System Programming & OS 실습

8. Process

Myshell

최민국

Dankook University

mgchoi@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

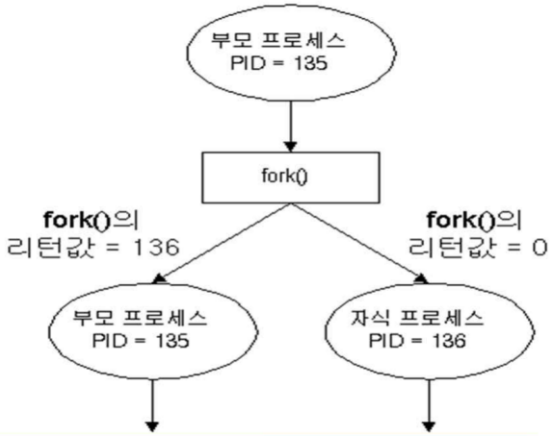
P1. fork()

- 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

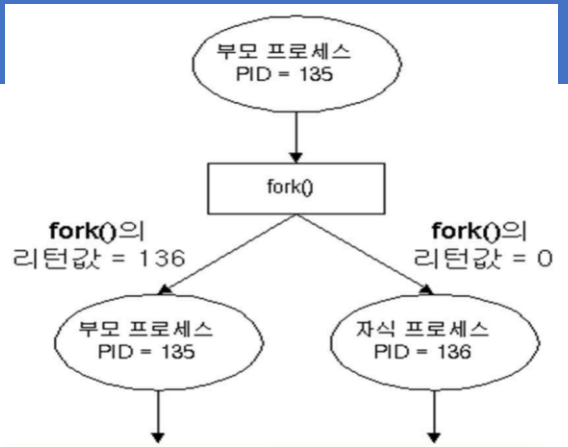
- Two return values (program's viewpoint)
  - Parent task: child's pid (always larger than 0)
  - Child task: 0

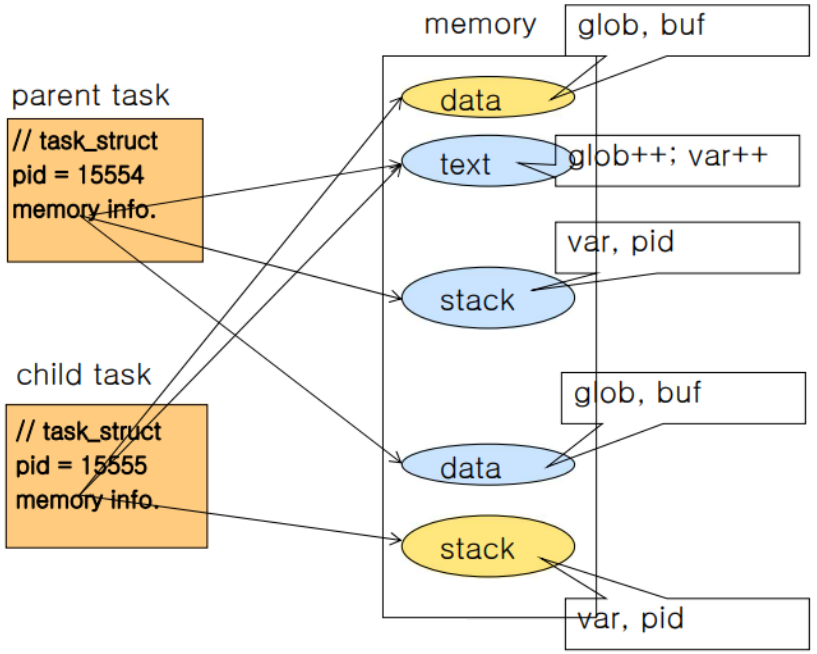
- wait()
  - wait for a task's termination (child or designated)



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)

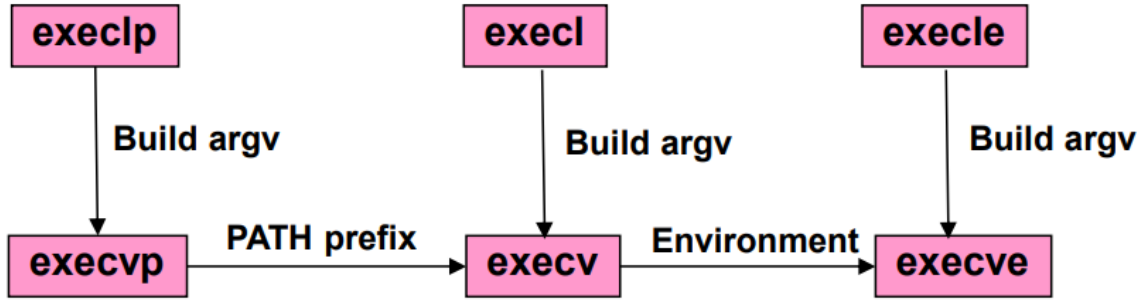
```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o fork.out fork.c
mingu@server:~/TABA_OS_2023/myshell$ ./fork.out
Hello, my pid is 2911226
before fork
child: pid = 2911227, ppid = 2911226
pid = 2911227, glob = 7, var = 89
Bye, my pid is 2911227
parent: I created child with pid=2911227
pid = 2911226, glob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

P2. execvp()

- execve() system call
  - Six interfaces
    - **int execvp(const char **file*, char *const *argv*[]);**



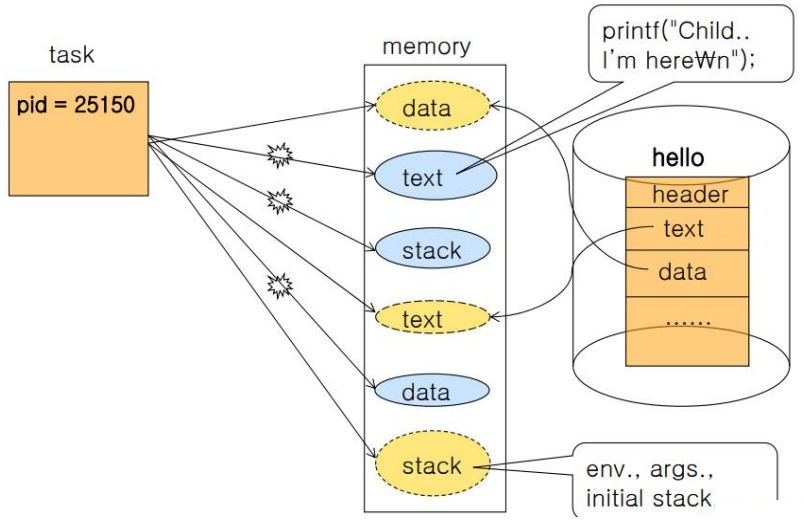
Syntax

```
int execlp(const char *filename, const char *arg0, …, const char argn, (char *) 0);
int execvp(const char *filename, char *const argv[ ]);
int execl(const char *pathname, const char *arg0, …, const char *argn, (char *) 0);
int execv(const char *pathname, char *const argv[ ]);
int execle(const char *pathname, const char *arg0, …, const char *argn, (char *) 0,
    char *const envp[ ]);
int execve(const char *pathname, char *const argv[ ], char *const envp[ ]);
```

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);
    }
}
```

