Operating Systems - Lab 1

Tuesday 14th February, 2023

Instructions

- Group enrolment on Themis
  `https://themis.housing.rug.nl/course/2022-2023/os`
- Submit in pairs
- Deadline: 24th Feb, at 23:59am
- Programming language: C

Requirements

- Simple commands: (`echo hello`)
- Command composition (`echo hello && echo bye`)
- A basic parser is provided
- The use of `system()` is not allowed!

Command Examples

- `echo "hello world"`
- `echo "a" && echo "b" && echo "c"`
- `echo "a && b"`
- `echo "hello" || echo "not hello"`
- `ls . && cat 1.in && echo "File printed" || echo "File not printed"`
- `cat 1.out || echo "File does not exists"; echo "Task done"`
- `echo "hello" && exit; echo "Task done"`
- `hello`
- `./folder/hello`

Functions

- `fork()`
  - To create a child process.
- `exec()`
  - To execute a specific file in the current process.
- `exit()`
  - To terminate the process with an exit status.
- `wait()`
  - To suspend execution until child process terminates.

fork()

```
pid_t fork(void);
```

- Used for creating a new process called a **child process**
- Child process
  - Duplicates the **parent process**
  - Runs concurrently with the parent process
- Returns:
  - < 0 if the fork failed
  - 0 in the child process
  - > 0 in the parent process.
    - The returned value is the pid of the created child process.

fork() Simple Example

```c
int main(){
    fork();
    printf("Hello world!\n");
    return 0;
}
```

Output:

```
Hello world!
Hello world!
```

fork() Example

```c
int main() {
    pid_t pid;
    pid = fork();
    if(pid < 0) {
        fprintf(stderr, "fork() could not create a child process!");
        exit(0);
    } else if (pid == 0) { // Only child process gets here
        printf("Child process is executing...");
    } else { // Only parent process gets here
        wait(NULL); // Wait for any child process to finish
        printf("Child process terminated! Parent process executing...");
        exit(0);
    }
    return 0;
}
```

Output:

```
Child process is executing...
Child process terminated! Parent process executing...
```

exec() family

```
int execv(const char *path, char *const argv[]);
```

- Allows a process to run a program file, which includes binary executables or a shell scripts
- It replaces the current process image with a new process image

Functions:

- `execv()`
- `execlp()`
- `execl()`
- `execvp()`
- `execvpe()`
- `execle()`

exit()

```
void exit(int status);
```

- Terminates the calling process immediately.
- Processes (esp. Child processes) should use it, when they are finished executing.
- The status value is returned to the parent process.

wait()

```
pid_t wait(int* status);
```

- A call to `wait()` blocks the calling process until one of its child processes exits or a signal is received.
- If there are multiple child processes, then the parent will wait until **one** of them exits.

Orphan Process

A running child process without a running parent process

- In the assignment code, make sure to prevent orphan processes, it is explicitly checked by Themis. Not handling it will cause the submission to fail!

Zombie Process

A process whose execution is completed but it still has an entry in the process table

- No resources (memory, etc) are allocated to zombie process, in a way it is dead, thus, called a "zombie".
- It usually occurs for child processes, as the parent process still needs to read its child's exit status, using `wait()`.
- If the parent process does not use the `wait()` system call, the zombie process is left in the process table. This creates a resource leak.
- If there are many zombie processes, and their parent processes are not running anymore, then this can create a big issue, such as system running out of process table entries.

RTFM

Use the man pages!

- `man exec`
- `man getenv`
- `man fork`
- `man wait`