

운영체제및실습 01분반 실습 3회차

프로세스 시스템 콜

2021.03.31(수)

Contents

1. 리눅스 명령어(2)

- 1) Wildcards
- 2) Link (ln)
- 3) 알아두면 유익한 대부분의) (명령어 공통 옵션 (쉬어가기))
- 4) Redirection

2. System Calls for Process Management

- 1) fork
- 2) wait
- 3) exec
- 4) open

3. 과제

- 1) fork
- 2) wait
- 3) exec
- 4) open

리눅스 명령어(2)

1. 도입
2. 도우미
3. 들여다보기
4. 길 찾기
5. 루트 디렉토리
6. 파일 및 디렉토리 조작하기

리눅스 명령어(2)

■ Wildcards

: 파일을 집단적으로 처리하기 위해 사용하는, 특수한 의미를 갖는 기호

- ◆ * : 모든 문자들
 - * : 모든 파일
 - j* : j로 시작하는 모든 파일
 - *.txt : .txt로 끝나는 모든 파일
- ◆ ? : 아무거나 딱 한 문자
 - bab? : bab로 시작하는 4글자 파일
- ◆ [characters] : 작성한 문자 집합에 존재하는 문자 하나
 - [abc123]* : [abc123] 중 하나로 시작하는 모든 파일
- ◆ ![characters] : 작성한 문자 집합에 존재하지 않는 문자 하나
 - ![abcd]abcd : [abcd]로 시작하지 않는 5글자 파일

리눅스 명령어(2)

■ Wildcards

◆ `[:class:]` : 지정된 클래스에 속하는 문자 하나

◆ `class`

- `alnum` : 알파벳, 숫자
- `alpha` : 알파벳
- `digit` : 숫자
- `lower` : 소문자
- `upper` : 대문자

◆ 예제

- `abc[:alnum:].txt`
- `[:lower:]*.c`
- `[:alpha:][:digit:][:digit:][:digit:].txt`
- `*[:upper:]abc`

리눅스 명령어[2]

■ Link (ln)

◆ ln : 링크 생성

- ln file link : **hard link** 생성 (directory X)
- ln -s item link : **symbolic link** 생성 (item = file or directory)

◆ 하드 링크 Hard Links (original)

: 같은 데이터를 참조하는, 이름만 다른 파일

- 모든 파일은 최소 하나의 하드 링크가 존재함
- 링크와 동일한 디스크 파티션에 없는 파일 참조 불가
- 디렉토리 참조 불가
- 파일 자체와 구분 불가 (링크 write=파일 write), 특수한 링크 표시 없음
- 삭제할 경우 링크만 삭제되지만, 마지막 링크가 삭제되면 파일도 삭제됨

◆ 심볼릭 링크 Symbolic Links (modern, MS 윈도우 '바로 가기'와 유사)

: 특정한 파일 및 디렉토리에 대한 text pointer

- 파일 자체와 구분 불가, 타입 기호(l) 및 색상 표현 (ls 또는 file 명령어로 확인)
- 링크의 참조 파일이 삭제되더라도 링크는 남아있음 (broken 상태)

리눅스 명령어(2)

■ 알아두면 유익한 (대부분의) 명령어 공통 옵션 (쉬어가기)

- ◆ -v (--verbose) : 수행된 작업의 내용을 출력
- ◆ -i (--interactive) : 명령어 수행 전에 사용자에게 응답 요청
 - 덮어쓰기 할 경우
 - 프로그램을 설치할 경우 ...
- ◆ -f (--force) : 강제로 수행
 - 위 -i와 유사한 상황에서 prompt 없이 수행
 - ✓ mv, cp 등에서는 -f가 기본 적용 (기본 덮어쓰기)
 - ✓ ln 등의 명령어에서는 덮어쓰기를 시도할 경우, 옵션이 없으면 에러 메시지만 출력하고 수행되지 않지만, -f 옵션을 주면 파일을 덮어 써버린다.

