9장 프로세스 제어

리눅스 시스템 프로그래밍

청주대학교 전자공학과
한철수

목차

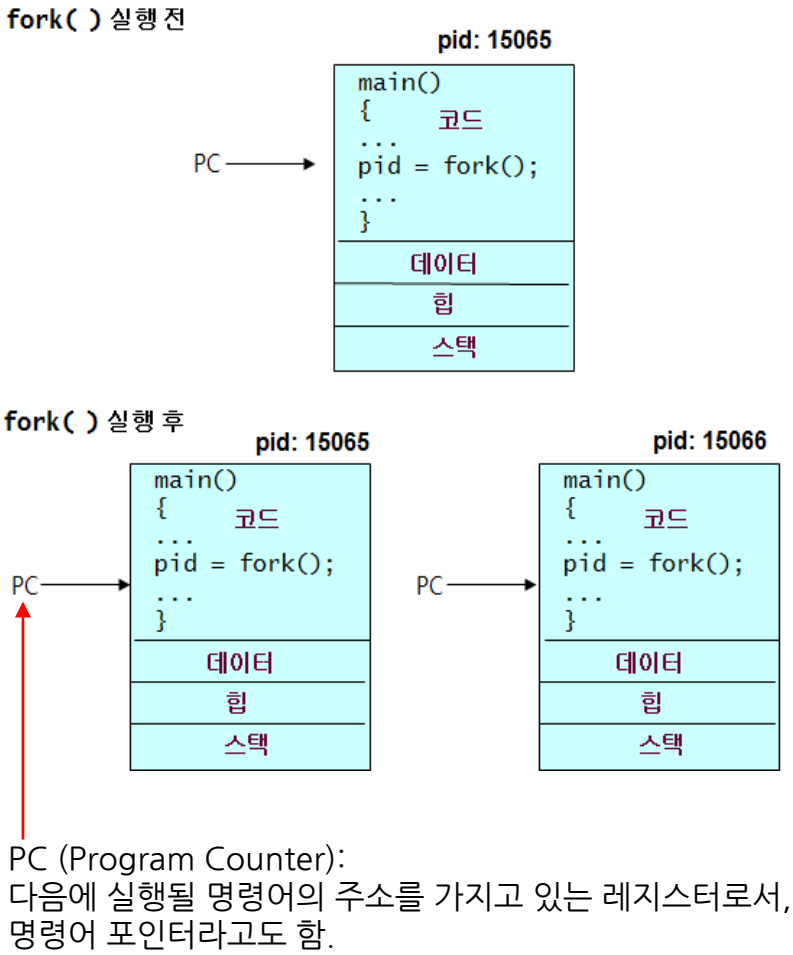
- 프로세스 생성
- 프로그램 실행
- 입출력 재지정
- 프로세스 그룹
- 시스템 부팅

프로세스 생성

- fork() 시스템 호출
  - 새로운 프로그램을 실행하기 위해서는 먼저 새로운 프로세스를 생성해야 하는데, fork() 시스템 호출이 새로운 프로세스를 생성하는 유일한 방법임.
- 함수 프로토타입
  - pid_t fork(void);
    - 새로운 자식 프로세스를 생성함.
    - 자식 프로세스에서는 0을 반환하고, 부모 프로세스에서는 자식 프로세스의 PID를 반환함. (매우 중요)

fork() 시스템 호출 실행 전과 후

- fork() 시스템 호출은 부모 프로세스를 똑같이 복제하여 자식 프로세스를 생성함.
  - 자기 복제
- fork() 시스템 호출이 실행되면 자식 프로세스가 즉시 생성됨.
- 그 후, 부모 프로세스와 자식 프로세스는 각각 독립적으로 실행을 계속함.



fork1.c

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

int main()
{
    printf("[%d] 프로세스 시작₩n", getpid());
    int pid = fork();
    printf("[%d] 프로세스: 반환 값 %d₩n", getpid(), pid);
    return 0;
}
```

부모 프로세스가 실행. →
부모 프로세스가 실행. →
자식 프로세스가 실행. →