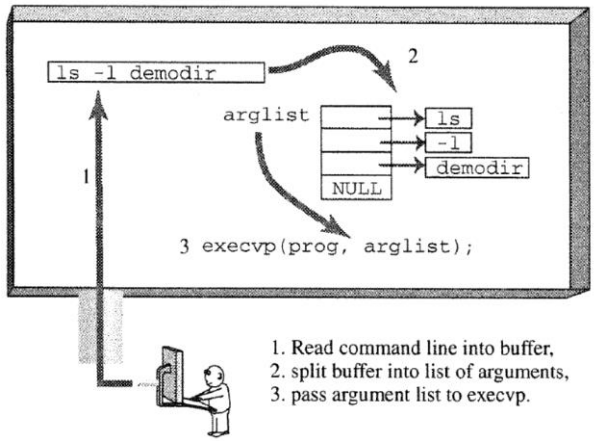
```c
// hello.c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

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

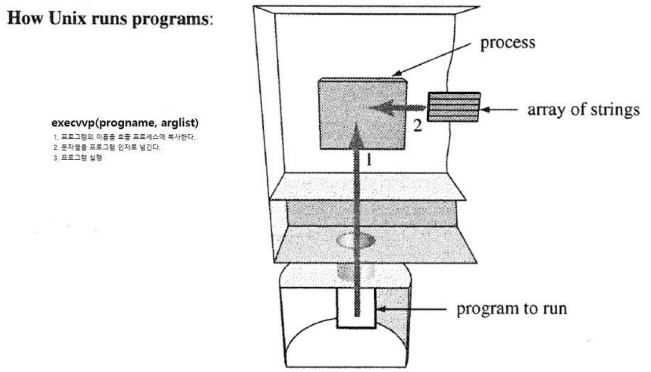


P2. execvp(): Run

- 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$
```





P3. Shell

- Command interpreter
  - Execute commands requested by users

- Basic logic
  - display prompt, input parsing
  - for external commands:
    - do fork() and execve() at child task
  - for internal commands:
    - perform in shell without fork() and execve()

- Background processing (&)
- Redirection (<, >)
- Pipe (|)

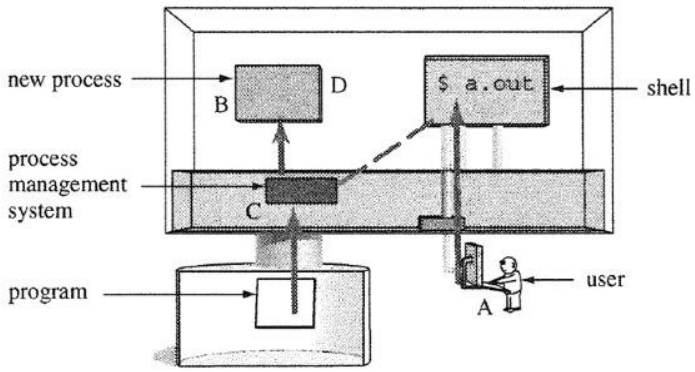


FIGURE 8.4
A user asks a shell to run a program.

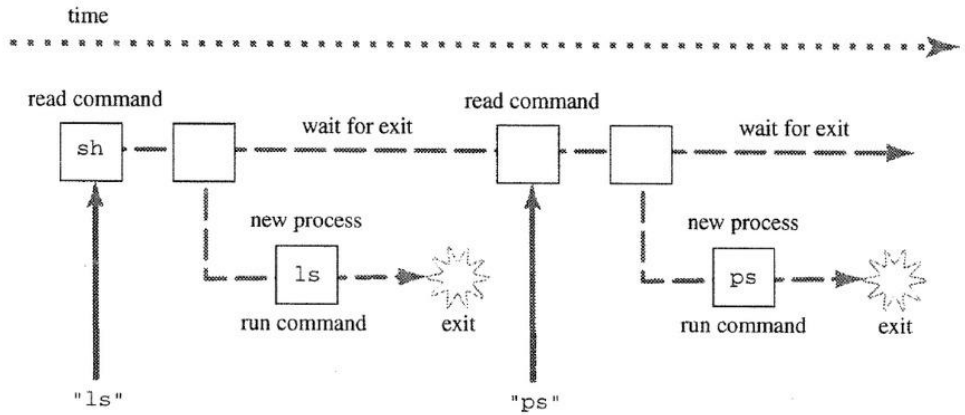


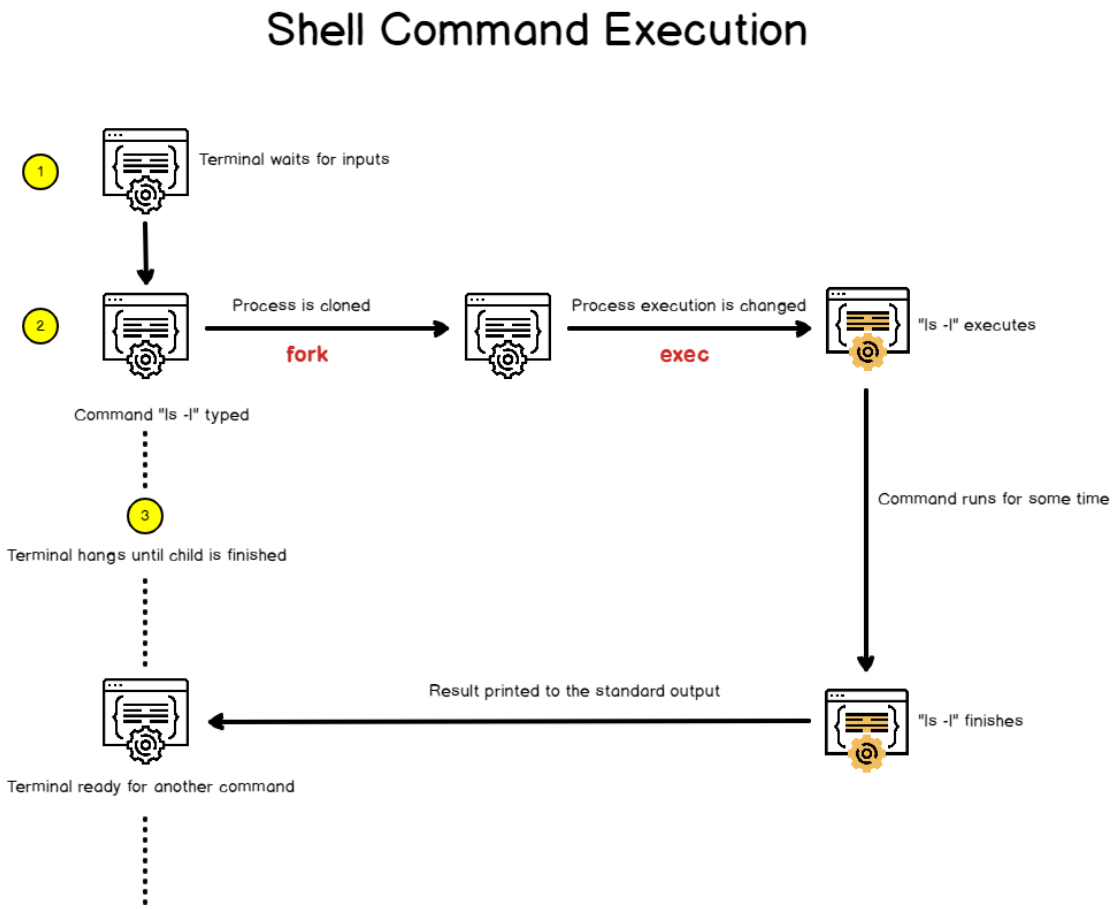
FIGURE 8.5
A time line of the main loop of the shell.

