Operating Systems – I (CS3510)
Assignment-1 Report

Task – 1 : Reverse the lines of a file and output them in an another file.

execve **(const char *_filename_, char *const _argv_[], char *const _envp_[])** – This system call takes an excecutable/binary file as input, which in this case, is the ./a.out executable. There are three arguments passed to this system call, **const char* filename**, **char** const argv**, which is the list of argument strings passed to program and char** const envp, the array of strings passed as environment to the program. The **/* 71 vars */** are the environment variables inherited from the shell. The middle argument is argv[0] (command line arg).

```
execve("./a.out", ["./a.out"], [/* 71 vars */]) = 0
```

brk (NULL) – This system call asks the kernel to let the user read and write to a contiguous chunk of memory which is **the** heap, if this is not done, the program may give segmentation faults. It is invoked after "**munmap**" system call as well, when mappings are removed from the memory.

access **(const char *_pathname_, int _mode_)** – The strace output for this program has several access system calls, since the program involves file I/O. Access checks whether the calling process can access the file pathname, if the pathname is a symbolic link, it is dereferenced. On success, this returns 0, else -1 is returned. The arguments of this system call is pathname and mode for accessibility check, which in this case if **F_OK** (tests for existence of file).

mmap **(void *_addr_, size_t _length_, int _prot_, int _flags_, int _fd_, off_t _offset_)** – The program makes several mmap system calls. This system call maps files into memory. It creates a new mapping in virtual address space of the calling process, with starting address and length of mapping as arguments. The **malloc** and **calloc** library functions usually use it internally. If addr is not NULL, kernel takes it as hint to create mapping on nearby page boundary, prot states desired memory protection of the mapping. Here, addr is NULL when no file is being open using fstream, else not NULL. The length is length of the mapping.

open **(const char *_pathname_, int _flags_)** – Since the program requires opening of file for reading and writing as well, this system call given a pathname for file, returns a non-negative integer as a file-descriptor (fd) for use in subsequent system calls like access and mmap. Fd is set to remain open across an execve system call. **O_RDONLY** and **O_WRONLY** are specified as read and write only file modes.

```
open("input.txt", O_RDONLY)             = 3
open("output.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
```

read **(int _fd_, void *_buf_, size_t _count_)** – This system call reads file from a file descriptor. Once the program starts, the system call has its first argument, fd set to 0 and both "**input.txt**" and "**output.txt**" are opened. The "*buf*" specifies the name, typecasted to **const char *** ;

```
read(0, "input.txt\n", 1024)            = 10
write(1, "Enter output filename : ", 24) = 24
read(0, "output.txt\n", 1024)           = 11
```

The system call returns 0 if operation is successful, and the file position is advanced by 1, as is evident here (10, 11).

Later while reading the contents of input file, the “input.txt\n” is replaced by the content of the file. The third argument is number of bytes read till, as shown below (size = 8191 bytes) :

```
read(3, "Hi,\nHello,\nBye.\n", 8191)    = 16
read(3, "", 8191)                       = 0
```

mprotect – This system call is invoked multiple times throughout the execution of the program, which is done to set protection on a region of memory, which is allocated/mapped by the preceding mmap system call, required for loading file into memory.

fstat (**int** fildes, **struct stat** * buf) – This system call is used to determine information about a file based on its file descriptor.
It is invoked whenever there is any I/O operation (taking file name as input or *read* syscall to read file input) or memory mapping associated to a file. It returns various information that unix filesystem keeps about a file, like size, last modified, permissions, etc.

```
fstat(3, {st_mode=S_IFREG|0644, st_size=141012, ...}) = 0
mmap(NULL, 141012, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f64d7519000
```

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 4), ...}) = 0
write(1, "Enter file name : ", 18)      = 18
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 4), ...}) = 0
read(0, "os2.cpp\n", 1024)              = 8
open("os2.cpp", O_RDWR)                 = 3
```

close (**int** file_descriptor) – This system call is invoked after every **mmap / mmap + mprotect** system call throughout the execution of the program. It closes a file descriptor, such that it no longer refers to the same file and may be reused.

