System Programming & OS 실습
13. MyShell

이성현, 최민국

Dankook University

{leesh812, mgchoj}@dankook.ac.kr

Index

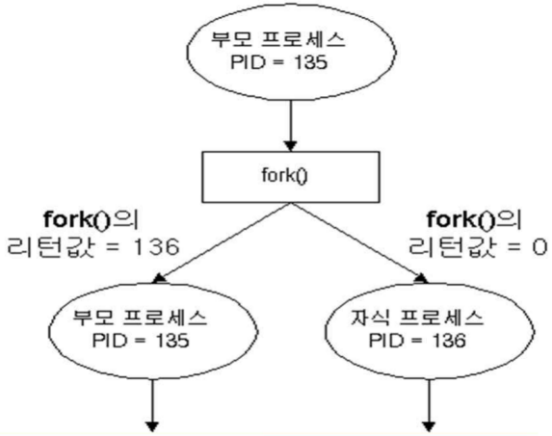
- Practice 1 : fork()
- Practice 2 : exec()
- Practice 3 : Myshell
  - 3-1. Tokenizing
  - 3-2. Built in Command
  - 3-3. Background
  - 3-4. Redirection
  - 3-5. Pipe

System calls

- Basic
  - fork(), clone() : create a task
  - execve() : execute a new program (binary loading)
  - exit() : terminate a task
  - wait(), waitpid() : wait for a task's termination (child or designated)
  - getpid() : get a task ID

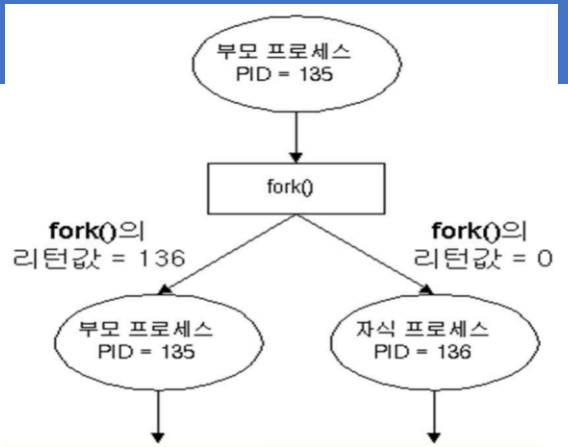
P1. fork()

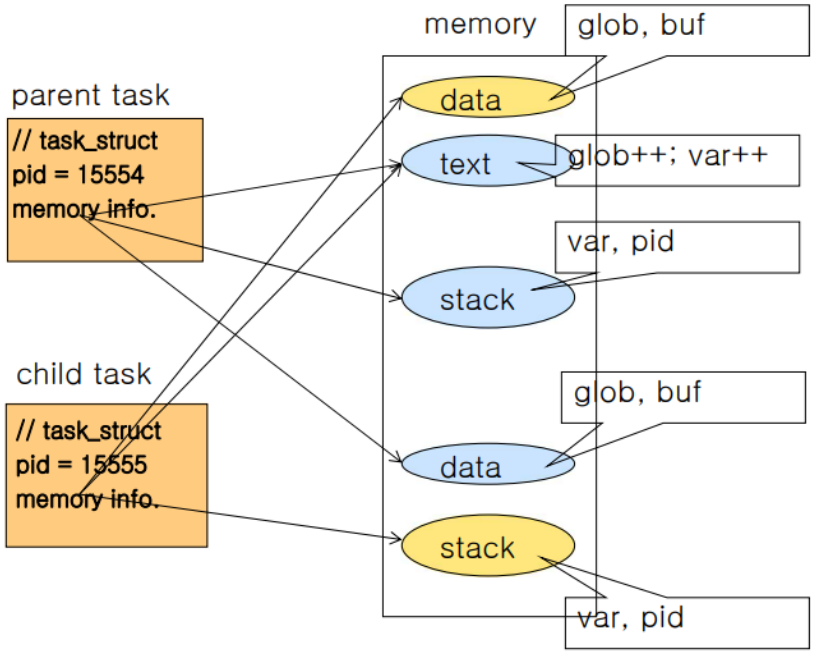
- Make a new task whose memory image (text, data, …) is the same as the existing task
  - Existing task: parent task
  - New task: child task
- Split the flow control into two (system's viewpoint)
  - One for parent and the other for child task
- Two return values (program's viewpoint)
  - Parent task: child's pid (always larger than 0)
  - Child task: 0
- wait()
  - wait for a task's termination (child or designated)



P1. fork(): Code


```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
int glob = 6;

int main()
{
    int var = 88; pid_t fork_return;

    printf("Hello, my pid is %d\n", getpid());
    printf("before fork\n"); /* we don't flush stdout */

    if ((fork_return = fork()) < 0) {
        perror("fork error");
        exit(1);
    } else if (fork_return == 0) {
        /* child process */
        glob++; var++; /* modify variables */
        printf("child: pid = %d, ppid = %d\n", getpid(), getppid());
    } else {
        /* parent process */
        wait(NULL); sleep(1);
        printf("parent: I created child with pid=%d\n", fork_return);
    }

    /* Following line is executed by both parent and child */
    printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
    printf("Bye, my pid is %d\n", getpid());
}
```





P1. fork(): Code


```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
int glob = 6;

int main()
{
    int var = 88; pid_t fork_return;

    printf("Hello, my pid is %d\n", getpid());
    printf("before fork\n"); /* we don't flush stdout */

    if ((fork_return = fork()) < 0) {
        perror("fork error");
        exit(1);
    } else if (fork_return == 0) {
        /* child process */
        glob++; var++; /* modify variables */
        printf("child: pid = %d, ppid = %d\n", getpid(), getppid());
    } else {
        /* parent process */
        wait(NULL); sleep(1);
        printf("parent: I created child with pid=%d\n", fork_return);
    }

    /* Following line is executed by both parent and child */
    printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
    printf("Bye, my pid is %d\n", getpid());
}
```