P3. Shell

- Command interpreter
  - Execute commands requested by users

- Basic logic
  - display prompt, input parsing
  - for external commands:
    - do fork() and execve() at child task
  - for internal commands:
    - perform in shell without fork() and execve()

- Background processing (&)
- Redirection (<, >)
- Pipe (|)



P3. Shell: Code


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#define MAX_BUF 1024

int main(void)
{
    char line[MAX_BUF];
    char currentDir[MAX_BUF];

    while(1){
        if (getcwd(currentDir, sizeof(currentDir)) == NULL) {
            perror("getcwd() error");
            return 0;
        }

        printf("%s $", currentDir);      // Hint. Where are you?
        fgets( line, sizeof(line) - 1, stdin);
        if(run(line) == false)
            break;
    }
    return 0;
}
```

```c
bool run(char* line) {
    char delims[] = " \r\n\t";
    char* tokens[128];    int token_count;
    int i;    int status;    pid_t child;

    token_count = 2;
    tokens[0] = "ls";
    tokens[1] = "-al";
    tokens[2] = NULL;

// Run process
    child = fork();
    if (child < 0){
        printf("Failed to fork()!");
        _exit(0);
    }

    if (/* fill the blanks.*/) {
        execvp(/* fill the blanks.*/);
        printf("No such file\n");
        _exit(0);
    } else{
        /* fill the blanks.*/
    }
    return true;
}
```

P3. Shell: Command & Result

- Run Command

  > vim myshell.c (편집)

  > gcc -o myshell.out myshell.c (컴파일)

  >./myshell.out (실행)

  $ ls -al

  $ mkdir new

  $ cat myshell.c (ctrl+c)

  > ls -al

  > cd ..

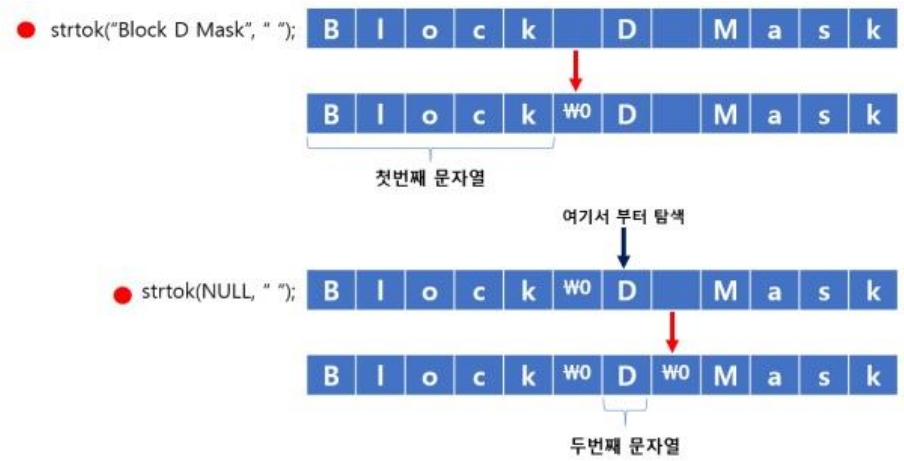
  > cd myshell

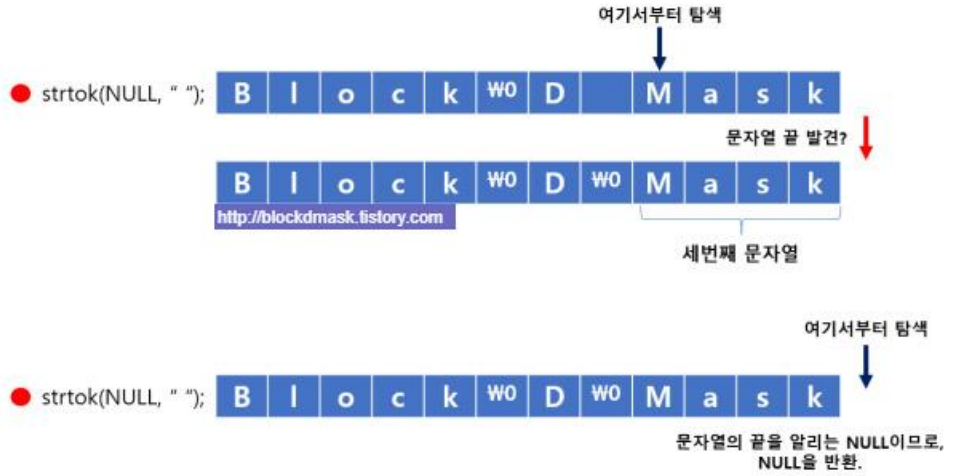
```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $ls -al
total 208
drwxrwxr-x  2 mingu mingu  4096  9월 11 23:08 .
drwxrwxr-x 10 mingu mingu  4096  9월 11 19:04 ..
-rw-rw-r--  1 mingu mingu    12  9월 11 22:08 abcde
-rw-rw-r--  1 mingu mingu   744  9월 11 16:57 execvp.c
-rwxrwxr-x  1 mingu mingu 16960  9월 11 16:57 execvp.out
-rw-rw-r--  1 mingu mingu   867  9월 10 04:09 fork.c
-rwxrwxr-x  1 mingu mingu 17072  9월 11 16:21 fork.out
-rw-rw-r--  1 mingu mingu   128  9월 10 03:54 hello.c
-rw-rw-r--  1 mingu mingu  3491  9월 11 21:12 myshell_back.c
-rw-rw-r--  1 mingu mingu  3011  9월 11 21:08 myshell_built.c
-rw-rw-r--  1 mingu mingu  1075  9월 11 23:08 myshell.c
-rwxrwxr-x  1 mingu mingu 17168  9월 11 23:08 myshell.out
-rw-rw-r--  1 mingu mingu  5849  9월 11 23:06 myshell_pipe.c
-rw-rw-r--  1 mingu mingu  4382  9월 11 22:53 myshell_redirect.c
-rw-r----x  1 mingu mingu    12  9월 11 22:10 new_abcde
-rw-rw-r--  1 mingu mingu   673  9월 11 21:37 pipe.c
-rwxrwxr-x  1 mingu mingu 17136  9월 11 22:35 pipe.out
-rw-rw-r--  1 mingu mingu   809  9월 11 22:07 redirection.c
-rw-rw-r--  1 mingu mingu   809  9월 11 22:08 redirection_copy.c
-rwxrwxr-x  1 mingu mingu 17008  9월 11 22:10 redirection.out
-rw-r--r--  1 mingu mingu 15088  9월 11 22:51 redirect.txt
-rw-rw-r--  1 mingu mingu   293  9월 10 04:38 token.c
-rwxrwxr-x  1 mingu mingu 16792  9월 10 04:38 token.out
/home/mingu/TABA_OS_2023/myshell $mkdir new
total 208
drwxrwxr-x  2 mingu mingu  4096  9월 11 23:08 .
drwxrwxr-x 10 mingu mingu  4096  9월 11 19:04 ..
-rw-rw-r--  1 mingu mingu    12  9월 11 22:08 abcde
-rw-rw-r--  1 mingu mingu   744  9월 11 16:57 execvp.c
-rwxrwxr-x  1 mingu mingu 16960  9월 11 16:57 execvp.out
-rw-rw-r--  1 mingu mingu   867  9월 10 04:09 fork.c
```

P3.1. Tokenizing

#include <string.h>

- char *strtok(char *string1, const char *string2);
  - strtok() 함수는 0개 이상의 토큰 시리즈로 string1을 읽고 string1에서 토큰 분리문자 역할을 하는 문자 세트로 string2를 읽습니다.
  - 지정된 string1에 대한 strtok() 함수의 첫 번째 호출에서 strtok() 함수는 선행 분리문자를 건너뛰고 string1에서 첫 번째 토큰을 검색합니다. 첫 번째 토큰에 대한 포인터가 리턴됩니다.
  - NULL string1 인수로 strtok() 함수가 호출된 경우 마지막 널이 아닌 string1 매개변수의 저장된 사본에서 다음 토큰을 읽습니다.
  - strtok() 함수는 버퍼로 데이터를 기록합니다.
  - strtok() 함수를 처음 호출할 때 string1에서 첫 번째 토큰에 대한 포인터를 리턴합니다. 토큰 스트링이 동일한 나중 호출에서 strtok() 함수는 스트링에서 다음 토큰에 대한 포인터를 리턴합니다. 토큰이 더 없으면 NULL 포인터가 리턴됩니다. 모든 토큰은 널로 종료됩니다.





P3.1. Tokenizing: Code