리눅스 명령어(2)

■ Redirection

◆ 표준 입력/표준 출력/표준 에러 (stdin/stdout/stderr)

: 기본적인 입,출,에러 메시지 (파일)

- 커맨드에 대한 입력은 표준 입력에 작성됨
 - ✓ 키보드를 누르면 stdin 파일에 작성되고 화면에 출력
- 커맨드에 대한 출력은 표준 출력 또는 표준 에러에 작성됨
 - ✓ 둘 다 화면으로 연결
 - ✓ 결과적으로 화면에 보여지지만, 실제 작성되는 파일은 서로 다름(stdout/stderr)
 - ✓ 기본적으로 디스크에 저장되지 않음

◆ Redirection이란?

: stdin/stdout/stderr가 전달되는 목적지를 재정의

리눅스 명령어(2)

■ Redirection

◆ 표준 출력 리디렉션

- `ls -l /bin > ls-output.txt` (표준 출력 리디렉션 수행, 새로 쓰기)
- `ls -l ls-output.txt` (파일 크기 확인)
- `vi editor` 또는 `less` 명령어로 해당 파일 내용 확인

- `ls -l /babo > ls-output.txt` (표준 출력 리디렉션 수행, 새로 쓰기)
- `ls -l ls-output.txt` (파일 크기 확인)
- `vi editor` 또는 `less` 명령어로 해당 파일 내용 확인

- 첫 번째 수행의 결과로, 화면에 출력될 결과물이 파일에 쓰여짐
- 두 번째 수행의 결과로, 화면에 출력될 결과물이 파일에 쓰여짐
 - ✓ 에러 메시지는 `stderr`이므로 `stdout`과 별개
 - ✓ 실제 `stdout` 출력이 없으므로, 덮어쓰기 되어서 내용물이 사라짐!

- `ls-output.txt`에 아무 내용 작성
- `> ls-output.txt` (표준 출력 리디렉션 수행)
 - ✓ 출력이 없으므로 위의 두 번째 수행과 유사 (에러 메시지만 없다는 차이)

리눅스 명령어(2)

■ Redirection

◆ 표준 출력 리디렉션 (계속)

- `ls -l /bin >> ls-output.txt` (표준 출력 리디렉션 수행, 이어 쓰기 append)
- 할 때마다 파일 크기가 늘어나는 것 확인 가능

◆ 표준 에러 리디렉션

- `ls -l /babo 2> ls-error.txt` (표준 에러 리디렉션 수행, 새로 쓰기)
- `ls -l ls-error.txt` (파일 크기 확인)
- vi editor 또는 less 명령어로 해당 파일 내용 확인
- 파일을 관리(참조)하기 위한 파일 번호를 file descriptor라고 부름
- 표준 입/출/에러의 fd는 순서대로 각각 0/1/2

◆ 표준 입력 리디렉션

- 키보드 대신에 파일의 내용을 통해서 입력을 주고 싶은 경우

리눅스 명령어(2)

■ Redirection

◆ `cat [file...]` : 파일 연결하여 출력

- `cat ls-output.txt`
- `cat text01.txt text02.txt text03.txt`
- `cat movie.mpeg.0* > movie.mpeg`
 - ✓ 분할된 영상을 결합시킬 때
- `cat`
 - ✓ 표준 입력을 표준 출력으로
 - ✓ `ctrl+d` (EOF, end-of-file) 입력시 입력 종료
- `cat > babo.txt`
 - ✓ 표준 입력 > 표준 출력 > 파일 (= 입력한 내용이 파일에 쓰임, 새로 쓰기)
- `cat babo.txt`
- `cat < babo.txt` (표준 입력 리디렉션 수행, 출처가 키보드에서 파일로 전환)
 - ✓ 키보드 입력 대신 파일 내용이 표준 입력으로 적용 > 표준 출력
 - ✓ 결과적으로 위와 같은 출력

리눅스 명령어(2)

■ Redirection

◆ Pipelines

: 한 명령어의 출력을 다른 명령어의 입력으로 연결

- **command1 | command2** (| : shift + back slash)

