************************
 * read() system call
 ***************************************/
#include <stdio.h>
#include <unistd.h>
#define MAX 101

int main(){
         char buff[MAX] ;
         int charRead ;

         charRead = read(STDIN_FILENO, buff, MAX-1) ;
         buff[charRead] = '\0' ;
         printf("Stream read: %s\n", buff) ;
         return 0 ;
}
```

11. Close stdin, open a file, and use scanf to read from the file.

```
/***************************************
 * close() system call
 ***************************************/
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <fcntl.h>
#define MAX 101

int main(){
        char buff[MAX] ;
        int  fd ;

        close(STDIN_FILENO) ;
        perror("close") ;
        fd = open("./inFile", O_RDONLY, 0) ;
        perror("open") ;
        printf("\t\tfile descriptor: %d\n", fd) ;
        scanf("%[^\n]", buff);
        printf("Stream read:\n%s\n", buff) ;
        return 0 ;
}
```

12. When a child process is created, the file descriptor table of the parent is copied to the file descriptor table of the child. Both are pointing to the same entry of the global file table. Both the process see the same offset values in the file.

```
/***************************************************
 * The file table of the parent is copied to the    *
 * child: cc -Wall fileDescCh.c                     *
 * *************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

#define MAX 1000

int main() {
     int chPID, fd ;

     fd = open("./inFile", O_RDONLY, 0) ;
     if((chPID = fork()) != 0) { // Parent
          char buff[MAX] ;

          read(fd, buff, 10) ;
          buff[10] = '\0' ;
          printf("Parent: %s\n", buff) ;
     }
     else { // Child
```

```
        char buff[MAX] ;

        read(fd, buff, 15) ;
        buff[15] = '\0' ;
        printf("Child: %s\n", buff) ;
    }

    return 0 ;
}
```

13. **lseek()** - sets the read/write offset in the open file (specified by the file descriptor).

```
/***********************************
 * lseek() system call
 ***********************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#define MAX 100

int main(){
          int fd, offset ;
          char buff[MAX] ;

          fd = open("./inFile", O_RDONLY, 0) ;
          perror("open") ;

          read(fd, buff, 40) ;
          buff[40] = '\0' ;
          printf("1st line ./inFile: %s\n", buff) ;

          offset = lseek(fd, 10, SEEK_SET) ; // From the beginning
          perror("lseek") ;

          read(fd, buff, 40) ;
          buff[40] = '\0' ;
          printf("After offset: %s\n", buff) ;
          return 0 ;
}
```

14. One can open the same file more than once, simultaneously in a program. There will be two descriptors and two entries in the global file table. The usage count of the in-memory inode table entry will be 2.

```
/***********************************
```

```
 * opening a file more than once
 *********************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#define MAX 100

int main(){
          int fd1, fd2, offset ;
          char buff[MAX] ;

          fd1 = open("./inFile", O_RDONLY, 0) ;
          perror("open") ;
          fd2 = open("./inFile", O_RDONLY, 0) ;
          perror("open") ;
          printf("\t\t\tfd1: %d, fd2: %d\n", fd1, fd2) ;

          read(fd1, buff, 20) ;
          buff[20] = '\0' ;
          printf("\nfd1: 1st read from ./inFile: %s\n", buff) ;

          offset = lseek(fd2, 41, SEEK_CUR) ; // From current pos
          perror("lseek") ;

          read(fd2, buff, 10) ;
          buff[10] = '\0' ;
          printf("\n\t\t\tfd2: 1st read: %s\n", buff) ;

          read(fd1, buff, 10) ;
          buff[10] = '\0' ;
          printf("\nfd1: 2nd read from ./inFile: %s\n", buff) ;
          return 0 ;
}
```

15. The maximum offset that the system call lseek() can produce is $\pm(2^{31} - 1)$. This is insufficient for a disk of size more than 4-GB. A **long long** version of the function **llseek()** is provided.

```
/*******************************************
 * llseek() can produce long long offset
 * Only root can execute this code
 *******************************************/
#include <stdio.h>
#include <unistd.h>
#include <linux/unistd.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define SECT 512

int main(){
        long long unsigned offset = 0x123456789ABCDEF0LLU;
        int fd, i, dat ;
        char buff[SECT] ;

        fd = open("/dev/hda", O_RDONLY, 0) ;
        read(fd, buff, SECT) ;
        for(i=0; i<SECT; ++i) {
                if(i%20 == 0) printf("\n") ;
                dat =  0x000000FF & (int)buff[i] ;
                printf("%x ", dat) ;
        }
        llseek(fd, offset, SEEK_CUR) ;
        printf("\noffset: %llu\n", offset) ;

        return 0 ;
}
```

16. **dup(), dup2()** - duplicates a file descriptor of an open file. Both the descriptors points to the same entry of global file table. Change in offset in one descriptor also reflected in the copied one.