```c
#include <stdio.h>
#include <string.h>

int main(void)
{
   char *token;
   char string[] = "a string, of, ,tokens\0,after
null terminator";

   token = strtok(string, ", \n");
   do
   {
      printf("token:%s\n", token);
   }
   while (token = strtok(NULL, ", \n"));

   return 0;
}
```

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o token.out token.c
mingu@server:~/TABA_OS_2023/myshell$ ./token.out
token:a
token:string
token:of
token:tokens
mingu@server:~/TABA_OS_2023/myshell$
```

Practice 3-1: Command & Result

- Run Command

  > vim myshell.c (편집)

  > gcc -o myshell.out myshell.c (컴파일)

  >./myshell.out (실행)

  $ ls -al

  $ mkdir new

  $ rm -d new

  $ ls

  $ cd ..

  $ ^C (ctrl+c)

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $mkdir new
/home/mingu/TABA_OS_2023/myshell $ls -al
total 112
drwxrwxr-x  3 mingu mingu  4096  9월 11 20:20 .
drwxrwxr-x 10 mingu mingu  4096  9월 11 19:04 ..
-rw-rw-r--  1 mingu mingu   744  9월 11 16:57 execvp.c
-rwxrwxr-x  1 mingu mingu 16960  9월 11 16:57 execvp.out
-rw-rw-r--  1 mingu mingu   867  9월 10 04:09 fork.c
-rwxrwxr-x  1 mingu mingu 17072  9월 11 16:21 fork.out
-rw-rw-r--  1 mingu mingu   128  9월 10 03:54 hello.c
-rw-rw-r--  1 mingu mingu  1539  9월 11 19:05 myshell.c
-rwxrwxr-x  1 mingu mingu 17248  9월 11 20:20 myshell.out
drwxrwxr-x  2 mingu mingu  4096  9월 11 20:20 new
-rw-rw-r--  1 mingu mingu   293  9월 10 04:38 token.c
-rwxrwxr-x  1 mingu mingu 16792  9월 10 04:38 token.out
/home/mingu/TABA_OS_2023/myshell $rm -d new
/home/mingu/TABA_OS_2023/myshell $ls
execvp.c    fork.c    hello.c    myshell.out  token.out
execvp.out  fork.out  myshell.c  token.c
/home/mingu/TABA_OS_2023/myshell $^C
mingu@server:~/TABA_OS_2023/myshell$
```

P3.1. Tokenizing: Code


```c
// 1) Tokenize
int tokenize( char* buf, char* delims, char* tokens[], int maxTokens);
int run( char* line);

int main(void){
        // …
}

// 1) Tokenize
int tokenize(char* buf, char* delims, char* tokens[], int maxTokens) {
    int token_count = 0;
    // Hint. Token the buffer.
    char* token = /* fill the blanks.*/
    // Hint. Token buffer until ..?
    while (token != /* fill the blanks.*/ && token_count < maxTokens)
{
        // Hint. Store token in tokens.
        tokens[token_count] = /* fill the blanks.*/
        token_count++;
        // Hint. Continue tokening the buffer.
        token = /* fill the blanks.*/
    }
    // Hint. End tokens.
    tokens[token_count] = /* fill the blanks.*/
    return token_count;
}
```

```c
int run(char* line) {
    // ...
    int i;    int status;    pid_t child;

    token_count = 2;
    tokens[0] = "ls";
    tokens[1] = "-al";
    tokens[2] = NULL;

    token_count = tokenize(line, delims, tokens,
                           sizeof(tokens) / sizeof(char*));

    if (token_count == 0)
        return 1;

// Run process
    child = fork();
    // ...
}
```

P3.2. Built-in-command

4 Shell Builtin Commands

Builtin commands are contained within the shell itself. When the name of a builtin command is used as the first word of a simple command (see Simple Commands), the shell executes the command directly, without invoking another program. Builtin commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities.

cd

```
cd [-L|[-P [-e]] [-@] [directory]
```

Change the current working directory to *directory*. If *directory* is not supplied, the value of the HOME shell variable is used. If the shell variable CDPATH exists, it is used as a search path: each directory name in CDPATH is searched for *directory*, with alternative directory names in CDPATH separated by a colon (':'). If *directory* begins with a slash, CDPATH is not used.

help

```
help [-dms] [pattern]
```

Display helpful information about builtin commands. If *pattern* is specified, help gives detailed help on all commands matching *pattern*, otherwise a list of the builtins is printed.

exit

```
exit [n]
```

Exit the shell, returning a status of *n* to the shell's parent. If *n* is omitted, the exit status is that of the last command executed. Any trap on EXIT is executed before the shell terminates.

P3.2. Built-in-command

4 Shell Builtin Commands

Builtin commands are contained within the shell itself. When the name of a builtin command is used as the first word of a simple command (see Simple Commands), the shell executes the command directly, without invoking another program. Builtin commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities.

```
mingu@server:~/TABA_OS_2023/myshell$ cd ~
mingu@server:~$ cd TABA_OS_2023/myshell/
mingu@server:~/TABA_OS_2023/myshell$ cd
mingu@server:~$ cd TABA_OS_2023/myshell/
mingu@server:~/TABA_OS_2023/myshell$ help
GNU bash, version 5.0.17(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.

 job_spec [&]
 (( expression ))
 . filename [arguments]
 :
 [ arg... ]
 [[ expression ]]
 alias [-p] [name[=value] ... ]
 bg [job_spec ...]
 bind [-lpsvPSVX] [-m keymap] [-f filename] [-q name] [-u name] [-r keyseq] [-x
 break [n]
 builtin [shell-builtin [arg ...]]
 caller [expr]
 case WORD in [PATTERN [| PATTERN]...) COMMANDS ;;]... esac
 cd [-L|[-P [-e]] [-@]] [dir]
 command [-pVv] command [arg ...]
 compgen [-abcdefgjksuv] [-o option] [-A action] [-G globpat] [-W wordlist]  [-
 complete [-abcdefgjksuv] [-pr] [-DEI] [-o option] [-A action] [-G globpat] [-W
 compopt [-o|+o option] [-DEI] [name ...]
 continue [n]
 coproc [NAME] command [redirections]
```

P3.2. Built-in-command: Command & Result

- Run Command

  > vim myshell.c (편집)

  > gcc -o myshell.out myshell.c (컴파일)

  >./myshell.out (실행)

  $ cd ..
  $ cd myshell
  $ help
  $ help exit
  $ exit

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $cd ..
/home/mingu/TABA_OS_2023 $cd mysh
USAGE: cd [dir]
/home/mingu/TABA_OS_2023 $^C
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $cd ..
/home/mingu/TABA_OS_2023 $cd myshell
/home/mingu/TABA_OS_2023/myshell $help
cd        : change directory
exit      : exit this shell
quit      : quit this shell
help      : show this help
?         : show this help
/home/mingu/TABA_OS_2023/myshell $help exit
exit      : exit this shell
/home/mingu/TABA_OS_2023/myshell $exit
mingu@server:~/TABA_OS_2023/myshell$
```

