Lecture 29 (4-8-26)

Systems Calls, File Systems, I/O (System Calls)

Motivating Questions:

1. What is a **system call?**
2. How do we lookup a file's **inode** information?
3. How do we walk a **directory**?
4. How do we copy a file's **data**?

Computing Stack

Top: Applications

- Libraries
- Operating System (kernel)
  Bottom: Computer Hardware

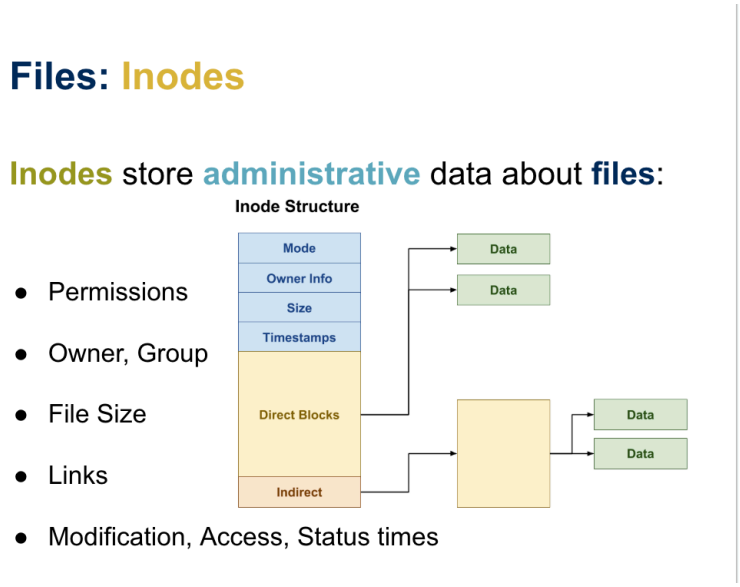
System Calls

- when libraries talk to the operating system / kernel
- Request a **service** from the **operating system:**
- **File system / Files / I/O**
  - Open/Close
  - Read/Write
  - Stat
  - Opendir/Readdir
- **Processes**
  - Fork/Exec
  - Wait
  - Kill
  - Signal
- **Networking**

- Getaddrinfo
- Socket
- Listen/Bind
- Connect

Files: Inodes

- **Inodes** store **administrative** data about **files/directories:**



- `strace stat Makefile`
  - this command gives you all the system calls that were made when running `stat Makefile`
- All the system calls have a man page
  - `man stat` -> Manual section 1 (User Commands)
  - `man 2 stat` -> Manual section 2 (System Calls)
- everytime you make a system call, you need to check if it fails
  - if a system call fails it assigns a number to the global `errno` variable
    - from this number you can identify the error

`list.c`

```
/* list.c */

#include <errno.h>
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

/* Node Structure */

typedef struct Node Node;
struct Node {
     char *path;
     size_t size;
     Node *next;
};

// TODO

Node *  node_create(const char *path, size_t size) {
    // TODO
    Node *n = calloc(1, sizeof(Node));
    if (n){
         n->path = (char *)path;
         n->size = size;
    }
    return n;
}

void    node_delete(Node *n) {
    // TODO
}

/* List Structure */

// TODO
typedef struct{
     Node *head;
}List;

void    list_add(List *l, const char *path, size_t size) {
    // TODO
    Node *n = node_create(path, size);
    // Case 1: Empty List
```

```c
    if (!l->head){
        l->head = n;
    } else { // Case 2: Non-Empty List
        Node *tail = l->head;

        while (tail->next){
            tail = tail->next;
        }

        tail->next = n;
    }

}

/* Functions */

void    add_files(const char *root, List *files) {
    // TODO
    DIR *d = opendir(root);
    if (!d){
        fprintf(stderr, "opendir(%s): %s\n", path, strerror(errno));
        return;
    }

    for (struct dirent *e = redir(d); e; e = readdir(d)){
        if (strcmp(e->d_name, "." == 0 || strcmp(e->d_name, ".." == 0){
            continue;
        }
        printf("%s\n", e->d_name);
    }
    closedir(d);
}

void    print_files(List *files) {
    // TODO
    for (Node *curr = files->head; curr; curr = curr->next){
        printf("[%lu] $s\n", curr->size, curr->path);
    }
}

/* Main Execution */

int main(int argc, char *argv[]) {
    // TODO: Determine directory
    char root[BUFSIZ];
```

```
    if (argc >= 2){
        strcpy(root ,argv[1]);
    } else {
        getcwd(root, BUFSIZ); //system call!!!!
    }
    printf("root is %s\n", root);

    // TODO: Add files and print them
     List l = {NULL};
     add_files = (root, &files);
     print_files(&files);

    // TODO: Release resources

    return EXIT_SUCCESS;
}

/* vim: set sts=4 sw=4 ts=8 expandtab ft=c: */
```