```
cju@cju-VirtualBox:~/chap9$ gcc -o fork1 fork1.c
cju@cju-VirtualBox:~/chap9$ ./fork1
[3814] 프로세스 시작
[3814] 프로세스: 반환 값 3815
[3815] 프로세스: 반환 값 0
cju@cju-VirtualBox:~/chap9$
```

fork() 시스템 호출의 반환값 처리

- fork() 시스템 호출 후에 실행되는 코드들은 부모 프로세스와 자식 프로세스에서 각각 독립적으로 모두 실행됨.
- 그러면 부모 프로세스와 자식 프로세스가 서로 다른 일을 하게 하려면 어떻게 해야 할까?
  - 반환 값을 이용함.
  - 왜냐하면 fork() 시스템 호출은 자식 프로세스에서는 0을 반환하고, 부모 프로세스에서는 자식 프로세스의 PID를 반환하기 때문에.
- 서로 다른 일을 하도록 처리하는 코드

```
pid = fork();
if( pid == 0 ){
    자식 프로세스가 실행할 코드;
}
else {
    부모 프로세스가 실행할 코드;
}
```

fork2.c

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

int main()
{
    int pid = fork();
    if(pid==0) {
        printf("[Child] : Hello, world pid=%d₩n", getpid());
    }
    else {
        printf("[Parent] : Hello, world pid=%d₩n", getpid());
    }
}
```

← 자식 프로세스가 실행.

← 부모 프로세스가 실행.

```
cju@cju-VirtualBox:~/chap9$ gcc -o fork2 fork2.c
cju@cju-VirtualBox:~/chap9$ ./fork2
[Parent] : Hello, world pid=3860
[Child] : Hello, world pid=3861
cju@cju-VirtualBox:~/chap9$
```

fork3.c

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

int main()
{
    int pid1 = fork();
    if(pid1==0) {
        printf("[Child 1] : Hello, world ! pid=%d₩n", getpid());
        exit(0);
    }
    int pid2 = fork();
    if(pid2==0) {
        printf("[Child 2] : Hello, world ! pid=%d₩n", getpid());
        exit(0);
    }
    printf("[Parent] : Hello, world ! pid=%d₩n", getpid());
}
```

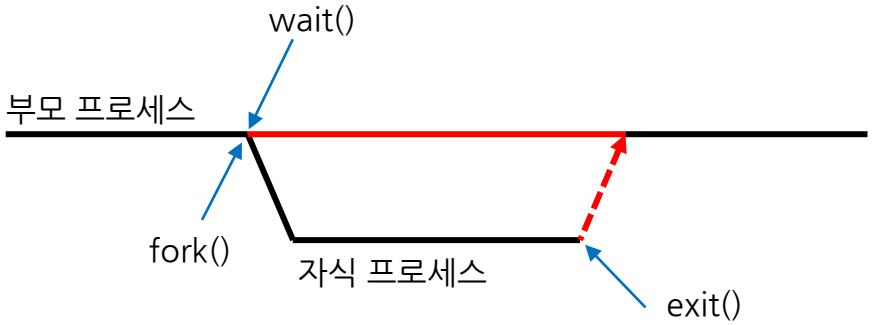
첫 번째 자식 프로세스가 종료됨.

두 번째 자식 프로세스가 종료됨.

