Lecture 2
Process Management
January 17, 2005

Processes
- A process is an instance of program during execution, e.g., this Powerpoint program.
- I can run Powerpoint several times.
- Other processes are running at the same time (calendar, email, browser, ...).
- Only a process can create another process.
- Linux: init process.

Boot Procedure
- BIOS: lowest-level interface between computer & peripherals.
- BIOS: integrity check in memory and seeks instructions in MBR (master boot record) on disk.
- Linux: MBR points to bootloader (LILO, GRUB).
- Bootloader loads kernel image and starts the init process (/sbin/init).
- Init is the root/parent of all processes in Linux.
- Init starts processes defined in /etc/inittab to make system functional.
- Linux selects ‘run-level’ (0=shutdown, 3+=fully operational)

What’s a Process?
- PID (identifier)
- executing program (code)
- memory (code, data)
- program counter
- signal handlers
- set of registers
- ...
- -> state

How to Determine my ID?
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void); //my pid
pid_t getppid(void); //my parent’s pid

Creating New Processes
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
fork()

- Child is EXACT copy of parent.
- Same program, same PC position, memory is exact copy, signal handlers, file descriptors are copied.
- What’s different?
  - child: fork() returns 0
  - parent: fork() return child’s PID or -1 (error)

Example

```c
pid_t x;
printf("pid=%d\n", getpid());
printf("ppid=%d\n", getppid());
x = fork();
printf("pid=%d\n", getpid());
printf("ppid=%d\n", getppid());
printf("x=%d\n", x);
```

exec()

- Switch to another program.
- Code/data is destroyed.
- Environment variables and file descriptors are kept identical.
- New code loaded, then started.
- No way back.

```c
int x=3;
int err=exec("hello");
printf("x=%d\n", x);
```

More About exec()

- Fork/exec often used together.
- 4 versions:
  - absolute path     search in $PATH
  - n parameters      execv()     execvp()
  - 1 array           execv()     execvp()
  
  `char *params[]={"ls", ",-l", "/home/cpoeellab", NULL};
  execv("/bin/ls", params);`

  - Parameter list must start from argv[0] and end with NULL pointer.

Stopping a Process

- `main()` returns.
- Explicit `exit()`.
- A process can send a signal to another process.
- Signal handler determines action, may stop.
- SIGINT: stops process (^C), handler can be overwritten.
- SIGKILL: same effect, can’t be overwritten.
Sending Signals

```c
#include <sys/types.h>
#include <signal.h>
int kill (pid_t pid, int sig);
```

Catching Signals

```c
void myhandler(int sig) {
    printf("I received signal %d\n", sig);
    signal(SIGINT, myhandler);
}

int main () {
    signal(SIGINT, myhandler);
    ...
}
```

Zombie Processes

- Finished execution, stays until parent received termination status. If parent is gone, it’s adopted by process 1 (init).
- Must be dealt with (steal resources).
- Parent must “wait” for it, block until children are dead or set up signal handler.

Waiting for Child Process

```c
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
```

- `wait()` waits for completion of child (any), including zombies; `status` is process’ return status.
- Also: `waitpid()`, offers more control.

```c
void sig_child(int sig) {
    pid_t pid;
    int stat;
    while ((pid=waitpid(-1, &status, WNOHANG))>0) {
        printf("Child %d terminated with %d\n", pid, status);
    }
    signal(SIGCHLD, sig_child);
}
```