```
mmap(NULL, 141012, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f495a92d000
close(3)                                 = 0
```

arch_prtctl – This system call sets up thread local storage for the newly launched process arguments involve code (**int**), which selects a subfunction and adds another argument which may be “**unsigned long**” for “set” operations or “**unsigned long ***” for the “get” operations.

munmap – After all file I/O is done, this system call unmap files into memory. This deletes the mappings for the specified address range, the region is automatically unmapped when process is terminated.

write – This is used to write to a file descriptor, arguments involve file descriptor fd, **const void* buf**, and writes up to third arg count bytes from the buffer pointed buf to the above file.

At the end, exit_group syscall is invoked, which exits all threads in a process, terminates the calling thread as well as the threads in calling process's thread group.

Task – 2 : Find file size using "lseek" and "stat" as well as blocks allocated to the file.

This program has two key functions invoked, **lseek** and **stat**, which are used to find the file size as well as blocks allocated to it. The strace output is nearly same as in the above case, for example, the same *execv*, *open*, *mmap*, *mprotect*, *munmap*, *read* and *write* which are associated with compilation of a program and running it. The following are the key system calls :

lseek (int *fd***, off_t** *offset***, int** *whence***)** – The function has three arguments, file-descriptor (fd – this is returned by the **open** syscall), offset and the third argument, being one of SEEK_SET, SEEK_CUR and SEEK_END, where in this case, it is SEEK_END, where the file offset is set to the size of the file plus offset bytes, which is 0 as given below ("os2.cpp" being the input file name for size calculation purposes) :

```
open("os2.cpp", O_RDWR)                 = 3
lseek(3, 0, SEEK_END)                   = 724
```

open syscall returns 3 as *fd*, which is argument in the following lseek syscall. The same syscall is called at the end of execution as well, with offset set by -1, signifying illegal seek, and close of opened files.

stat – This system call returns information about the input file, in the bufffer pointed to by a struct pointer, buf. It retrieves information about the file pointed to by **const char* filename**.

```
stat("os2.cpp", {st_mode=S_IFREG|0664, st_size=724, ...}) = 0
```

Difference between stat and lseek – file size and block calculation

The key difference between **stat** and **lseek** is the way they take their input. **lseek()** does not work from a filename, which expects an integer file handle, whereas **stat()** relies on filename. The second argument of *lseek* is **off_t offset**, and the third argument is the directive, **whence**, which affects the way lseek repositions the offset of the open file. If the **whence** directive is **SEEK_HOLE**, the file offset is adjusted to the next hole in the file greater than or equal to offset. This allows applications to map holes in a sparsely allocated file (useful for backup utils, preserving of holes) . A hole is a sequence of zeros that has not been allocated to the underlying file storage, so the difference in the file size can arise when the filesystem can support the operations by making **SEEK_HOLE** always return the offset of the end of the file, and making **SEEK_DATA** always return offset (i.e., even if the location referred to by offset is a hole, it can be considered to consist of data that is a sequence of zeros). This often leads to difference in calculation of number of blocks associated with a file

calculated using stat and lseek, since sparse file systems have holes in them, which changes the number of blocks. The number of blocks is usually smaller than file size in such systems. Also, in case of **lseek**, we simply divide obtained size by 512 bytes to get block number, but in case of **stat**, **st_blocks** is used. The field, **st_blocks** is how the OS indicates how much space is used by the file on disk. The actual units of allocation on disk are the choice of the file system, thus again creating a difference. **st_blocks** should be viewed as the amount of disk space used by the file, in units of 512 bytes, not as blocks, thus different from the way, we calculate blocks using lseek.

Task – 3 : Delete a directory and its all constituent folders/files

This program involves the deletion of a directory, along all its constituent files and folders and subfiles and subfolders. The key command used here is **nftw**, which walks throught the directory tree located under the specified input directory, dirpath, and calls **fn()** once, to *remove* each entry in the tree.

stat (const char *_pathname_, struct stat *_buf_) – This system call is invoked once again same as in the above program, but in this case, the first argument is the directory located under the specified input directory path.

```
stat("/home/saurabh_cs_iith/Desktop/test", {st_mode=S_IFDIR|0775, st_size=4096, ...}) = 0
```

lstat - Since the key function used is **nftw**, the whole directory is tree-walked in a recursive manner and each data entry (subfolders and subfiles) is examined for deletion, lstat system call needs execute (search) permission on all the directories in path that lead to file, and the only difference between *lstat* and *stat* is that, if input file path is symbolic (in case of directory), this link is "**stat-ed**" and not the file itself.

```
lstat("/home/saurabh_cs_iith/Desktop/test", {st_mode=S_IFDIR|0775, st_size=4096, ...}) = 0
```

getdents – This system call provides directory entries, as required for tree walking the directory under the specified directory path. The arguments are namely, the *file descriptor*, returned by the **open** system call, then the *number of entries* and the last, *size of the buffer*.

```
getdents(3, /* 2 entries */, 32768)     = 48
getdents(3, /* 0 entries */, 32768)     = 0
```

unlink – This system call deletes the name and the file it refers to, if it exists. The single argument is the pathdir, path of the directory.

```
unlink("/home/saurabh_cs_iith/Desktop/test") = -1 EISDIR (Is a directory)
```

rmdir – This system call is invoked after unlink deletes internal files, such that the directory is empty now, so that rmdir can delete it. The directory must be empty in order to be deleted by this system call.

```
rmdir("/home/saurabh_cs_iith/Desktop/test") = 0
```