```
cju@cju-VirtualBox:~/chap9$ gcc -o fork3 fork3.c
cju@cju-VirtualBox:~/chap9$ ./fork3
[Parent] : Hello, world ! pid=3888
[Child 1] : Hello, world ! pid=3889
[Child 2] : Hello, world ! pid=3890
cju@cju-VirtualBox:~/chap9$
```

wait() 시스템 호출

- 부모 프로세스는 wait() 시스템 호출을 실행하여 자식 프로세스 중의 하나가 종료될 때까지 기다릴 수 있음.



- 함수 프로토타입
  - pid_t wait(int *status);
    - 실행하면 자식 프로세스 중 하나가 종료될 때까지 기다림.
    - 자식 프로세스가 종료될 때의 종료 코드가 *status에 저장됨.
    - 종료된 자식 프로세스의 PID를 반환함.

forkwait.c

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    printf("[%d] 부모 프로세스 시작₩n", getpid());
    int pid = fork();
    if(pid == 0) {
        printf("[%d] 자식 프로세스 시작₩n", getpid());
        exit(1);
    }
    int status=0;
    int child = wait(&status);
    printf("[%d] 자식 프로세스 %d 종료₩n", getpid(), child);
    printf("₩t종료 코드 %d₩n", status>>8);
}
```

```
cju@cju-VirtualBox:~/chap9$ gcc -o forkwait forkwait.c
cju@cju-VirtualBox:~/chap9$ ./forkwait
[3910] 부모 프로세스 시작
[3911] 자식 프로세스 시작
[3910] 자식 프로세스 3911 종료
        종료 코드 1
cju@cju-VirtualBox:~/chap9$
```

exit(1); ← 자식 프로세스가 종료됨. 종료 코드: 1

int child = wait(&status); ← 자식 프로세스가 종료되기를 기다림.

waitpid() 시스템 호출

- 부모 프로세스는 waitpid() 시스템 호출을 실행하여 자식 프로세스 중 하나를 지정하여 그것이 종료될 때까지 기다릴 수 있음.
- 함수 프로토타입
  - pid_t waitpid(pid_t pid, int *status, int options);
    - PID가 pid인 자식 프로세스가 종료될 때까지 기다림.
    - 자식 프로세스가 종료될 때의 종료 코드가 *status에 저장됨.
    - options는 부모 프로세스의 대기 방법을 나타내며, 특정한 대기 방법을 지정하지 않으려면 0을 사용함.
    - 함수는 종료된 자식 프로세스의 PID를 반환함.

waitpid.c

```c
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    printf("[%d] 부모 프로세스 시작 ₩n", getpid( ));
    int pid1 = fork();
    if(pid1 == 0) {
        printf("[%d] 자식 프로세스[1] 시작 ₩n", getpid( ));
        sleep(1);
        printf("[%d] 자식 프로세스[1] 종료 ₩n", getpid( ));
        exit(1);
    }
    int pid2 = fork();
    if(pid2 == 0) {
        printf("[%d] 자식 프로세스 #2 시작 ₩n", getpid( ));
        sleep(2);
        printf("[%d] 자식 프로세스 #2 종료 ₩n", getpid( ));
        exit(2);
    }
    int status=0;
    int child = waitpid(pid1, &status, 0);
    printf("[%d] 자식 프로세스 #1 %d 종료 ₩n", getpid( ), child);
    printf("₩t종료 코드 %d₩n", status>>8);
}
```

```
cju@cju-VirtualBox:~/chap9$ ./waitpid
[3935] 부모 프로세스 시작
[3936] 자식 프로세스[1] 시작
[3937] 자식 프로세스 #2 시작
[3936] 자식 프로세스[1] 종료
[3935] 자식 프로세스 #1 3936 종료
        종료 코드 1
cju@cju-VirtualBox:~/chap9$ [3937] 자식 프로세스 #2 종료

cju@cju-VirtualBox:~/chap9$
```