P3.2. Built-in-command: Code


```c
struct COMMAND{
        char* name;
        char* desc;
        int (*func )( int argc, char* argv[]);
};

int cmd_cd(int argc, char* argv[]);     // 2) Built-in Command
int cmd_help(int argc, char* argv[]);   // 2) Built-in Command
int cmd_exit( int argc, char* argv[]); // 2) Built-in Command

struct COMMAND builtin_cmds[]={
        { "cd", "change directory", cmd_cd},
        {"exit", "exit this shell", cmd_exit },
        {"quit", "quit this shell",cmd_exit },
        {"help", "show this help",cmd_help},
        { "?","show this help", cmd_help}
};

// 1) Tokenize
int tokenize( char* buf, char* delims, char* tokens[], int maxTokens);
// 2) Built-in Command
int execute_builtin_command(char* tokens[], int token_count);
int run(char* line);

int main(void){
       // …
}
```

```c
int run(char* line) {
    pid_t child;
    int built_in = 1;

    token_count = tokenize(line, delims, tokens,
                           sizeof(tokens) / sizeof(char*));

    if (token_count == 0)
        return 1;

    if ((built_in = execute_builtin_command(tokens, token_count)) != 0)
        return built_in;

    // Run process ...
```

```c
// 2) Built-in Command
int execute_builtin_command(char* tokens[], int token_count) {
    int i;
    // Hint. 1*5=5*1=5
    for (i = 0; i < sizeof(builtin_cmds) / /* fill the blanks.*/; i++)
{
        // Hint. What your Command?
        if (strcmp(builtin_cmds[i].name, /* fill the blanks.*/) == 0)
            // Hint. argc, argv
            return builtin_cmds[i].func(/* fill the blanks.*/);
    }
    return 0;
}
```

P3.2. Built-in-command: Code

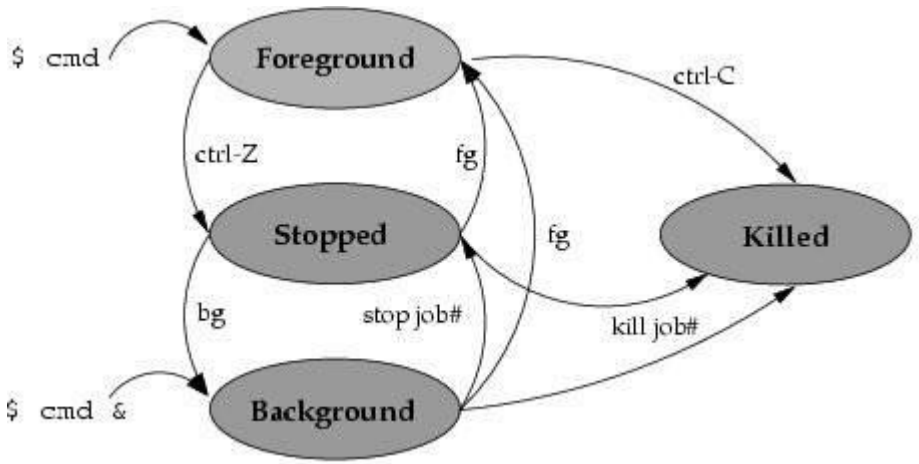

```c
// 2) Built-in Command
int cmd_help(int argc, char* argv[]) {
    int i;
    // Hint. 1*5=5*1=5
    for (i = 0; i < sizeof(builtin_cmds) / /* fill the blanks.*/; i++) {
        // Hint. Run "help" in bash shell.
        if (argc == /* fill the blanks.*/ /* choose one. (==,||,&&) */
             strcmp(builtin_cmds[i].name, /* fill the blanks.*/) == 0)
            // Hint. Go to definition
            printf("%-10s: %s\n", builtin_cmds[i]. /* fill the blanks.*/,
                    builtin_cmds[i]. /* fill the blanks.*/);
    }
    return 1;
}
```

```c
// 2) Built-in Command
int cmd_cd(int argc, char* argv[]) {
    if (argc == 1)
        // Hint. Go to home dir.
        chdir(/* fill the blanks.*/);
    else if (argc == 2) {
        // Hint. Go to ?
        if (chdir(/* fill the blanks.*/))
            printf("USAGE: cd [dir]\n");
    } else
        printf("No directory\n");
    return 1;
}
```

```c
// 2) Built-in Command
int cmd_exit( int argc, char* argv[]){
    // Hint. Return value for break while loop
    return /* fill the blanks.*/;
}
```

P3.3. Backgroud

- Background processing (&)
  - Both shell and command run concurrently
  - How to: do not use wait()




```
mingu@server:~/TABA_OS_2023/myshell$ ls /etc -al &
[1] 2923159
mingu@server:~/TABA_OS_2023/myshell$ total 1272
drwxr-xr-x 144 root root    12288  9월  5 06:25 .
drwxr-xr-x  22 root root     4096  7월 29  2022 ..
drwxr-xr-x   3 root root     4096  8월 19  2021 acpi
-rw-r--r--   1 root root     3028  8월 19  2021 adduser.conf
drwxr-xr-x   3 root root     4096  8월 19  2021 alsa
drwxr-xr-x   2 root root    12288  9월  9  2022 alternatives
-rw-r--r--   1 root root      401  7월 17  2019 anacrontab
drwxr-xr-x   3 root root     4096 12월 30  2021 apache2
-rw-r--r--   1 root root      433 10월  2  2017 apg.conf
drwxr-xr-x   5 root root     4096  8월 19  2021 apm
drwxr-xr-x   3 root root     4096  8월 19  2021 apparmor
drwxr-xr-x   7 root root     4096  8월 31 06:50 apparmor.d
drwxr-xr-x   4 root root     4096  4월 15 06:29 apport
-rw-r--r--   1 root root      769  1월 19  2020 appstream.conf
drwxr-xr-x   7 root root     4096  2월  3  2022 apt
drwxr-xr-x   3 root root     4096  6월  2 06:58 avahi
```

P3.3. Backgroud: Command & Result

- Run Command
  > vim myshell.c (편집)

  > gcc -o myshell.out myshell.c (컴파일)
  >./myshell.out (실행)

