Operating Systems

Recitation 1

ציון

- 25% תרגילים
  - עבודה עצמית ביחידים.
  - מותר להשתמש בדוגמאות קוד לשימוש כללי שמצאתם (לדוגמה, קוד C לפתיחת קובץ), כל עוד:
    - הקוד בו השתמשתם אינו קשור ספציפית לקורס שלנו, **כולל בשנים קודמות**.
    - סימנתם במפורש את הקוד הזה בתוך הקובץ שלכם בהערה הכוללת את המחרוזת **"CREDIT"**, כולל מצביע למקום בו מצאתם את הקוד.
    - שילבתם את הקוד הזה בקוד שלכם באופן חלק ועקבי, מבחינת שמות משתנים, אינדנטציה, טיפול בשגיאות וכו'.
  - כל שימוש אחר בקוד של מישהו אחר יחשב העתקה.
  - העתקה מהווה אי-עמידה בחובות הקורס, ותמנע גישה למבחן.
  - פרטים נוספים בתרגול.
- 75% מבחן.
  - המבחן יכלול בעיקרו שאלות רב ברירה, ועד 20% שאלות פתוחות.

Assignments

- All homework assignments must be performed **individually**. We are serious about this!
  - **CHEATING == COPYING == FAILING!**

- **Five** homework assignments (25% of final grade)
  - Grades published on website
  - Solutions & **issues** – part of the material for the exam

- HW is **extension** of APIs that were taught in class
  - Requires **independent study beyond** recitations!

Submission guidelines

- Submission by **moodle**
- 2 weeks for submitting

- **All** count for your final grade (25%)

- All due on **Thursday**
- Sick / miluim? **Notify in advance**, submit **with note**

Submission guidelines

- Read the moodle forums and notifications constantly
  - It’s your responsibility to stay updated!

- ***Carefully read detailed guidelines in moodle***
  - Working submission is basic requirement and **not** enough!
  - Point deductions for wrong file types, names, directory structure, proper error handling, memory handling, etc.

  - **All** detailed in submission guidelines

~~Recitations~~

- **Labs**, not recitations
- **Not necessarily in order of lectures!**
- Will try uploading presentation before class
  - But changes are made last minute, **final** slides only in class!

~~Recitations~~

Each class generally consists of 2 parts:

- **Part I:**
  - Review of general OS concept and its Unix implementation & APIs
- **Part II:**
  - “On-projector” development/coding of an application which makes use of the presented concept

Useful Resources

- Sivan Toledo book on Operating Systems (בעברית)
- http://linux.die.net – Linux documentation, All man pages, various guides and more
- http://www.trembath.co.za/commands2.html - useful summary of command line utilities
- http://beej.us/guide/bgnet/ - illustrative guide to network programming
- http://lwn.net/Kernel/LDD3/ - book (pdf) on linux device drivers and kernel internals

Useful Resources

- Everything at www.google.com.
  - **Really!**

Plan

- Virtual Machine
- System calls
  - Intro
  - File-related
  - Directories (time permitting)

Unix Development

- **University –**
  - Use nova/soul
  - Any command line editor (kate, vim, pico, emacs, gedit)
  - No makefile required!

- **At home –**
  - VM or popular distribution (Ubuntu, Debian)
  - All of the above editors

  - Most assignments: single C file without header
    - From command line: *gcc hw1.c –o hw*

Virtual Machine

- Emulation of a computer system / operating system
- Allows us to use various OS under (almost) any OS
  - Such as Linux under Windows

- Currently a black-box for us
- We will learn later how it works

Virtual Machine

- We’ll do some sensitive programming on **Linux**
- For consistency & safety, use VM at home
  - We’ll use **VirtualBox** with **Lubuntu**

- *Installation guidelines are in moodle*

System Calls

- A service the operating system (OS) provides to applications.
- Kernel sensitive operations - hardware access (IO), special privileges, etc.

- We'll explain later in the course how they really work

Command Shell Processor

- Terminal in Ubuntu (LXTerminal)
- Linux shell: bash

- Assignments and in-class examples will be Unix console applications.