질문

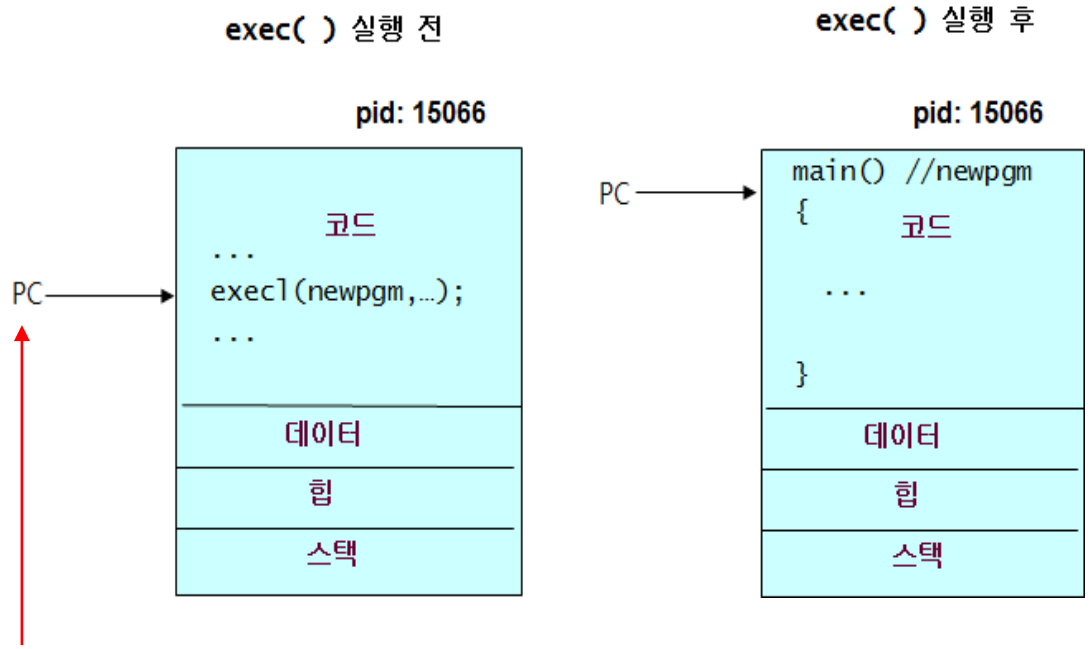
Q&A

프로그램 실행

- 부모 프로세스가 fork() 시스템 호출을 통해 자식 프로세스를 생성하면, 자식 프로세스는 부모 프로세스와 동일한 코드를 실행함.
  - 부모 프로세스와 자식 프로세스에서 fork() 시스템 호출의 반환 값이 서로 다르다는 점을 이용하여, 부모 프로세스와 자식 프로세스가 서로 다른 일을 하도록 하였음.
- 그러면 자식 프로세스에게 새로운 프로그램을 실행 시키기 위해서는 어떻게 해야 할까요?
  - exec() 시스템 호출을 실행하여 프로세스는 새로운 프로그램을 실행시킬 수 있음.

exec() 시스템 호출 실행 전과 후

- 프로세스가 exec() 시스템 호출을 실행하면, 그 프로세스 내의 프로세스 이미지(코드, 데이터, 힙, 스택 등)는 새로운 프로세스 이미지로 대치됨(바뀜).
  - 자기 대치
  - 단, PID는 바뀌지 않음.



PC (Program Counter):
다음에 실행될 명령어의 주소를 가지고 있는 레지스터로서,
명령어 포인터라고도 함.

exec() 시스템 호출의 사용 예

- exec() 시스템 호출이 성공하면 기존 프로세스 이미지는 새로운 프로세스 이미지로 대치됨.
  - exec() 시스템 호출이 성공한 경우에는 반환 값이 반환되지 않고, 실패한 경우에만 반환 값(-1)이 반환됨.
- fork() 시스템 호출을 실행한 후에 exec() 시스템 호출을 실행하는 것이 일반적임.

```
if((pid=fork())==0){
    exec(arguments);
    exit(1);
}
// 부모 프로세스는 아래를 계속 실행
```

자식 프로세스에게 실행시킬 프로그램을 arguments로 전달함.

exec() 시스템 호출이 실패한 경우에만 실행됨.