```
/*********************************************
 * dup() system call
 *********************************************/
#include <stdio.h>
#include <unistd.h>
 #include <stdlib.h>
#define MAX 10

int main(){
        int fd ;
        char buff[MAX+1] ;

        fd = dup(STDIN_FILENO) ;
        perror("dup") ;
        printf("New fd for stdin: %d\n", fd) ;

        printf("\t\tEnter an integer: ") ;
        fflush(stdout) ;

        read(fd, buff, MAX) ; // Read from STDIN
```

```
        buff[MAX] = '\0' ;
        printf("\t\tData: %d\n", atoi(buff)) ;
        return 0 ;
}
```

17. File can be used for communication between processes:

```
/**********************************
 * File descriptor (open file)
 * after execve
 * cc -Wall fdExec1.c
 * ./a.out factorial
 **********************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#define MAX 10

int main(int argc, char *argv[], char *envp[]) {
        int n, chPID, status, fd ;
        char buff[MAX], **agv = argv+1;

        printf("Enter a +ve integer: ") ;
        scanf("%d", &n) ;
        sprintf(buff, "%d", n) ;

        fd = open("./fdExec1_IO", O_WRONLY | O_CREAT, 0666) ;
        perror("open W") ;
        write(fd, buff, MAX) ;
        close(fd) ;
        perror("close") ;

        fd = open("./fdExec1_IO", O_RDONLY, 0) ;
        perror("open R") ;

        if((chPID = fork()) != 0) { // Parent
             printf("\t\tInside Parent\n") ;
             close(fd) ;
             perror("close P") ;
             waitpid(chPID, &status, 0) ;
        }
        else execve(agv[0], agv, envp) ;
        return 0 ;
```

```
}
```

The executable required for exec -

```
/**********************************************
 * Factorial: cc -Wall factorial.c -o factorial *
 * *********************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define MAX 10

int main() {
       int n, i, fact = 1 ;
       char buff[MAX] ;

       read(3, buff, MAX) ;
       perror("\t\t\tRead in factorial") ;
       n = atoi(buff) ;
       for(i=1; i<=n; ++i) fact *=i ;
       printf("\t\t\t%d! = %d\n", n, fact) ;

       return 0 ;
}
```

18. File status can be accessed using the system call **fstat()**

```
/********************************************
 * File meta-data: fstat1.c                  *
 * cc -Wall fstat1.c                         *
 * ./a.out <filename>                        *
 * ******************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>

int main(int count, char *vect[]) {
     int fd ;
     struct stat buff ;
     struct tm *ltmP ;
```

```
    if(count < 2) {
         printf("File name not there\n") ;
         exit(0) ;
    }
    fd = open(vect[1], O_RDONLY) ;
    fstat(fd, &buff) ;

    printf("\tDevice: (%d, %d)\n", major(buff.st_dev), minor(buff.st_dev)) ;
    printf("\tInode: %d\n", (int) buff.st_ino) ;
    printf("\tProtection: %o\n", (int) buff.st_mode) ;
    printf("\tHard Links: %d\n", (int) buff.st_nlink) ;
    printf("\tUser ID: %d\n", (int) buff.st_uid) ;
    printf("\tGroup ID: %d\n", (int) buff.st_gid) ;
    printf("\tFile Size: %d\n", (int) buff.st_size) ;
    printf("\tNumber of Blocks: %d\n", (int) buff.st_blocks) ;

    ltmP = localtime(&buff.st_atime) ;
    printf("\tDate & time: %2d/%2d/%4d:%2d:%2d:%2d\n",
                    ltmP -> tm_mday,
                    ltmP -> tm_mon,
                    1900 + ltmP -> tm_year,
                    ltmP -> tm_hour,
                    ltmP -> tm_min,
                    ltmP -> tm_sec) ;

    return 0 ;
}
```

References

[1] http://asm.sourceforge.net//syscall.htmli#3

[2] http://www-128.ibm.com/developerworks/library/l-ia.html#h1