운영체제론 실습

- UNIX 쉘 기능 구현 (History 기능 제외)

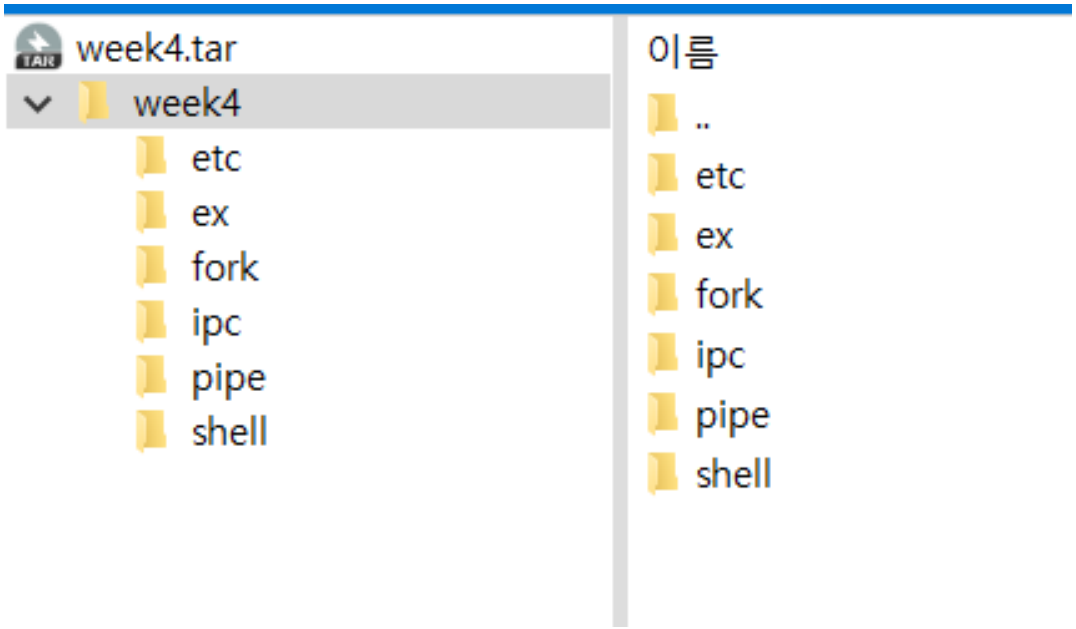
목 차

- **프로세스**
  - 프로세스 생성(pid, fork(), execv(), wait())
  - 프로세스 관련 리눅스 명령어(ps, pgrep)
- **UNIX shell 프로젝트 설명**
  - 프로젝트 설명
  - 프로젝트에 필요한 함수들(fgets, goto, string 함수)
  - 프로젝트 결과화면

예제 코드 다운로드 경로

아래 명령어를 linux 환경에서 치면 다운받을 수 있음.

$ wget http://ce.hanyang.ac.kr/week4.tar



프로세스

프로세스 생성 : fork()

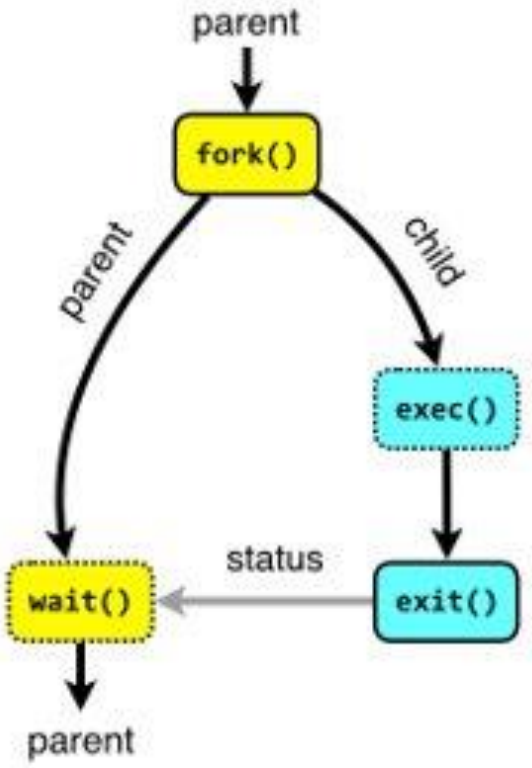
- pid_t **fork**(void) 자식 프로세스를 생성
- fork()를 기점으로 부모 프로세스로부터 자식 프로세스가 분기한다.
- 또한 fork()가 반환하는 pid 값은 다음과 같다. (실패 시, pid = -1)
  - 부모 프로세스에서 본 pid 값: 자식 프로세스의 pid
  - 자식 프로세스에서 본 pid 값: 0