exec() 시스템 호출

- 함수 프로토타입
  - int execl(char* path, char* arg0, char* arg1, ... , NULL);
  - int execv(char* path, char* argv[ ]);
  - int execlp(char* file, char* arg0, char* arg1, ... , NULL);
  - int execvp(char* file, char* argv[ ]);
    - 프로세스 이미지를 path(또는 file)가 가리키는 새로운 프로그램의 프로세스 이미지로 대치한 후 새 코드를 실행함.
    - exec( ) 시스템 호출이 성공하면 반환 값을 반환하지 않고, 실패하면 -1을 반환함.
- 실행 예
  - execl("/bin/echo", "echo", "hello", NULL);
  - execv(argv[1], &argv[1]);

execute1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main( )
{
    printf("부모 프로세스 시작₩n");
    if(fork( ) == 0) {
        execl("/bin/echo", "echo", "hello", NULL);
        fprintf(stderr,"첫 번째 실패");
        exit(1);
    }
    printf("부모 프로세스 끝₩n");
}
```

자식 프로세스의 프로그램이 echo hello로 교체됨.

exec()가 실패한 경우에만 실행됨.

```
cju@cju-VirtualBox:~/chap9$ gcc -o execute1 execute1.c
cju@cju-VirtualBox:~/chap9$ ./execute1
부모 프로세스 시작
부모 프로세스 끝
cju@cju-VirtualBox:~/chap9$ hello


cju@cju-VirtualBox:~/chap9$
```

execute2.c

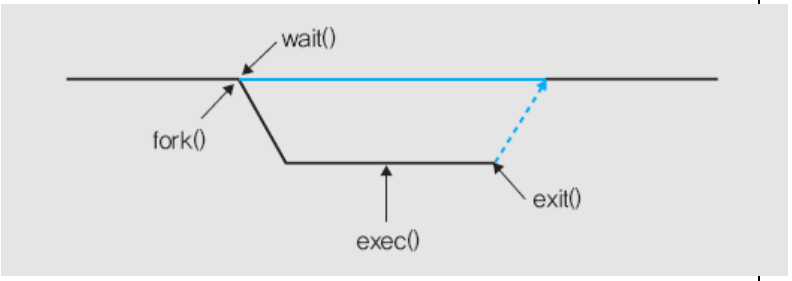
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
   printf("부모 프로세스 시작₩n");
   if(fork() == 0) {
      execl("/bin/echo", "echo", "hello", NULL);
      fprintf(stderr,"첫 번째 실패");
      exit(1);
   }
   if(fork() == 0) {
      execl("/bin/date", "date", NULL);
      fprintf(stderr,"두 번째 실패");
      exit(2);
   }
   if(fork() == 0) {
      execl("/bin/ls","ls", "-l", NULL);
      fprintf(stderr,"세 번째 실패");
      exit(3);
   }
   printf("부모 프로세스 끝₩n");
}
```

```
cju@cju-VirtualBox:~/chap9$ gcc -o execute2 execute2.c
cju@cju-VirtualBox:~/chap9$ ./execute2
부모 프로세스 시작
부모 프로세스 끝
cju@cju-VirtualBox:~/chap9$ hello
2023. 05. 24. (수) 21:16:37 KST
합계 140
-rwxrwxr-x 1 cju cju 16176  5월 24 21:12 execute1
-rw-rw-r-- 1 cju cju   305  5월 24 21:12 execute1.c
-rwxrwxr-x 1 cju cju 16176  5월 24 21:16 execute2
-rw-rw-r-- 1 cju cju   571  5월 24 21:14 execute2.c
-rwxrwxr-x 1 cju cju 16048  5월 18 03:09 fork1
-rw-rw-r-- 1 cju cju   209  5월 18 03:09 fork1.c
```

execute3.c

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
    int pid = fork( );
    if(pid == 0) {
        execvp(argv[1], &argv[1]);
        fprintf(stderr, "%s:실행 불가₩n",argv[1]);
    }
    else {
        int status=0;
        int child = wait(&status);
        printf("[%d] 자식 프로세스 %d 종료 ₩n", getpid(), pid);
        printf("₩t종료 코드 %d ₩n", status>>8);
    }
}
```