✓ ls -l | less

◆ Filters

- **wc** : word count, 줄 수, 단어 수, 바이트 수 출력
- **sort** : 정렬
- **uniq** : 중복 제거 (-d 옵션 주면, 반대로 중복만 출력)
 - ✓ ls -l /bin /usr/local/bin | sort | uniq | wc
- **grep pattern [file...]** : 패턴과 일치하는 줄 출력 (나중에 더 자세하게)
 - ✓ ls /bin /usr/local/bin | uniq | grep gcc
- **head/tail** : 파일 처음부터/끝부터 출력
 - ✓ ls /bin | head -n 5
 - ✓ tail -f /var/log/messages : 시스템 로그 모니터링
- **tee** : 표준 입력을 읽어서 표준 출력과 파일로 출력
 - ✓ ls /bin | tee ls-bin.txt | grep zip => cat ls-bin.txt (화면과 파일에 동시 출력 확인)

SYSTEM CALLS FOR PROCESS MANAGEMENT

1. fork
2. wait
3. exec
4. open

System Calls for Process Management

■ 예제 코드 공유

◆ git clone https://github.com/JoonheeJeong/cnu_os_prac.git

System Calls for Process Management

■ `pid_t fork(void);`

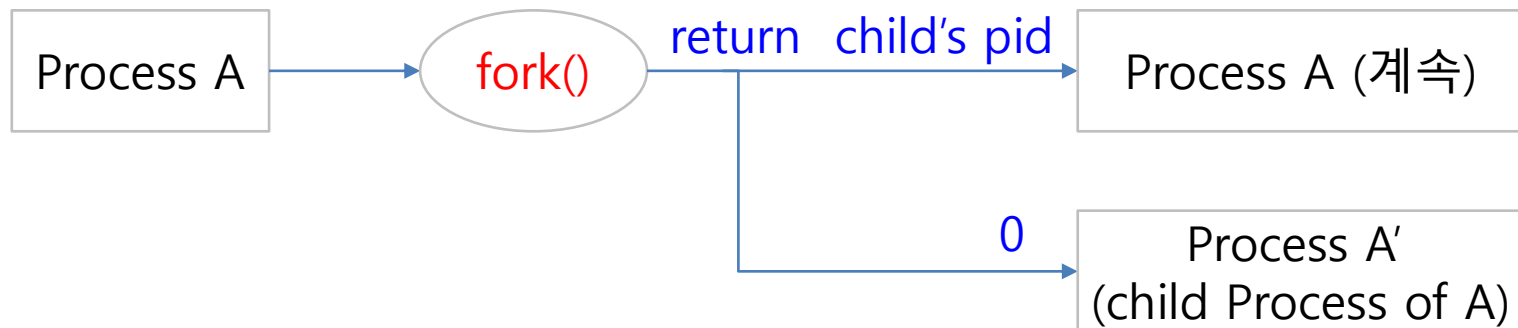
: 프로세스 '복사' 생성 시스템 콜

◆ 부모 프로세스

- `fork()`를 호출하는 프로세스
- 자식 프로세스의 pid를 return 받음

◆ 자식 프로세스

- `fork()`를 호출 결과 생성되는 프로세스
- 0을 return 받음
- (참고) return이 0보다 작으면 error
- `fork()` 호출 후 두 프로세스의 실행 순서는 보장되지 않음!



System Calls for Process Management

■ fork 예제

```
1 #include <stdio.h>
2 #include <stdlib.h> // exit
3 #include <unistd.h> // fork, getpid
4
5 int main(int argc, char *argv[])
6 {
7     printf("hello world (pid:%d)\n", (int) getpid());
8     int ret_fork = fork();
9     if (ret_fork < 0) {
10         // fork failed; exit
11         fprintf(stderr, "fork failed\n");
12         exit(1);
13     } else if (ret_fork == 0) {
14         // child (new process)
15         printf("hello, I am child (pid:%d)\n", (int) getpid());
16     } else {
17         // parent goes down this path (original process)
18         printf("hello, I am parent of %d (pid:%d)\n",
19             ret_fork, (int) getpid());
20     }
21     return 0;
22 }
```