Parent

```
main()        pid = 3456
{
   pid=fork();
   if (pid == 0)
      ChildProcess();
   else
      ParentProcess();
}

void  ChildProcess()
{
   .....
}

void  ParentProcess()
{
   .....
}
```

Child

```
main()        pid = 0
{
   pid=fork();
   if (pid == 0)
      ChildProcess();
   else
      ParentProcess();
}

void  ChildProcess()
{
   .....
}

void  ParentProcess()
{
   .....
}
```



프로세스 생성 : fork() 코드

예제코드 경로:

week4/fork/create_child.c

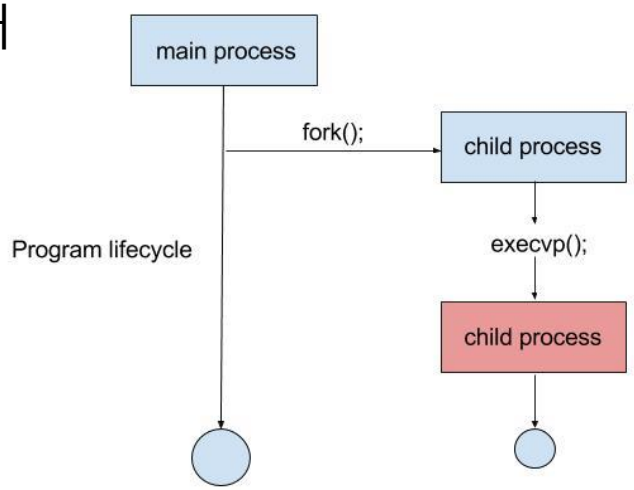
```c
/* 새로운 자식을 fork 한다. */
pid = fork();

if (pid < 0) { /* 에러가 발생한 경우 */
    fprintf(stderr, "Fork failed");
    return 1;

}else if(pid == 0) { /* 자식 프로세스 */
    printf("================================\n");
    printf("CHILD: ls command\n");
    execlp("/bin/ls","ls",NULL);

    printf("/'execlp/' call was unsuccessful\n");
    /* 해당 줄의 코드는 출력되지 않는다. 왜? */

}else { /* 부모 프로세스 */
    /* parent will wait for the child to complete */
    wait(NULL);
    /* 해당 pid에 대해서 wait하도록 다음 명령어로 대체하여 수행해본다.
     *
     * waitpid(pid, NULL, 0);
     *
     * 같은 결과를 가져오지만 항상 같은 것은 아니다.
     */
    printf("--------------------------------\n");
    printf("PARENT: Child Complete\n");
    printf("--------------------------------\n");
}
```

프로그램 실행 : execv()

예제코드 경로:

week4/fork/execv.c

- int **execv**(const char *path, char *const argv[])
- 어떤 프로그램 경로, 이름, 그리고 옵션을 인자로 받아 실행하는 함수이다.
  - 1st Argument: 실행될 프로그램의 경로/명령어
  - 2nd Argument: argv, 인자들의 모음
- 현재의 프로세스를 새로운 프로세스로 대체하는 방식으로 수행된다.