  $ ls /etc -al &
  $ quit

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $ls /etc -al &
[BG] 2923237
/home/mingu/TABA_OS_2023/myshell $total 1272
drwxr-xr-x 144 root root    12288  9월  5 06:25 .
drwxr-xr-x  22 root root     4096  7월 29  2022 ..
drwxr-xr-x   3 root root     4096  8월 19  2021 acpi
-rw-r--r--   1 root root     3028  8월 19  2021 adduser.conf
drwxr-xr-x   3 root root     4096  8월 19  2021 alsa
drwxr-xr-x   2 root root    12288  9월  9  2022 alternatives
-rw-r--r--   1 root root      401  7월 17  2019 anacrontab
drwxr-xr-x   3 root root     4096 12월 30  2021 apache2
-rw-r--r--   1 root root      433 10월  2  2017 apg.conf
drwxr-xr-x   5 root root     4096  8월 19  2021 apm
drwxr-xr-x   3 root root     4096  8월 19  2021 apparmor
drwxr-xr-x   7 root root     4096  8월 31 06:50 apparmor.d
drwxr-xr-x   4 root root     4096  4월 15 06:29 apport
-rw-r--r--   1 root root      769  1월 19  2020 appstream.conf
```

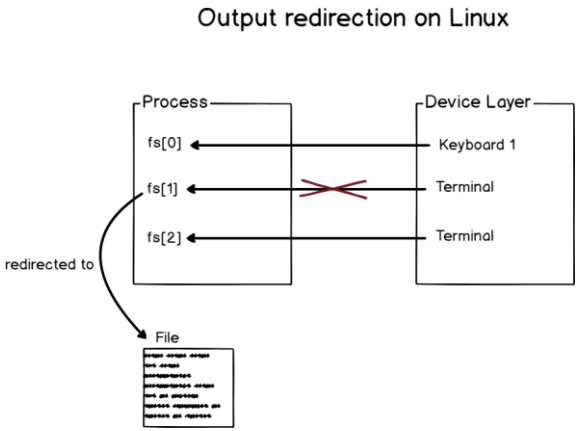
P3.3. Backgroud: Code

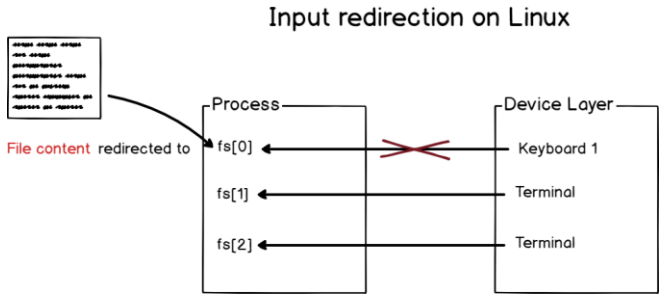

```
bool run(char* line) {
    // Init & Tokenize...
    // (3) Background
    bool is_background = handle_background(tokens, token_count);

    // Run process ...
    child = fork();
    if (child == 0) {
        execvp(tokens[0], tokens);
    } else{
        // wait(0);

        // (3) Background
        if (/* fill the blank*/) {
            while (wait(NULL) > 0);
        } else {
            // Hint. Print the child process ID
            printf("[BG] %d\n", /* fill the blank*/);
        }
    }
    return true;
}
```

```
// 3) Background
bool handle_background(char* tokens[], int token_count) {
    // Hint. Position of background token
    if (strcmp(tokens[/* fill the blanks.*/], "&") == 0) {
        // Hint. Nullify the background token
        tokens[/* fill the blanks.*/] = NULL;
        return true;
    }
    return false;
}
```

P3.4. Redirection

- Redirection (<, >)
  - read/write data from/to file instead of STDIN/STDOUT
  - how to: replace STDIN/STDOUT with file's fd using dup2() before execve()

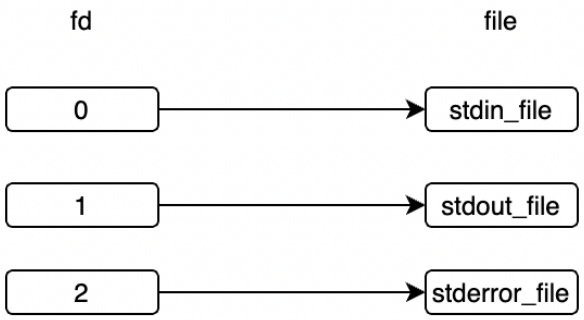


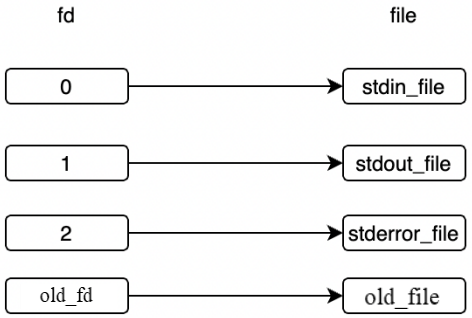



```
mingu@server:~/TABA_OS_2023/myshell$ ls
a.out  execvp.c  execvp.out  fork.c  fork.out  hello  hello.c
 myshell_ori.c  myshell.out  myshell_quiz2.c  token.c  token.out
mingu@server:~/TABA_OS_2023/myshell$ ls > ls_result.txt
mingu@server:~/TABA_OS_2023/myshell$ cat ls_result.txt
a.out
execvp.c
execvp.out
fork.c
fork.out
hello
hello.c
ls_result.txt
myshell.c
myshell_ori.c
myshell.out
myshell_quiz2.c
token.c
token.out
mingu@server:~/TABA_OS_2023/myshell$
```

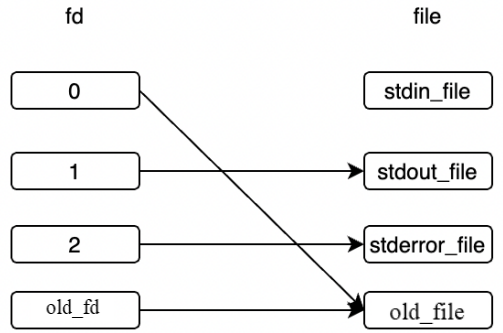
P3.4. dup2()

- dup2(int oldfd, int newfd)
  - makes newfd be the copy of oldfd, closing newfd first if necessary
    - If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed.





old_fd = open("old_file");



dup2(old_fd, STDIN_FILENO);

P3.4. dup2()


```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#define MAX_BUF 64

int main(int argc, char *argv[])
{
    int fd1, fd2, read_size, write_size;
    char buf[MAX_BUF];

    if (argc != 4) {
        printf("USAGE: %s input_file_name \">\" output_file_name\n", argv[0]);
        exit(-1);
    }

    // for redirection. (eg. "mycat inputfile.txt > outputfile.txt“)
    fd1 = open(argv[1], O_RDONLY);
    fd2 = open(argv[3], O_RDWR | O_CREAT, 0641);

    dup2(fd2, STDOUT_FILENO);
    close(fd2);

    while (1){
        read_size = read(fd1, buf, MAX_BUF);
        if (read_size == 0)
            break;
        write_size = write(STDOUT_FILENO, buf, read_size);
    }
    close(fd1);
}
```

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o redirection.out redirection.c
mingu@server:~/TABA_OS_2023/myshell$ cat abcde
abcdefghijk
mingu@server:~/TABA_OS_2023/myshell$ ls
abcde     execvp.out  fork.out  myshell_back.c   myshell.c    pipe.c          redirection_copy
.c  token.c
execvp.c  fork.c      hello.c   myshell_built.c  myshell.out  redirection.c   redirection.out
    token.out
mingu@server:~/TABA_OS_2023/myshell$ ./redirection.out abcde ">" new_abcde
mingu@server:~/TABA_OS_2023/myshell$ ls
abcde     execvp.out  fork.out  myshell_back.c   myshell.c    new_abcde  redirection.c
redirection.out  token.out
execvp.c  fork.c      hello.c   myshell_built.c  myshell.out  pipe.c     redirection_copy.c
token.c
mingu@server:~/TABA_OS_2023/myshell$ cat new_abcde
abcdefghijk
mingu@server:~/TABA_OS_2023/myshell$
```