"p1.c" 22 lines --95%--

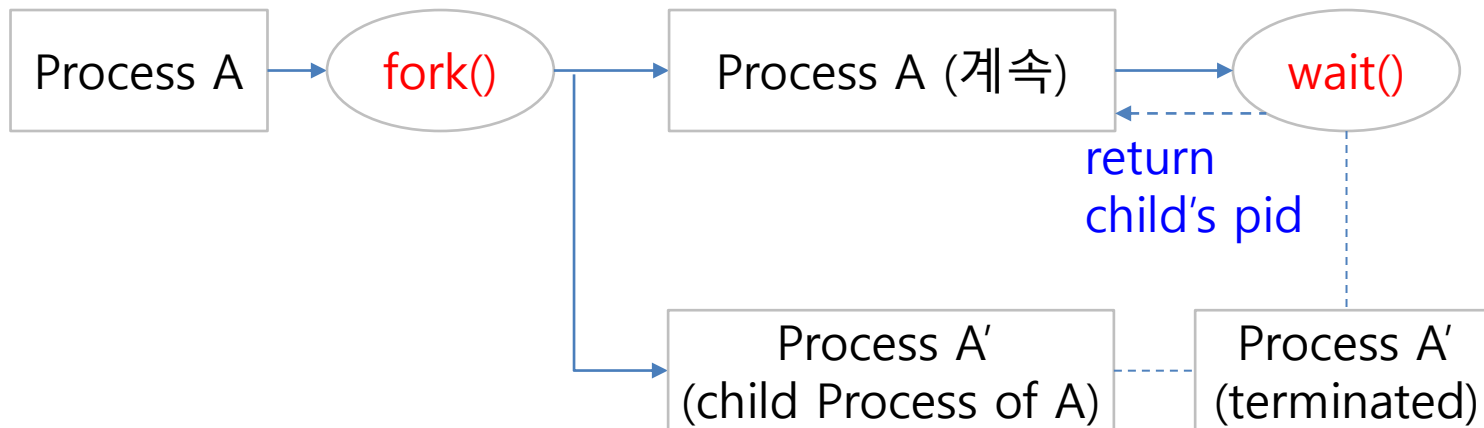
```
joonhee@joonhee-laptop:~/os_prac/03$ ./p1
hello world (pid:3589)
hello, I am parent of 3590 (pid:3589)
hello, I am child (pid:3590)
```


System Calls for Process Management

■ `pid_t wait(int *wstatus);`

: 자식 프로세스 종료 대기 시스템 콜

- 임의의 자식 하나 프로세스 종료 대기
- `wait()`이 호출되기 전까지는 'zombie' 상태로, PCB가 남아있음
- (참고) `pid_t waitpid(pid_t pid, int *wstatus, int options);`



