How to read a code repository?
Read a repository: 3 passes

1st pass
- read documents and filenames

2nd pass
- track the execution: earth → grass → applications

3rd pass
- read the details of a module, such as the SD card driver
Documents of egos-2000

- README.md
  - Explain why the project is important
- references/USAGES.md
  - Explain how to use this project
- references/README.md
  - Explain the internal design of the project
Read filenames: earth

• From the documents:

Earth layer (hardware specific)

• earth/dev_disk: ROM and SD card (touched by P4)
• earth/dev_tty: keyboard input and tty output
• earth/cpu_intr: interrupt and exception handling (touched by P3)
• earth/cpu_mmu: memory paging and address translation (touched by P3)
Read filenames: **earth**

- `cpu_intr`, `cpu_mmu`, `dev_disk`, `dev_tty` are explained in the documents.
- `gpio` and `uart` are buses, just like `usb`; Search them on Wikipedia.
Read filenames: earth

- `cpu_intr`, `cpu_mmu`, `dev_disk`, `dev_tty` are explained in the documents

- `gpio` and `uart` are buses, just like `usb`; Search them on Wikipedia

- `earth.S` and `earth.c` are for initialization

- `earth.lds` specifies the memory layout
Read filenames: `earth/sd`

- `sd.h` provides basic definitions
- `sd_init.c` initializes the SD card
- `sd_rw.c` provides SD card read and write
- `sd-utils.c` provides helper functions
- We will read this module in details later
Read filenames: **grass**

- From the documents:

  Grass layer (hardware independent)
  - `grass/timer`: timer control registers
  - `grass/syscall`: system call interfaces to user applications
  - `grass/process`: data structures for managing processes (touched by P1)
  - `grass/scheduler`: preemptive scheduling and inter-process communication
Read filenames: grass

- process, syscall, timer, scheduler are explained in the documents
- grass.S and grass.c are for initialization
- grass.lds specifies the memory layout
Read a repository: 3 passes

• 1st pass
  • read documents and filenames

→ 2nd pass
  • track the execution: earth → grass → applications

• 3rd pass
  • read the details of a module, such as the SD card driver
The Key:

Find `main()` functions and track executions from there
grep is a useful command

> cd egos-2000
> grep "mainC" -r *
Main functions in the repository

> cd egos-2000  
> grep "main(" -r * 

earth/earth.S: /* Call main() of earth.c */

```
grass/grass.S: /* Call main() of grass.c */
grass/grass.c:int main() {
```

apps/*.c: /* Every application has a main() function */
Main function in earth

• Read earth.s and earth.c
  • Boot loader disable interrupt and call earth main()
• Earth main()
  • Initialize memory for earth layer
  • Initialize dev_tty, dev_disk, cpu_intr, cpu_mmu
• Load and enter the grass layer
Main function in grass

- Read `grass.s` and `grass.c`
  - Initialize PCB data structures
  - Initialize the timer and enable interrupt
  - Load and enter the first application: `GPID_PROCESS`
- Where is `GPID_PROCESS` defined?
Find GPDID_PROCESS

> cd egos-2000
# Find which header file contains GPDID_PROCESS
> grep "GPDID_PROCESS" -r * | grep "\.h"

library/servers/servers.h: GPDID_PROCESS,
library/servers/servers.h:/* GPDID_PROCESS */
Kernel Processes (aka. Daemon)

```c
enum grass_servers {
    G PID UNUSED,
    G PID_PROCESS,
    G PID_FILE,
    G PID_DIR,
    G PID_SHELL,
    G PID_USER_START
};
```

- **G PID_PROCESS**
  - spawn and kill processes
- **G PID_FILE & G PID_DIR**
  - something about file system
- **G PID_SHELL**
  - shell for entering commands
Control Flow Sketch

- **During boot up**
  - earth main() → grass main() → GPID_PROCESS
  - GPID_PROCESS → GPID_FILE
  - GPID_PROCESS → GPID_DIR
  - GPID_PROCESS → GPID_SHELL
- **After boot up**
  - GPID_SHELL → user applications
Two more main functions to read

> cd egos-2000
> grep "main()" -r *
earth/earth.S: /* Call main() of earth.c */
earth/earth.c: int main() {
grass/grass.S: /* Call main() of grass.c */
grass/grass.c: int main() {
tools/mkrom.c: int main() {
tools/mkfs.c: int main() {

apps/*.c: /* Every application has a main() function */
mkfs and mkrom

• During **make**, the RISC-V compiler compiles **egos-2000**
  • i.e., create everything under **build/**

