33231

CLASSMATE
Date :
Page :

ASSIGNMENT 9

Title : File Handling System Calls

Problem Statement : Implement an assignment using File Handling System Calls (low level system calls like open, read, write, etc)

Theory :

File Descriptor Table – It is the collection of integer array indices that are file descriptors in which elements are provided to field table entries. One unique field descriptors table is provided in operating system for each process.

Field Table Entry – Field table entries is a structure in-memory surrogate for an open file, which is created when process request to open fails and these entries maintains file position.

Standard File Descriptors – When any process starts, then that process file descriptors table's fd (file descriptor) 0, 1, 2 open automatically (by default) each of these 3 fd references file table entry for a file name dd /dev /tty

/dev /tty : in-memory surrogate for the terminal
terminal : combination keyboard / video screen

There are total 5 types of I/O system calls :

1) **Create :** used to create a new empty file
Syntax – int create (char *filename, mode_t mode)
Parameter – • filename : name of the file you want to create
• mode : indicates permissions of new file
Returns : return first unused file descriptor (generally 3

when first create use in process because 0, 1, 2 fd are reserved)
return -1 when error

How it works in OS

- Create new empty file on disk
- Create file table entry
- Set first unused file descriptor to point to file table entry
- Return file descriptor used -1 upon failure

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

How it works in OS

- Find existing file on disk
- Create file table entry
- set first unused file descriptor to point to file table entry.
- Return file descriptor used, -1 upon failure.

3) close : tells the OS you are done with a file descriptor and close the file which pointed by fd.
Syntax - int close (int fd);
Parameter - fd : file descriptor
Return - 0 on success, 1 on error
How it works in OS :
- Destroy file table entry referenced by element fd of file descriptor table as long as no other process is pointing to it
- Set element fd of file descriptor table to NULL

4) 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
Syntax - size_t read (int fd, void *buf, size_t cnt);
Parameters : • fd - file descriptor
• buf - buffer to read data from
• cnt - length of buffer
Returns : return no of bytes read on success
return 0 on reaching end of file
return -1 on error
return -1 on signal interrupt

5) write : write 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
#include <fcntl.h>
size_t write (int fd, void *buf, size_t cnt);
Parameters : • fd - file descriptor

buf : buffer to write data to
cnt : length of buffer

Returns : return no. of bytes written on success
return 0 on reaching end of file
return -1 on error
return -1 on signal interrupt

Important Points :

1) The file needs to be opened for write operations
2) 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.
3) cnt is the requested number of bytes to write, while the return value is the actual number of bytes written. This happens when fd have a less number of bytes to write than cnt.
4) if write() is interrupted by a signal, the effect is one of the following :
   - if write () has not written any data yet, it returns -1 and sets errno to EINTR
   - if write () has successfully written some data, it returns the number of bytes it ~~works~~ wrote before it was interrupted

Conclusion : I have successfully understood and implemented an assignment using File Handling system calls (Low level system calls like open, read, write, etc)