System Calls for Process Management

■ wait 예제

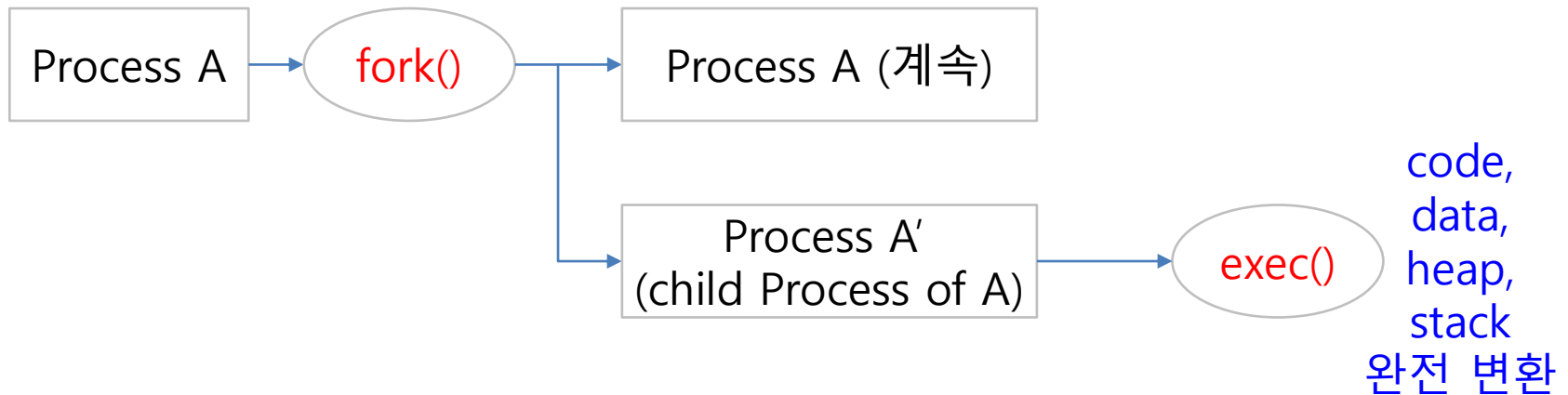
```
1 #include <stdio.h>
2 #include <stdlib.h>      // exit
3 #include <unistd.h>      // getpid, fork, sleep
4 #include <sys/wait.h>
5
6 int main(int argc, char *argv[])
7 {
8     printf("hello world (pid:%d)\n", (int) getpid());
9     int ret_fork = fork();
10    if (ret_fork < 0) {
11        // fork failed; exit
12        fprintf(stderr, "fork failed\n");
13        exit(1);
14    } else if (ret_fork == 0) {
15        // child (new process)
16        printf("hello, I am child (pid:%d)\n", (int) getpid());
17        sleep(1); // sleep for 1 second
18    } else {
19        // parent goes down this path (original process)
20        int ret_wait = wait(NULL);
21        printf("hello, I am parent of %d (ret_wait:%d) (pid:%d)\n",
22               ret_fork, ret_wait, (int) getpid());
23    }
24    return 0;
25 }
```

"p2.c" 25 lines --100%--

```
joonhee@joonhee-laptop:~/os_prac/03$ ./p2
hello world (pid:3753)
hello, I am child (pid:3754)
hello, I am parent of 3754 (ret_wait:3754) (pid:3753)
```

System Calls for Process Management

- `int exec(const char *file, char *const argv[]);`
: 존재하는 프로세스의 context에서 다른 실행 파일을 실행 (덮어 씌움)
 - `execl, execlp, execl, execv, execvp, execvpe`
 - 참고) [https://en.wikipedia.org/wiki/Exec_\(system_call\)](https://en.wikipedia.org/wiki/Exec_(system_call))



◆ fork-exec 왜 쓸까?

- unix의 shell 운용에 적합한 형태 (shell 수행 중 커맨드 실행)

System Calls for Process Management

■ exec 예제

```
1 #include <stdio.h>
2 #include <stdlib.h>      // exit
3 #include <unistd.h>      // getpid, fork, execvp
4 #include <string.h>      // strdup
5 #include <sys/wait.h>
6
7 int main(int argc, char *argv[])
8 {
9     printf("hello world (pid:%d)\n", (int) getpid());
10    int ret_fork = fork();
11    if (ret_fork < 0) {
12        // fork failed; exit
13        fprintf(stderr, "fork failed\n");
14        exit(1);
15    } else if (ret_fork == 0) {
16        // child (new process)
17        printf("hello, I am child (pid:%d)\n", (int) getpid());
18        char *myargs[3];
19        myargs[0] = strdup("wc"); // program: "wc" (word count)
20        myargs[1] = strdup("p3.c"); // argument: file to count
21        myargs[2] = NULL; // marks end of array
22        execvp(myargs[0], myargs); // runs word count
23        printf("this shouldn't print out");
24    } else {
25        // parent goes down this path (original process)
26        int ret_wait = wait(NULL);
27        printf("hello, I am parent of %d (ret_wait:%d) (pid:%d)\n",
28               ret_fork, ret_wait, (int) getpid());
29    }
30    return 0;
31 }
```

joonhee@joonhee-laptop:~/os_prac/03\$./p3

hello world (pid:4352)

hello, I am child (pid:4353)