• During **make install**,
  • **mkfs** creates **disk.img**
  • **mkrom** creates **bootROM.bin**
Reading **main()** provides a **rough picture**

```plaintext
> cd egos-2000
> grep "main(" -r *
earth/earth.S:    /* Call main() of earth.c */
earth/earth.c:int main() {
grass/grass.S:    /* Call main() of grass.c */
grass/grass.c:int main() {
tools/mkrom.c:int main() {
tools/mkfs.c:int main() {

apps/*.c:    /* Every application has a main() function */
```
We know the structure of the work and some details.
Read a repository: 3 passes

- 1st pass
  - read documents and filenames

- 2nd pass
  - track the execution: earth → grass → applications

- 3rd pass
  - read the details of a module, such as the SD card driver
Now is a **good time** to read the SD driver

- CS4411 has 12 lectures:
  - Step #1: understand computer architecture
  - Step #2: understand interrupt and exception
  - Step #3: understand context-switch and multi-threading
  - Step #4: understand privilege levels
  - Step #5: understand i/o devices
  - Step #6: understand file systems
In your career:

**Find and read** the module that is **the most relevant** to your assigned job
Part #1 of earth/sd.h

- `spi` is a bus, just like `usb`
- `section19` of SiFive document
- The document is a dictionary for reference, instead of a textbook!
- read only when necessary

```
#define SPI1_BASE 0x10024000UL
#define SPI1_SCKDIV 0UL
#define SPI1_SCKMODE 4UL
#define SPI1_CSID 16UL
#define SPI1_CSDEF 20UL
#define SPI1_CSMODE 24UL
#define SPI1_FMT 64UL
#define SPI1_TXDATA 72UL
#define SPI1_RXDATA 76UL
#define SPI1_FCTRL 96UL
```
Part #2 of earth/sd.h

Read and write disk blocks

- void sdinit();
- int sdread(int offset, int nbblock, char* dst);
- int sdwrite(int offset, int nbblock, char* src);

Send commands to SD card

- char sd_exec_cmd(char*);
- char sd_exec_acmd(char*);

Send bytes to SD card

- char recv_data_byte();
- char send_data_byte(char);
Part #2 of earth/sd.h

void sdinit();
int sdread(int offset, int nbblock, char* dst);
int sdwrite(int offset, int nbblock, char* src);

char sd_exec_cmd(char*);
char sd_exec_acmd(char*);
char recv_data_byte();
char send_data_byte(char);
What to understand in `sd_utils.c`

```c
char send_data_byte(char byte) {
    while (REGW(SPI1_BASE, SPI1_TXDATA) & (1 << 31));
    REGB(SPI1_BASE, SPI1_TXDATA) = byte;

    long rxdata;
    while (((rxdata = REGW(SPI1_BASE, SPI1_RXDATA)) & (1 << 31));
    return (char)(rxdata & 0xFF);
}

inline char recv_data_byte() {
    return send_data_byte(0xFF);
}
```

How to send and receive bytes to/from the SD card?
What to understand in `sd_utils.c`

```c
char sd_exec_cmd(char* cmd) {
    for (int i = 0; i < 6; i++) send_data_byte(cmd[i]);

    for (int reply, i = 0; i < 8000; i++)
        if ((reply = recv_data_byte()) != 0xFF) return reply;

    FATAL("SD card not responding cmd%d", cmd[0] ^ 0x40);
}
```

How to **send commands** to the SD card?
What to understand in `sd_rw.c`

```c
static void single_read(int offset, char* dst) {
    /* Wait until SD card is not busy */
    while (recv_data_byte() != 0xFF);

    /* Send read request with cmd17 */
    char *arg = (void *)&offset;
    char reply, cmd17[] = {0x51, arg[3], arg[2], arg[1], arg[0], 0xFF};

    if (reply = sd_exec_cmd(cmd17))
        FATAL("SD card replies cmd17 with status 0x%2x", reply);

    /* Wait for the data packet and ignore the 2-byte checksum */
    while (recv_data_byte() != 0xFE);
    for (int i = 0; i < BLOCK_SIZE; i++) dst[i] = recv_data_byte();
    recv_data_byte();
    recv_data_byte();
}
```

How to read a block from the SD card?
Homework

- **P4** will be released and is optional.
- **P5** will be released and due on Dec 2.
- No class next week (Nov. 11)
  - switch to office hours in Gates 437 due to Veterans day
- Lecture on Nov. 18: file systems