Project 2-2
User Programs

Prof. Jin-Soo Kim (jinsookim@skku.edu)
T.A. – Sejun Kwon (sejun000@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

Project 2-2 Review

- **Pointer operation**
  - *((unsigned int *)argc) = 3;

- **Test**
  - In Make.vars file -> SIMULATOR = bochs
  - In the cmd “make check”
    - pintos ……. /dev/null

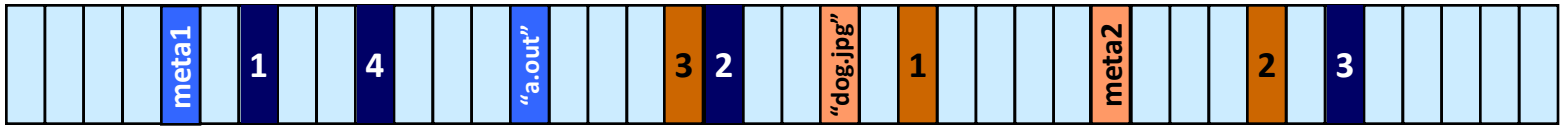
Supporting User Programs

- **What should be done to run user programs?**
  1. Provide file system accesses
  2. Process wait / exit
  3. Pass arguments
  4. Provide system calls

File Systems (1)

- **Block device abstraction**
  - A set of sectors (sector = 512 bytes)

- **File systems**
  - <filename, metadata, data> → <a set of sectors>



File Systems (2)

- **Pintos file system**
  - No internal synchronization
  - File size is fixed at creation time
  - File data is allocated as a single extent (i.e., in a contiguous range of sectors on disk)
  - No subdirectories
  - File names are limited to 14 characters
  - Unix-like semantics: If a file is open when it is removed, it may still be accessed by any threads that it open, until the last one closes it.

File Systems (3)

- **Using the Pintos file system**

```
$ pintos-mkdisk --filesys-size=2 filesys.dsk
```

  - Creates a 2MB disk named "fs.dsk"

```
$ pintos -f -q
```

  - Formats the disk (-f)

```
$ pintos -p ../../examples/echo -a echo -- -q
```

  - Put the file "../../examples/echo" to the Pintos file system under the name "echo"

```
$ pintos -q run 'echo x'
```

  - Run the executable file "echo", passing argument "x"

```
$ pintos --filesys-size=2 -
  p ../../examples/echo –a echo -- -f –q run
  'echo x'
```

File Systems (4)

- **Note:**
  - User programs are loaded from the file system
    - To run a program, it should be moved to the file system.
    - Executable files are in ELF format
    - Data files can be also stored in the file system
  - Refer to the interfaces provided by the file system
    - In `filesys/filesys.h` and `filesys/file.h`
    - You will need them to implement system calls related to file systems
      - » e.g., `create()`, `remove()`, `open()`, `filesize()`, `read()`, `write()`, `seek()`, `tell()`, `close()`
  - You don't need to modify the file system code (under the `filesys` directory) for this project

File Systems (5)

- **File descriptor**
  - Per process
- **Kernel File table (struct file)**
  - Global table
- **Inode**
  - Only one for a file

- **The thing you have to do is??**
  - Implement a file descriptor table per process

System Calls (1)

- **System calls related to processes**

```
void   exit (int status);
pid_t  exec (const char *cmd_line);
int    wait (pid_t pid);
```

  - All of a process's resources must be freed on exit()
  - The child can exit() before the parent performs wait()
  - A process can perform wait() only for its children
  - Wait() can be called twice for the same process
    - The second wait() should fail (-1)
  - Nested waits are possible: A → B, B → C
  - Waits for multiple threads are possible

System Calls (2)

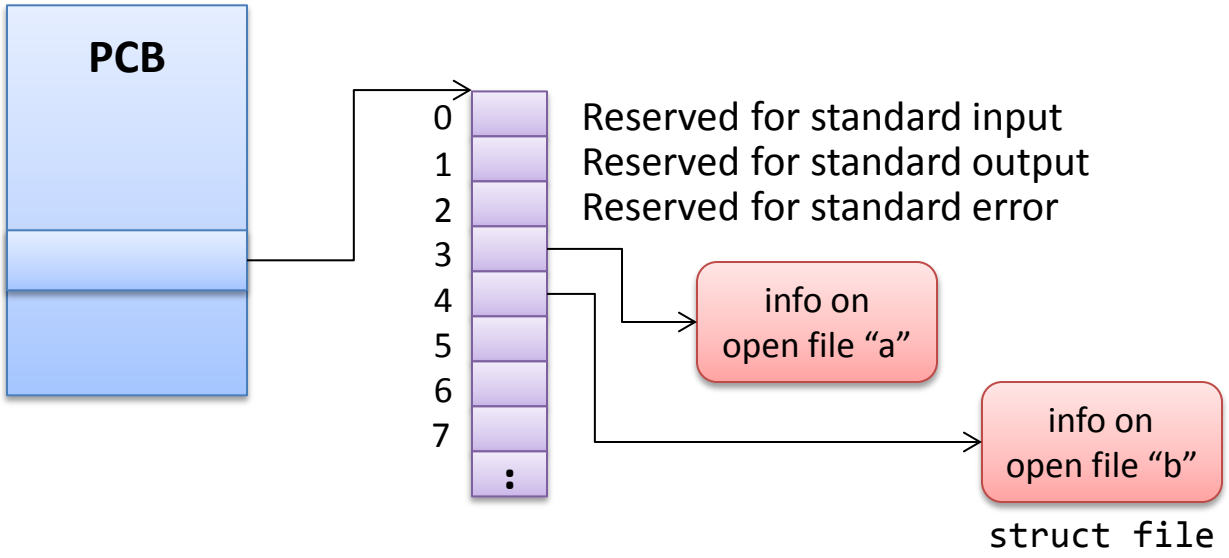
- **System calls related to files**

```
bool   create (const char *file, unsigned initial_size);
bool   remove (const char *file);
int    open (const char *file);
int    filesize (int fd);
int    read (int fd, void *buffer, unsigned size);
int    write (int fd, void *buffer, unsigned size);
void   seek (int fd, unsigned position);
unsigned tell (int fd);
void   close (int fd);
```

- create()/remove()/open() work on file names
- The rest of them work on file descriptors

System Calls (3)

- **File descriptor**
  - An integer (C type `int`)
  - An index for an entry in a kernel-resident data structure containing the details of all open files (file descriptor tables)



System Calls (4)

- **Implementing system calls related to files**
  - No need to change the code in the `filesys` directory
  - The existing routines in the `filesys` directory work on the "`file`" structure (`struct file *`)
  - Maintain a mapping structure from a file descriptor to the corresponding "`file`" structure
  - Deny writes to a running process's executable file
  - Ensure only one process at a time is executing the file system code

Project 2-2

- **make check in userprog/build**
- **Remainder of tests (65 tests)**
  - Bad pointer handling
  - Synchronization

Submission

- **Document**
  - Nov 11, 11:59PM
  - [group_number]_project2.pdf
  - File descriptor management
  - Management between parent and child processes
    - Pid(tid), lookup parent or children
  - Synchronization
  - 2pages

Submission

- **Source Code**
  - Nov 11, 11:59PM
  - [group_number]_project2-2.tar.gz
  - Tar and gzip your Pintos source codes
    - after make clean
  - Upload it at sys.skku.edu
  - threads, userprog, filesys directories