P3.4. Redirection: Command & Result

- Run Command

  > vim myshell.c (편집)

  > gcc -o myshell.out myshell.c (컴파일)

  >./myshell.out (실행)

  $ ls /etc -al > redirect.txt

  $ cat redirect.txt

  $ quit

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $ls /etc -al > redirect.txt
/home/mingu/TABA_OS_2023/myshell $cat redirect.txt
total 1272
drwxr-xr-x 144 root root    12288  9월  5 06:25 .
drwxr-xr-x  22 root root     4096  7월 29  2022 ..
drwxr-xr-x   3 root root     4096  8월 19  2021 acpi
-rw-r--r--   1 root root     3028  8월 19  2021 adduser.conf
drwxr-xr-x   3 root root     4096  8월 19  2021 alsa
drwxr-xr-x   2 root root    12288  9월  9  2022 alternatives
-rw-r--r--   1 root root      401  7월 17  2019 anacrontab
drwxr-xr-x   3 root root     4096 12월 30  2021 apache2
-rw-r--r--   1 root root      433 10월  2  2017 apg.conf
drwxr-xr-x   5 root root     4096  8월 19  2021 apm
drwxr-xr-x   3 root root     4096  8월 19  2021 apparmor
drwxr-xr-x   7 root root     4096  8월 31 06:50 apparmor.d
drwxr-xr-x   4 root root     4096  4월 15 06:29 apport
-rw-r--r--   1 root root      769  1월 19  2020 appstream.conf
```

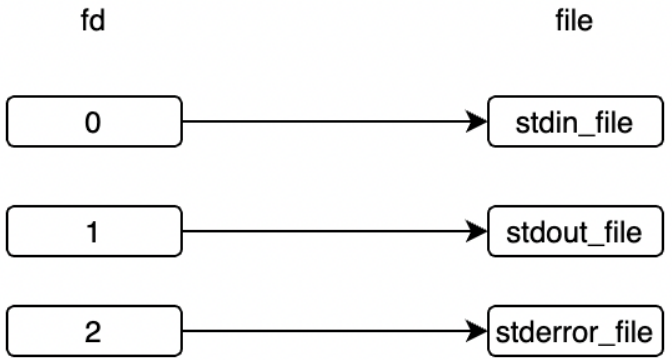
P3.4. Redirection: Code

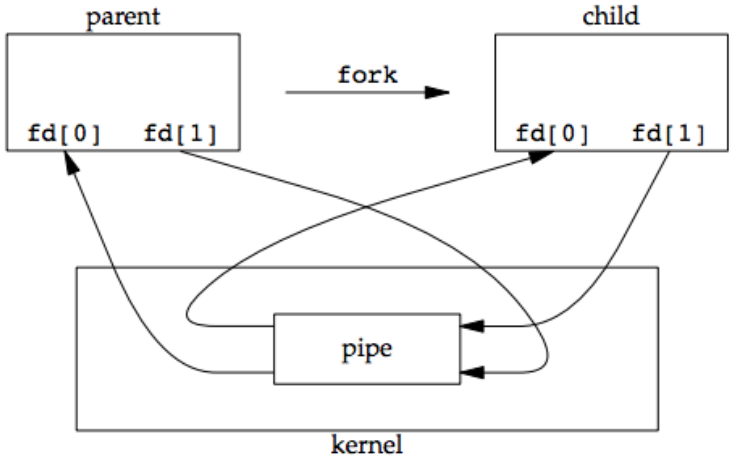

```c
int run(char* line) {
    // Init & Tokenize & Background ...

    // (4) Redirection
    int fd_out = handle_redirection(tokens, token_count);
    if (fd_out == -1) {
        return true;
    }

    // Run process ...
    child = fork();
    if (child == 0) {
        // (4) Redirection
        if (fd_out != -1) {
            dup2(/* fill the blanks.*/);
            close(fd_out);
        }

        execvp(tokens[0], tokens);
    } else{
        // Parent ...
    }
    return 1;
}
```

```c
// 4) Redirection
int handle_redirection(char* tokens[], int token_count) {
    int i;
    int fd_out = 0;

    for (i = 0; i < token_count; i++) {
        if (strcmp(tokens[i], ">") == 0) {
            // Hint. Check error msg below.
            if (i == /* fill the blanks.*/) {
                printf("Error: No redirection file specified.\n");
                return -1;
            }
            // Hint. Position of ">" and "target"
            fd_out = open(tokens[/* fill the blanks.*/], O_WRONLY |
                          O_CREAT | O_TRUNC, 0644);
            // Hint. File open error
            if (fd_out == /* fill the blanks.*/) {
                perror("Error opening file for redirection");
                return -1;
            }
            // Hint. Nullify the redirection token
            tokens[/* fill the blanks.*/] = NULL;
            break;
        }
    }
    // Hint. Return what you found.
    return /* fill the blanks.*/;
}
```

P3-5. Pipe

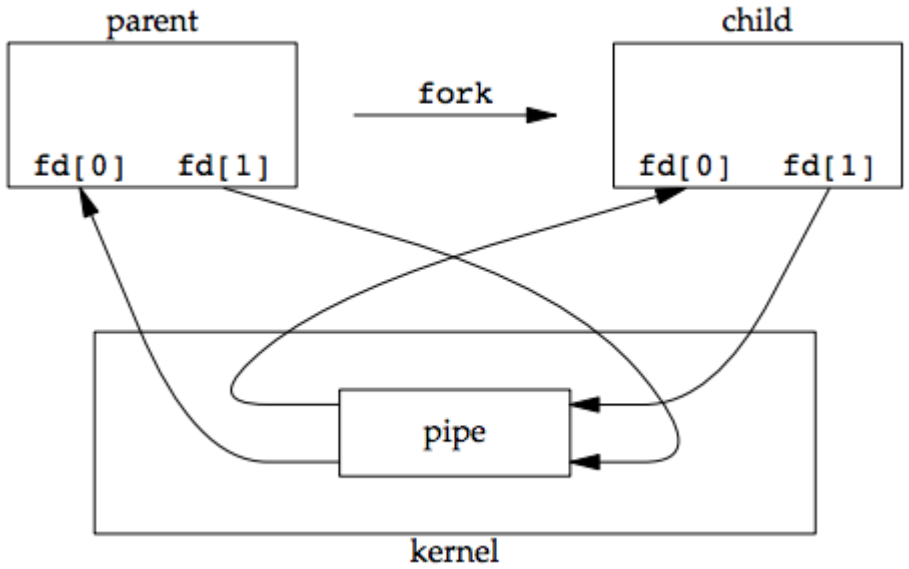
- pipe(int pipefd[2])
  - pipe() creates a pipe, a unidirectional data channel
    - used for interprocess communication.
  - The array pipefd is used to return two file descriptors referring to the ends of the pipe.
    - pipefd[0] : refers to the read end of the pipe.
    - pipefd[1] refers to the write end of the pipe.





P3-5. Pipe

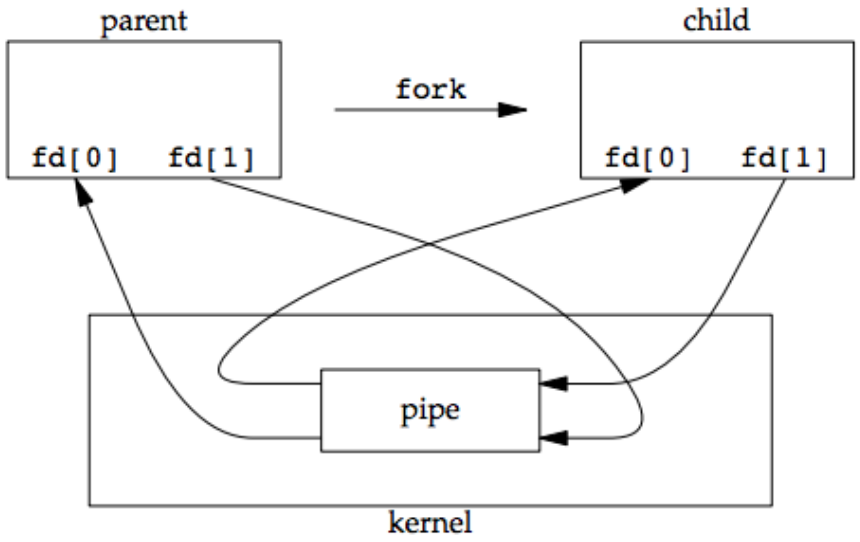

