SYSTEM SOFTWARE
STARTUP AN APPLICATION PROGRAM

Nguyen Huu Duc

School of Information and Communication Technology
Hanoi University of Science and Technology

Startup an application program

- Operating system loads an application program
  - Create a new process
  - Prepare memory space
  - Prepare environment information
  - Move to the entry point of the program `_start`
- Execute the loaded program
  - Prepare execution environment `crt1.o`
  - Call the function `_main`

Start up a application program for OS

- An application program is executed from a shell
  - Shell (Parent process) creates a new process (child process) by using the system call `fork()`
  - The child process's structure is the same to that of the parent process, but ID is different.
  - The child process calls the function `execve()` to load and execute the application program

A simple shell

```
while(1) {
    printf("> ");
    scanf("%s", str);  // fgets(str, 128, stdin);
                       // *strchr(str, '\n') = '\0';
    if((pid = fork()) == 0) {
       // child
       execve(str);    // argv, envp
    }
    wait(&status);
}
```

execve()

```
execve(const char *fn, char *argv[], char *env[])
```

- Execute the application program `fn` with a list of parameter `argv` and environment variables `env`
  - `fn` is a path to application file
  - `argv` is an array pointing to strings which are correspondent to command line parameters (The last element is NULL)
  - `env` is an array pointing to strings which are correspondent to environment variables (Ending by NULL, each element has a form of `key=value`)

execve()

- Allocate memory for the struct `linux_binprm` describing information about the execution application
- Read `dentry object`, `file object`, `inode object` from the execution file
- Execute `prepare_binprm()` to fill information in the struct `linux_binprm`
- Store the execution file name, command line parameter, environment variable onto one or more memory pages
- Execute `search_binary_handler()` to find and load the correspondent execution function (`load_binary`)

execve()

- Check the execution file's magic number
- Read the header of the execution file
- Get the path of application interpreter
- Read `dentry object`, `file object`, `inode object` of the application interpreter
- Check the execution permission and other factors of the application interpreter
- Release resources of the current process (`flush_old_exec()`)

execve()

- Mapping page which stores parameter and environment onto the memory space of the application (`setup_arg_pages()`)
- Allocate memory for pages text, data, etc. of the application program (`do_mmap()`)
- Load the interpreter (`load_elf_interp()`)
- Set values `start_code, end_code, ....`
- Allocate memory for the bss section (`do_brk`)
- Execute the application program (`start_thread`)

Execute an application program

- An application program starts from the symbol `_start` (defined in crt1.c)
  - Prepare to call `main()`
  - Register the called programs when finishing the program (`atexit()`)
  - Call `_init()` (Define in `crtbegin`)
  - Call `main()`
  - Call `exit()` if a control is returned from `main()`

Execution of main()