Log Book Week 3

Jose Devian Hibono
1706039603
System Programming - C

Stat System Calls

The `stat()` system call returns information about the size and parameters associated with a file. The system call used is a File Management System Call. The attributes returned if use `stat()` are file's permissions, size, last modified, and timestamp.

Stat Structure

Stat structure is a function that has a return value of information regarding a file. Stat structure stores different information such as device info, inode number, inode type, inode mode, author info, size, date last accessed, date last modified.

Example of stat structure (taken from https://man7.org/linux/man-pages/man2/stat.2.html):

```
struct stat {
               dev_t     st_dev;         /* ID of device containing file */
               ino_t     st_ino;         /* Inode number */
               mode_t    st_mode;        /* File type and mode */
               nlink_t   st_nlink;       /* Number of hard links */
               uid_t     st_uid;         /* User ID of owner */
               gid_t     st_gid;         /* Group ID of owner */
               dev_t     st_rdev;        /* Device ID (if special file) */
               off_t     st_size;        /* Total size, in bytes */
               blksize_t st_blksize;     /* Block size for filesystem I/O */
               blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */

               /* Since Linux 2.6, the kernel supports nanosecond
                  precision for the following timestamp fields.
                  For the details before Linux 2.6, see NOTES. */

               struct timespec st_atim;  /* Time of last access */
               struct timespec st_mtim;  /* Time of last modification */
               struct timespec st_ctim;  /* Time of last status change */

           #define st_atime st_atim.tv_sec      /* Backward compatibility */
```

```
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
 };
```

File Description and File Descriptor

In your program, the file descriptor is an integer that corresponds to the file description in the kernel. File descriptor is a kind of indicator to access a file or I/O resource. File descriptors act as an interface to do that.

File description is the kernel structure that maintains the open state of the file (its current location, blocking / non-blocking, etc.).

Almost all processes have file descriptors, but there is a process named daemon, it closes the standard file descriptor to avoid hanging onto their resource. This is because it needs to ensure the daemon is reparented into init and removes any chance of the daemon to reacquiring a controlling tty.

Examples of Stat System Calls

`fstat()`

```
int fstat(int fd, struct stat *buf);
```

`fstat()` is strikingly similar to `stat()`, except that the file to be *stat*-ed is specified by the file descriptor fd.

`lstat()`

```
int lstat(const char *path, struct stat *buf);
```

`lstat()` is just the same as `stat()`, except that if the pathname is a symbolic link, it returns the link information itself, not the file to which the link refers.

`open()` and `creat()`

These functions are equal because if the pathname doesn't exist, both commands will create a new regular file. But if the pathname does exist, `open()` will just open the previous file and `creat()` will create a new file.

`open()` and `close()`

To avoid unwanted consequences, we have to close the process using close(). Once your program closes, the system will clean up after you. It will close any files you left open when it terminates your process, and perform any other cleanup that is necessary (e.g. if a file was marked delete-on-close, it will delete the file then; note that that sort of thing is platform-specific).

But if you do not close the process manually, it will then close your process abnormally. This can be via calling `abort(3)` or `exit(3)`, dying from a signal/exception, etc. This condition may suffer from buffered data problem.