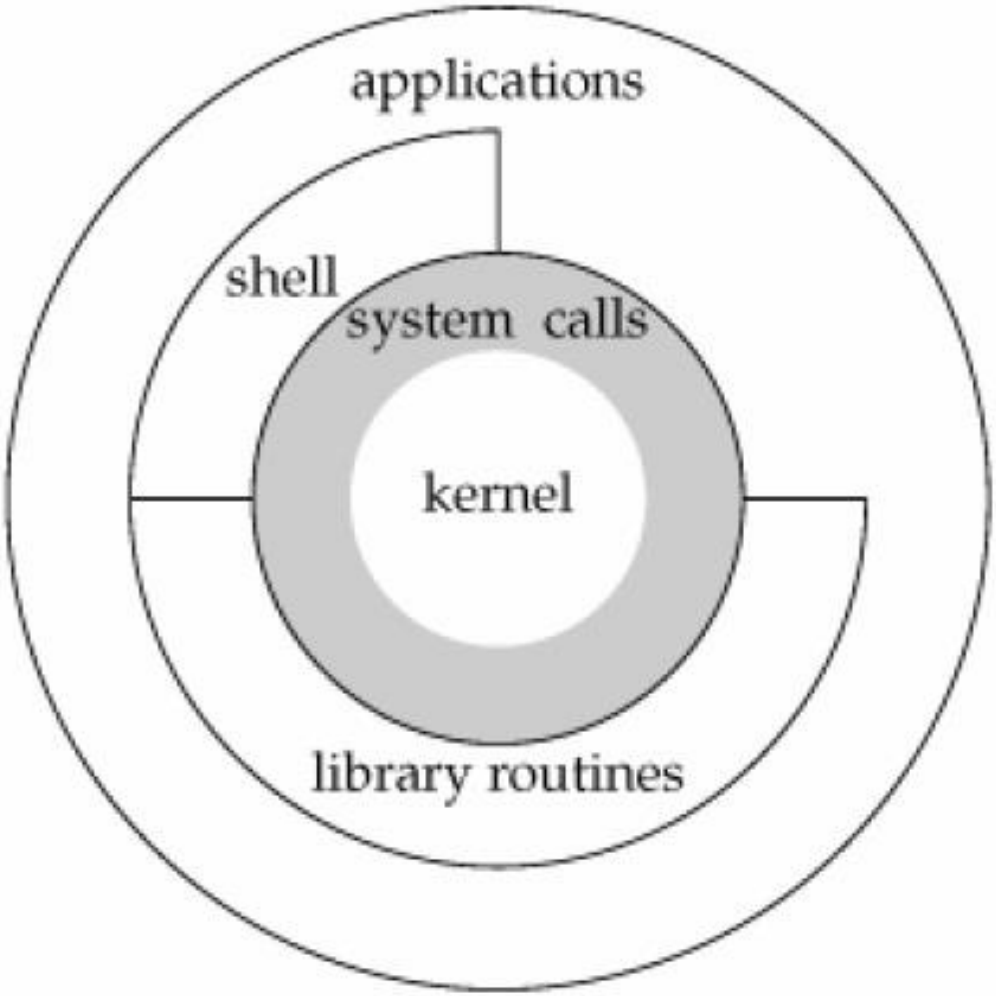
- Very useful:
  - *man*
  - *man 2* – for system calls

Kernel

- OS core is the Kernel
- OS Kernel manages internal data structures for users

- To really get the most out of it, sometimes need to know what goes on “under the hood”
  - How? taking operating systems course helps (sometimes)