- 예제 코드 (**week4/fork/execv.c**)

```
char *const paramList[] = {"/bin/ls", "-l", ".", NULL};
execv("/bin/ls", paramList);
```

자식 프로세스의 종료를 기다림: wait()

예제코드 경로:
week4/fork/wait_child.c

- pid_t **wait**(int *status): 자식 프로세스의 종료를 기다린다.
  - 반환값: 종료된 자식 프로세스의 pid
  - **status는 자식 프로세스의 상태를 나타냄**
    - 프로세스가 끝나서 종료되는 경우와 시그널에 의해 종료되는 것을 시도해볼 것
- pid_t **waitpid**(pid_t pid, int *status, int options)
  - 해당 pid를 갖는 자식 프로세스의 종료를 기다림
  - 자식이 다수일 경우, **명시적으로 지정**하여 기다릴 수 있는 장점을 가짐
- 예제코드 (**week4/fork/wait_child.c**)

```
if (WIFEXITED(status)) { // 자식 프로세스가 정상종료되었을 때
  printf("Child exited by process completeion : %d\n", WEXITSTATUS(status));
}
if (WIFSIGNALED(status)) { // 자식 프로세스가 시그널에 의해 종료되었을 때
  printf("Child exited by signal : %d\n", WTERMSIG(status));
```

wait()의 status를 통해 프로세스의 종료 정보 불러오기

1) 외부 입력 없이 **정상적인 종료** 상태를 확인 해보자

- $ ./wait_child

2) **시그널**을 한번 보내 보도록 하자

- $ ./wait_child
- $ kill –9 {pid} // 다른 터미널을 열어서 pid 정보를 가지고 프로세스를 kill

```
jsbaik@jsbaik:~/OS2019/week4/fork$ ./wait_child
Waiting for Child process (pid:7962)
Child : 0
Child : 1
Child : 2
Child : 3
Child : 4
Child : 5
Child : 6
Child : 7
Child : 8
Child : 9
Child exits (status:768)
Child exited by process completeion : 3
jsbaik@jsbaik:~/OS2019/week4/fork$ ./wait_child
Waiting for Child process (pid:8063)
Child : 0
Child : 1
Child : 2
Child : 3
Child exits (status:9)
Child exited by signal : 9
jsbaik@jsbaik:~/OS2019/week4/fork$
```

```
파일(F)  편집(E)  보기(V)  검색(S)  터미
jsbaik@jsbaik:~$ kill -9 8063
jsbaik@jsbaik:~$
```

pstree, ps, pgrep 사용해보기

예제코드 경로:
week4/ex/num_of_process.c

- 재미있는 질문: **과연 몇 개의 프로세스가 생성되었을까요?**

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
    int i;
    for (i = 0; i < 4; i++) fork();
    return 0;
}
```

- 4번 돌았으니까 4개?
- 부모까지 해서 5개?
- 자식이 자식을 낳는 경우는?

- 프로세스의 부모자식 관계를 트리 구조로 나타내주는 도구: **pstree**

  $ ./num_of_process &

  $ pstree {pid}

- 프로세스 정보 보기 **ps**

  $ ps aux | grep num_of_process

- 프로세스 검색하기 **pgrep**

  $ pgrep num_of_process | wc –l

실행 해보고 어떤 결과가 나오는지 보도록 하자

pstree, ps, pgrep 결과화면

예제코드 경로:

week4/ex/num_of_process.c