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o fork.out fork.c
mingu@server:~/TABA_OS_2023/myshell$ ./fork.out
Hello, my pid is 2911226
before fork
child: pid = 2911227, ppid = 2911226
pid = 2911227, glob = 7, var = 89
Bye, my pid is 2911227
parent: I created child with pid=2911227
pid = 2911226, glob = 6, var = 88
Bye, my pid is 2911226
mingu@server:~/TABA_OS_2023/myshell$
```

P1. fork(): Prepare & Run

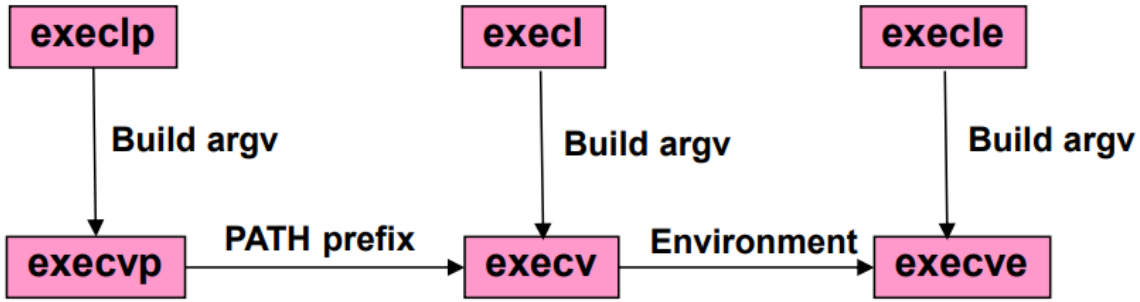
- Prepare Command
  > vim fork.c (코드 작성)

- Run Command
  > gcc -o fork.out fork.c (컴파일)
  > ./fork.out (실행)

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o fork.out fork.c
mingu@server:~/TABA_OS_2023/myshell$ ./fork.out
Hello, my pid is 2911226
before fork
child: pid = 2911227, ppid = 2911226
pid = 2911227, glob = 7, var = 89
Bye, my pid is 2911227
parent: I created child with pid=2911227
pid = 2911226, glob = 6, var = 88
Bye, my pid is 2911226
mingu@server:~/TABA_OS_2023/myshell$
```

P2. execvp()

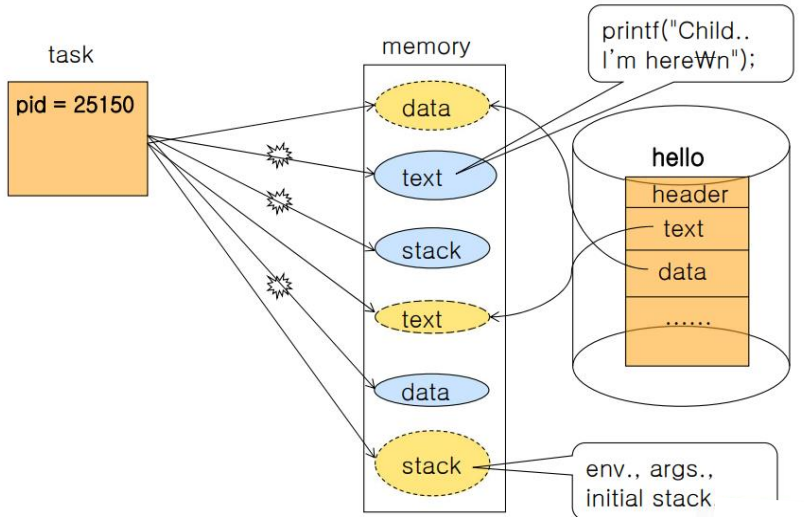
- execve() system call
  - Execute a new program
    - Replace the current task's memory image (text, data, stack) with new binary
  - System's viewpoint of execve()
    - Replace memory image (text, data, stack) with new one
    - The role of loader
  - Six interfaces
    - **int execvp(const char **file*, char **const** *argv*[]);**



P2. execvp(): Code


```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    pid_t fork_return, d_pid;
    int exit_status = -1;
    if ((fork_return = fork()) == -1)
    {
        // fork error handling
    }
    else if (fork_return == 0)
    { // child
        // NULL-terminated array of pointers for execvp
        char *args[] = {"./hello", NULL};
        execvp(args[0], args);
        printf("Child.. I'm here\n");
        // if execvp() succeeds, the above printf() is not performed!!
        exit(1);
    }
    else
    { // parent
        d_pid = wait(&exit_status);
        printf("Parent.. I'm here\n");
        printf("exit status of task %d is %d\n", d_pid, exit_status);
    }
}
```

```c
// hello.c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

int main()
{
        printf("Hello World\n");
        exit(0);
}
```



P2. execvp(): Run

- Run Command
  - > vim execvp.c (편집)
  - > gcc -o execvp.out execvp.c (컴파일)
  - >./execvp.out (실행)
  - > gcc -o hello hello.c (컴파일)
  - > ./execvp.out (실행)

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o execvp.out execvp.c
mingu@server:~/TABA_OS_2023/myshell$ ./execvp.out
Child.. I'm here
Parent.. I'm here
exit status of task 2910885 is 256
mingu@server:~/TABA_OS_2023/myshell$ gcc -o hello hello.c
mingu@server:~/TABA_OS_2023/myshell$ ./execvp.out
Hello World
Parent.. I'm here
exit status of task 2910900 is 0
mingu@server:~/TABA_OS_2023/myshell$
```