```c
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>

int main() {
    int fd[2]; char bufc[16], bufp[16]; int read_size = 0;

    pipe(fd);

    if (fork() == 0){
        write(fd[1], "Thank you", 10);
        sleep(1);
        read_size = read(fd[0], bufc, 16);
        bufc[read_size] = '\0';
        printf("%s by pid %d\n", bufc, getpid());
        exit(0);
    } else {
        read_size = read(fd[0], bufp, 16);
        bufp[read_size] = '\0';
        printf("%s by pid %d\n", bufp, getpid());
        write(fd[1], "My pleasure", 12);
        wait(0);
        close(fd[0]); close(fd[1]);
    }
}
```




```
minguᵍserver:~/TABA_OS_2023/myshell$ gcc -o pipe.out pipe.c
minguᵍserver:~/TABA_OS_2023/myshell$ ./pipe.out
Thank you by pid 2924988
My pleasure by pid 2924989
minguᵍserver:~/TABA_OS_2023/myshell$
```

P3-5. Pipe

- Pipe (|) in myshell
  - create two tasks and make them communicate via pipe
  - how to: replace STDIN/STDOUT with fd[0]/fd[1] using pipe() and dup2() before execve()




```
minguㅤ@server:~/TABA_OS_2023/myshell$ ls /etc -al | grep drwx
drwxr-xr-x 144 root root    12288  9월  5 06:25 .
drwxr-xr-x  22 root root     4096  7월 29  2022 ..
drwxr-xr-x   3 root root     4096  8월 19  2021 acpi
drwxr-xr-x   3 root root     4096  8월 19  2021 alsa
drwxr-xr-x   2 root root    12288  9월  9  2022 alternatives
drwxr-xr-x   3 root root     4096 12월 30  2021 apache2
drwxr-xr-x   5 root root     4096  8월 19  2021 apm
drwxr-xr-x   3 root root     4096  8월 19  2021 apparmor
drwxr-xr-x   7 root root     4096  8월 31 06:50 apparmor.d
drwxr-xr-x   4 root root     4096  4월 15 06:29 apport
drwxr-xr-x   7 root root     4096  2월  3  2022 apt
drwxr-xr-x   3 root root     4096  6월  2 06:58 avahi
```

grep: searches for PATTERNS in each FILE.

P3.5. Pipe: Command & Result

- Run Command

  > vim myshell.c (편집)

  > gcc -o myshell.out myshell.c (컴파일)

  >./myshell.out (실행)

  $ ls /etc -al | grep root

  $ quit

```
mingu@server:~/TABA_OS_2023/myshell$ gcc -o myshell.out myshell.c
mingu@server:~/TABA_OS_2023/myshell$ ./myshell.out
/home/mingu/TABA_OS_2023/myshell $ls /etc -al | grep drwx
drwxr-xr-x 144 root root    12288  9월  5 06:25 .
drwxr-xr-x  22 root root     4096  7월 29  2022 ..
drwxr-xr-x   3 root root     4096  8월 19  2021 acpi
drwxr-xr-x   3 root root     4096  8월 19  2021 alsa
drwxr-xr-x   2 root root    12288  9월  9  2022 alternatives
drwxr-xr-x   3 root root     4096 12월 30  2021 apache2
drwxr-xr-x   5 root root     4096  8월 19  2021 apm
drwxr-xr-x   3 root root     4096  8월 19  2021 apparmor
drwxr-xr-x   7 root root     4096  8월 31 06:50 apparmor.d
drwxr-xr-x   4 root root     4096  4월 15 06:29 apport
drwxr-xr-x   7 root root     4096  2월  3  2022 apt
drwxr-xr-x   3 root root     4096  6월  2 06:58 avahi
drwxr-xr-x   2 root root     4096  5월  2 06:15 bash_completion.d
```

P3-5. Pipe: Code


```
bool run(char* line) {
    // Init ...

    // 5) Pipe
    int pipefd[2];
    char** left_tokens = NULL;
    char** right_tokens = NULL;

    // & Tokenize & Background & Redirection ...

    //(5) Pipe
    //Hint1. Call by address(reference) or value?
    //Hint2. 포인터의 포인터의 포인터 = 문자열 배열의 포인터
    bool is_pipe = handle_pipe(tokens, token_count,
      /* fill the blanks.*/, /* fill the blanks.*/);

    if (is_pipe) {
        // Do pipe
        if (/* fill the blanks.*/ == -1) {
            perror("pipe error");
            return 1;
        }
    }

    // Run process ...
```

```
// 5) Pipe
bool handle_pipe(char* tokens[], int token_count,
                 char*** left_tokens, char*** right_tokens) {
    int i;
    for (i = 0; i < token_count; i++) {
        if (strcmp(tokens[i], "|") == 0) {
            tokens[i] = NULL;

            //Hint1. Call by address(reference) or value?
            //Hint2. 포인터의 포인터의 포인터 = 문자열 배열의 포인터
            (/* fill the blanks.*/)left_tokens = /* fill the blanks.*/
            (/* fill the blanks.*/)right_tokens = /* fill the blanks.*/
            return true;
        }
    }
    return false;
}
```

P3-5. Pipe: Code


```
    // Run process ...
    child = fork();
    if (child == 0) {
        // Redirection ...

        //(5) Pipe
        if (is_pipe) {
            close(pipefd[0]);
        // Hint. dup2() STDOUT & close previous fd
            /* fill the blanks.*/
            /* fill the blanks.*/
            execvp(left_tokens[0], left_tokens);
            perror("exec error left");
            _exit(0);
        }

        execvp(tokens[0], tokens);
    } else{
        // Parent ...
    }
    return 1;
}
```

```
    // Run process ...
    child = fork();
    if (child == 0) {
        // Child ...
    } else{
        // (5) Pipe
        if (is_pipe) {
            close(pipefd[1]);  // close write end
            child = fork();

            if (child < 0){
                printf("Failed to fork() for pipe!");
                _exit(0);
            }

            if (child == 0) {
                // Hint. dup2() STDIN & close previous fd
                /* fill the blanks.*/
                /* fill the blanks.*/
                execvp(right_tokens[0], right_tokens);
                perror("exec error right");
                _exit(0);
            }
        }

        if (!is_background) {
```

Q&A