```
jsbaik@jsbaik:~/OS2019/week4/ex$ ./num_of_process &
[1] 8665
jsbaik@jsbaik:~/OS2019/week4/ex$ pstree 8665
num_of_process─┬─num_of_process─┬─num_of_process─┬─num_of_process──num_of_process
               │                │                └─num_of_process
               │                ├─num_of_process──num_of_process
               │                └─num_of_process
               ├─num_of_process─┬─num_of_process──num_of_process
               │                └─num_of_process
               ├─num_of_process──num_of_process
               └─num_of_process
jsbaik@jsbaik:~/OS2019/week4/ex$ ps aux | grep num_of_process
jsbaik     8665  0.0  0.0   4372   816 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8666  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8667  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8668  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8669  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8670  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8671  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8672  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8673  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8674  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8675  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8676  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8677  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8678  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8679  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8680  0.0  0.0   4372    72 pts/0    S    19:59   0:00 ./num_of_process
jsbaik     8683  0.0  0.0  22820  1104 pts/0    S+   19:59   0:00 grep --color=auto num_of_process
jsbaik@jsbaik:~/OS2019/week4/ex$ pgrep num_of_process | wc -l
16
```

프로젝트: UNIX shell 기능 구현

프로젝트: UNIX shell 기능 구현

A. 사용자에게 문자열(shell 명령어)를 입력 받는다. (fgets 사용)

B. 입력받은 문자열을 "\n" 단위로 쪼갠다. (strtok 사용)

- 만약, 입력받은 문자열이 비어있다면,
  goto문을 써서 while 문 내에서 마지막으로 jump시킨다.

C. 쪼개진 문자열을 비교한다. (strcmp 사용)

- 쪼개진 문자열이 "exit"와 같다면, 프로그램을 종료시킨다.
- 쪼개진 문자열이 "&"로 마무리 된다면,
  background 프로세스로 실행하기 위해 flag를 활성화한다.

D. 현재 process의 자식 프로세스를 생성한다. (fork 사용)

- 만약 pid가 음수이면, fork 과정에 문제가 발생한 것임으로 error를 출력하고 빠져나온다.
- 만약 pid가 0이면, 자식 프로세스를 실행시킨다.
  - 자식 프로세스가 쪼개진 문자열(명령 한 줄)을 수행한다. (execvp 사용)
- 만약 pid가 0보다 크면, 부모 프로세스를 실행시킨다.
  - 만약 background flag가 활성화되어 있다면, 자식 프로세스의 종료를 기다린다. (waitpid 사용)
  - 그렇지 않다면, 부모 프로세스(프로그램)를 바로 종료시킨다.

프로젝트 관련 함수(1): fgets

예제코드 경로:

week4/etc/fgets.c

- shell 명령어를 입력 받기 위한 함수

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

#define MAX_LEN 100

int main(void)
{
    char *input = (char*)malloc(MAX_LEN*sizeof(char));
    // 고정된 공간을 활용하고 싶은 경우 다음과 같이 배열로 대체 가능:
    // char input[MAX_LEN];

    fgets(input, MAX_LEN, stdin);
    // scanf는 제약이 많기 때문에 본 예제에서는 fgets를 사용하는 것이 유리
    // 표준 입력으로부터 읽을 때 stdin 사용

    printf("INPUT: %s\n", input);
    // 출력해보면 한줄이 띄워질텐데 '\n'이라는 줄바꿈 문자를 입력받게 되서 그런 것임

    free(input);
    // 모든 처리가 완료되면, malloc을 통해 할당한 메모리를 해제해줌

    return 0;
}
```

프로젝트 관련 함수(2): goto

예제코드 경로:

week4/etc/goto.c

- 빈 입력을 했을 경우, 하위 처리 과정을 건너뛰기 위한 함수

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

int main() {
  /*
   * 랜덤한 숫자를 돌리다가 특별히 좋아하는 3의 배수가
   * 나오면 더 이상 코드를 수행하지 않고 건너뛰고
   * 무한루프를 빠져나오도록 하고 싶다.
   */
  int r;
  while (1) { /* 무한루프 */
    srand(time(NULL));
    r = rand();
    if (r % 3 == 0) {
      goto got_my_number;
    }

    /* 이 사이에는 건너뛰고 싶은 여러 코드들이 있다고 가정 */

  }
got_my_number:
  printf("My Favorite number: %d\n", r);
  return 0;
}
```

프로젝트 관련 함수(3): strtok

예제코드 경로:

week4/etc/strtok.c

- 입력받은 명령어, 옵션, 백그라운드(&) 등의 다수 입력이 있는데 이를 잘게 쪼개어 주는 함수

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

int main(void) {
  char str[] = "This is a sample string, just testing.";
  char *p;
  // strtok을 할 때마다 커서처럼 사용
  // 상위 str 문자열을 탐색하며
  // 부분적인 문자열을 출력하는데 사용

  printf("str[]=\"%s\"\n", str);

  p = strtok(str, " ");

  while (p != NULL) {
    printf("%s\n", p);
    /*
     * 출력을 받아와서 따로 복사를 해둬야 함
     * p는 일시적으로만 값을 가리키기 때문
     *
     * (1) 공간확보: 배열 혹은 메모리할당
     * (2) 일시적으로 p가 가리키고 있는 문자열을 확보한 공간에 복사:
     *
     *      strcpy() 사용
     *
     */

    p = strtok(NULL, " ,");
  }

  printf("str[]=\"%s\"\n", str);
  // strtok을 사용하면 원본 문자열을 잘라버리기 때문에 주의할 것!

  return 0;
}
```