31 131 1062 p3.c

hello, I am parent of 4353 (ret_wait:4353) (pid:4352)

~

"p3.c" 31 lines --19%--

System Calls for Process Management

- `int open(const char *pathname, char *flags, mode_t mode);`
: 파일 열기

- open 예제

- ◆ `O_CREAT`
: 없으면 생성
- ◆ `O_WRONLY`
: 쓰기 전용
- ◆ `O_TRUNC`
: 덮어쓰기
- ◆ `S_IRWXU`
: User 권한
(읽기/쓰기/실행)

```
1 #include <stdio.h>
2 #include <stdlib.h>      // exit
3 #include <unistd.h>      // fork, close
4 #include <string.h>      // strdup
5 #include <fcntl.h>       // open(flags, modes)
6 #include <sys/wait.h>
7
8 int main(int argc, char *argv[])
9 {
10     int ret_fork = fork();
11     if (ret_fork < 0) {
12         // fork failed; exit
13         fprintf(stderr, "fork failed\n");
14         exit(1);
15     } else if (ret_fork == 0) {
16         // child: redirect standard output to a file
17         close(STDOUT_FILENO);
18         open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
19         // now exec "wc"...
20         char *myargs[3];
21         myargs[0] = strdup("wc"); // program: "wc" (word count)
22         myargs[1] = strdup("p4.c"); // argument: file to count
23         myargs[2] = NULL; // marks end of array
24         execvp(myargs[0], myargs); // runs word count
25     } else {
26         // parent goes down this path (original process)
27         wait(NULL);
28     }
29     return 0;
30 }
```

"p4.c" 30 lines --23%--

과제

1. fork
2. wait
3. exec
4. open

과제



■ 1. fork()를 호출하는 프로그램 작성

- ◆ fork() 호출 전에 지역변수와 전역변수를 하나씩 만들고 초기화. 그 후 부모 프로세스와 자식 프로세스 각각에서 각 변수의 값이 무엇인지, 하나의 프로세스에서 값을 변경할 시 다른 프로세스에서 해당 변수의 값이 바뀌는지 확인하기

■ 2. wait()을 호출하는 프로그램 작성

- ◆ wait()을 호출하여 자식 프로세스에서는 "안녕", 부모 프로세스에서는 "잘 가"를 출력하되, 자식 프로세스가 먼저 수행되도록 작성하기
- ◆ 부모 프로세스에서 wait() 호출 이외에 다른 방법을 2가지를 이용해서 위의 결과와 동일한 프로그램을 작성하고, 각각의 차이는 무엇인지 설명하기 (총 3개의 소스코드 : wait1~3_학번, e.g. wait3_201212345)

■ 3. exec()을 호출하는 프로그램 작성

- ◆ fork() 호출 후 exec() family 중 적절하게 하나를 호출하여 현재 작업 디렉토리에 대한 /bin/lis 명령어를 수행하기
- ◆ 왜 이렇게 많은 종류의 exec()이 있는지 그 이유를 보고서에 작성

■ 4. open()과 fork()를 호출하는 프로그램 작성

- ◆ 부모와 자식 프로세스에서 같은 파일 스크립터에 동시에 접근하면 어떻게 되는지 작성해서 확인하기 (open 파라미터는 p4 예제와 동일하게)



과제



■ 포함 내용

◆ 소스코드(각 번호당 1개, wait은 3개)

- fork_학번, exec_학번, open_학번
- wait{1-3}_학번
- 코드 카피 주의

◆ 보고서

- 소스코드 이미지, 각각의 실행결과 이미지, 결과 분석 및 설명


■ 제출기한 및 양식

◆ ~4월 6일(화) 23:59

◆ OS01_03_학번_이름.zip

- src 폴더 이하 소스코드
- 보고서 pdf 같은 이름





Q&A

Thank you!!!