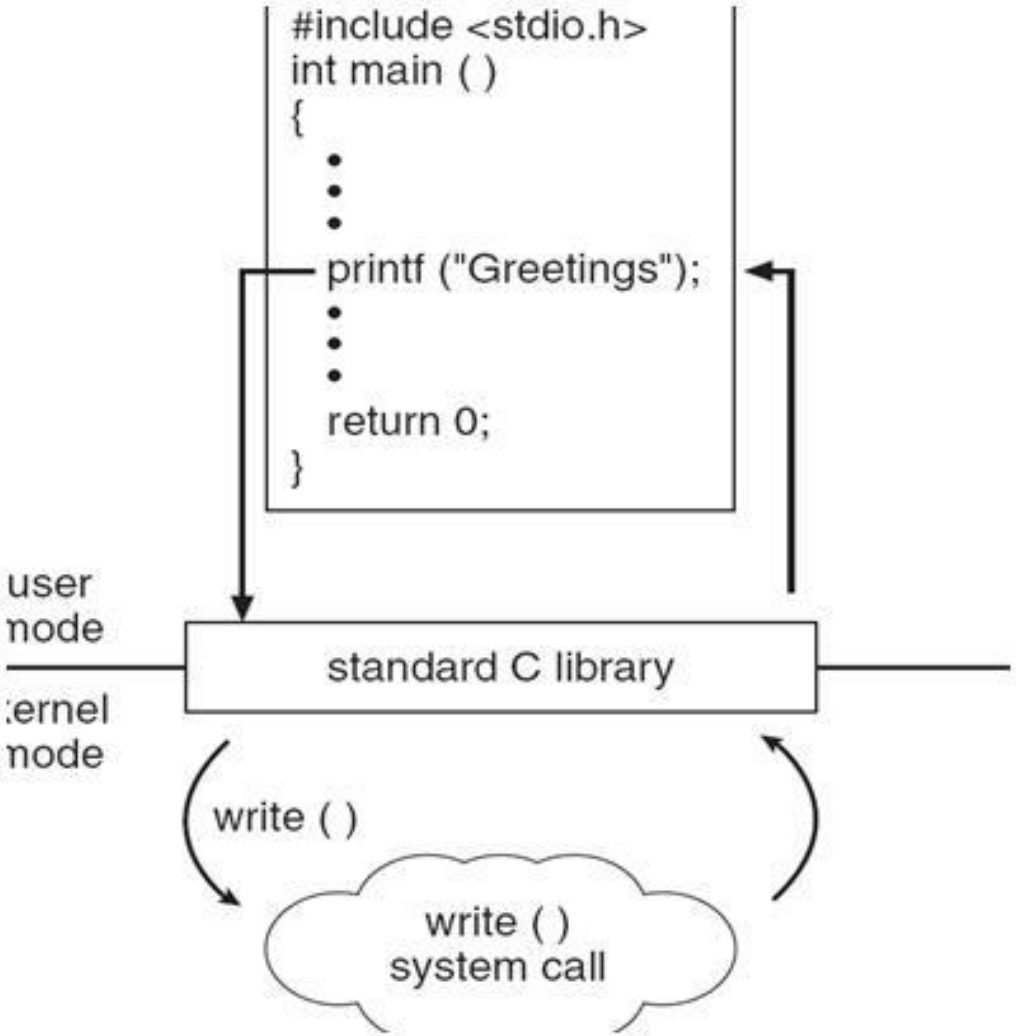
OS Structure



Unix Files

- Return/manipulate internal kernel handle
  - `creat()`
  - `open()`
  - `stat()`
  - `read()`
  - `write()`
  - `close()`

User Space/System call



open()/close()

```
int open(const char *pathname, int flags, mode_t mode)
int close(int fd)
```

- Open file, return “file descriptor” == internal handle
  - Handle == offset in kernel data structure…
  - path – path to new file (“/home/moshesulamy/newfile.txt”)
  - flags – access modes
    - `O_RDONLY, O_WRONLY, O_RDWR`
    - `O_CREAT` – create file (if doesnt exist)
  - mode – file permissions.
    - Restrict access of others to files
    - Not obligatory, except when?
  - Returns -1 on failure, or valid positive number on success
- `Close()` - deletes handle

creat()

```
int creat(const char *path, mode_t mode)
```

- Create new file
- What does man page say?

read()

```
ssize_t read(int fd, void* buf, size_t count)
```

- Read data from file
  - fd – file descriptor (internal handle) of open()ed file
  - buf – buffer to read data into
  - count – how many bytes to read

- Returns number of bytes read, or -1 on error.

write()

```
ssize_t write(int fd, void* buf, size_t count)
```

- Write data to file
  - fd – file descriptor (internal handle) of `open()`ed file
  - buf – buffer to read data into
  - count – how many bytes to write

- Returns number of bytes written, or -1 on error.

stat()

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

- Retrieve file information
  - path – path to file
  - buf – struct with relevant fields (see man 2 page)

- Returns 0 on success, or -1 on error

Directories

- Containers of files
- Basically files too
  - Accessed with special handles

mkdir()

```
int mkdir(const char *pathname, mode_t mode)
```

- Create a new directory
  - Pathname = full path to new directory
    - “/home/moshesulamy/new_firectory_name”
  - Mode = permissions

- Returns 0 on success, -1 if an error occurred

opendir()

```
DIR *opendir(const char *name)
```

- "open" a directory
  - name = path to directory to open
- Returns special handle of type DIR
  - Actually a wrapper for regular file descriptor.

readdir()

```
struct dirent *readdir(DIR *dirp)
```

- Read contents of a directory = serially iterate over directory entries
  - dirp = special handle to open directory
- Returns next directory entry to read
  - Repeated calls will eventually iterate all entries
  - After iterating all, returns NULL

Let’s do some programming

- Use console window:
  - Either:
    - Open and read first 10 characters of a file
    - Make system calls to fetch list of the files in current directory
  - Print out results

- See *rec1.c* in moodle