프로젝트 관련 함수(4): strcpy

예제코드 경로:

week4/etc/strcpy.c

- 쪼개진 쉘 명령어들을 일시적으로 가리키는 것이 아니라 실제 공간을 확보하여 복사하기 위함

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

int main () {
    char src[50];
    char dest[100];

    memset(dest, '\0', sizeof(dest));
    // Garbage 값이 들어있을 수 있으니, null로 초기화를 해줌

    strcpy(src, "Operating Systems are Amazing, aren't they?");

    printf("Before strcpy:\nSRC: %s\nDEST: %s\n\n", src, dest);

    strcpy(dest, src); // 방향주의! 왼쪽으로 <- 오른쪽에서 복사

    printf("After strcpy:\nSRC: %s\nDEST: %s\n", src, dest);

    return(0);
}
```

프로젝트 관련 함수(5): strcmp

예제코드 경로:

week4/etc/strcmp.c

- 쉘 명령어가 exit, &등의 정보를 가지고 있는지 검사하기 위한 함수

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

int main() {
  char str1[20];
  char str2[20];
  int result;

  strcpy(str1, "hello");

  strcpy(str2, "hEllo");

  result = strcmp(str1, str2);

  if (result > 0) {
    printf("ASCII value of first unmatched character of str1 is greater than "
           "str2");
  } else if (result < 0) {
    printf(
        "ASCII value of first unmatched character of str1 is less than str2");
  } else if (result == 0) {
    // strcmp는 ASCII 값 비교를 하고 같으면 0을 반환한다.
    printf("Both the strings str1 and str2 are equal");
  }

  return 0;
}
```

실습: 뼈대 코드 (simple_shell.c)

예제코드 경로:

week4/shell/simple_shell.c

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

#define MAX_LEN 100 /* The maximum length command */

int main(void) {
  char *args[MAX_LEN / 2 + 1]; /* command line arguments */
  int should_run = 1;          /* flag to determine when to exit program */
  int background = 0;

  while (should_run) {
    printf("my_shell> ");
    fflush(stdout);
    /**
     * 표준입출력으로부터 문자열을 입력 받은 후:
     * (1) fork()를 통해 자식 프로세스를 생성
     * (2) 자식 프로세스가 execvp()를 호출하도록 할 것
     * (3) 만약 입력받은 문자에 &가 포함되어 있으면,
     *     부모 프로세스는 wait() 호출하지 않음
     */
  }
  return 0;
}
```

실습: UNIX shell 결과화면

예제 프로그램 경로:

week4/shell/simple_shell

```
jsbaik@jsbaik:~/OS2019/week4/shell$ ./simple_shell
my_shell>
my_shell>
my_shell>
my_shell>
my_shell> ls -l
waiting for child, not a background process
합계 24
-rw-r--r-- 1 jsbaik jsbaik   121  4월  3 20:22 Makefile
-rwxr-xr-x 1 jsbaik jsbaik 16264  4월  3 20:23 simple_shell
-rw-r--r-- 1 jsbaik jsbaik  1816  4월  3 20:23 simple_shell.c
child process complete
my_shell> ls -l &
background process
my_shell> 합계 24
-rw-r--r-- 1 jsbaik jsbaik   121  4월  3 20:22 Makefile
-rwxr-xr-x 1 jsbaik jsbaik 16264  4월  3 20:23 simple_shell
-rw-r--r-- 1 jsbaik jsbaik  1816  4월  3 20:23 simple_shell.c

my_shell>
my_shell>
my_shell> pwd
waiting for child, not a background process
/home/jsbaik/OS2019/week4/shell
child process complete
my_shell>
my_shell>
my_shell> exit
jsbaik@jsbaik:~/OS2019/week4/shell$
```

수고하셨습니다.

- 다음 시간: **프로세스 목록 나타내는 모듈 프로그래밍**