```
cju@cju-VirtualBox:~/chap9$ gcc -o execute3 execute3.c
cju@cju-VirtualBox:~/chap9$ ./execute3 echo hi
hi
[21053] 자식 프로세스 21054 종료
        종료 코드 0
cju@cju-VirtualBox:~/chap9$
```



system() 함수

- system() 함수는 C 라이브러리 함수임.
  - system() 함수는 자식 프로세스를 생성하고 자식 프로세스로 하여금 명령어를 실행시키는 것을 한번에 수행해 줌.
  - system() 함수는 지금까지 다룬 fork(), exec(), waitpid() 시스템 호출을 이용하여 구현되었음.
- 함수 프로토타입
  - int system(const char *cmdstring);
    - 이 함수는 /bin/sh -c cmdstring를 호출하여 cmdstring에 지정된 명령어를 실행함.
    - 명령어가 끝난 후, 명령어의 종료코드를 반환함.
- 사용 예

```
system("ls -asl");
```

syscall.c

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int status=0;
    if((status = system("date")) < 0)
        perror("system() 오류");
    printf("종료코드 %d₩n", WEXITSTATUS(status));

    if((status = system("hello")) < 0)
        perror("system() 오류");
    printf("종료코드 %d₩n", WEXITSTATUS(status));

    if((status = system("who; exit 44")) < 0)
        perror("system() 오류");
    printf("종료코드 %d₩n", WEXITSTATUS(status));
}
```

system() 함수는 명령어의 종료 코드를 반환함.

WEXITSTATUS(status)는 앞에서 나온 status>>8 과 같음.

```
cju@cju-VirtualBox:~/chap9$ gcc -o syscall syscall.c
cju@cju-VirtualBox:~/chap9$ ./syscall
2023. 05. 24. (수) 21:20:35 KST
종료코드 0
sh: 1: hello: not found
종료코드 127
cju      tty2         2023-05-24 20:46 (tty2)
종료코드 44
cju@cju-VirtualBox:~/chap9$
```

system() 함수의 구현

```
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
int mysystem(const char *cmdstring)
{
    if(cmdstring == NULL)
        return 1;
    int pid=fork();
    if(pid == -1)
        return -1;
    if(pid == 0) {
        execl("/bin/sh", "sh", "-c", cmdstring, NULL);
        _exit(127);
    }
    int status=0;
    do {
        if(waitpid(pid, &status, 0) == -1){
            if(errno != EINTR)
                return -1;
        }
        else
            return status;
    } while(1);
}
```

자식 프로세스가 cmdstring을 실행

부모 프로세스의 기다림이 비정상적으로 풀렸다면

errno 변수를 확인해 최근 발생한 오류가 인터럽트에 의한 에러(EINTR)가 아니라면 함수를 빠져나가고, 맞으면 waitpid 함수를 다시 시도함

질문

Q&A

입출력 재지정

- 쉘에서의 입출력 재지정
  $ 명령어 > 텍스트 파일
  $ ls -l > list.txt
- dup() 혹은 dup2() 시스템 호출을 이용하여 입출력 재지정을 구현할 수 있음.
- 함수 프로토타입
  - int dup(int oldfd);
    - oldfd에 대한 복제본인 새로운 파일 디스크립터를 생성하여 반환함.
    - 실패하면 -1을 반환함.
  - int dup2(int oldfd, int newfd);
    - oldfd을 newfd에 복제하고, 복제된 파일 디스크립터를 반환함.
    - 실패하면 -1을 반환함.

실행 예

- int dup(int oldfd);
- int dup2(int oldfd, int newfd);
  - 파일 디스크립터 oldfd와 복제된 파일 디스크립터는 같은 파일을 공유하게 된다.
- 예

```
FILE *fp=fopen(argv[1], "w");
dup2(fp->fd, 1);
puts("hello");
```

1번 파일 디스크립터는 표준출력(STDOUT)의 파일 디스크립터임.

"hello"가 파일에 저장됨.

redirect1.c

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    FILE *fp=fopen(argv[1], "w");

    dup2(fileno(fp), 1);
    fclose(fp);
    printf("Hello stdout !₩n");
    fprintf(stderr, "Hello stderr !₩n");
}
```

열린 파일 fp의 파일 디스크립터를 반환함.

fp의 파일 디스크립터를 1에 복사함. 기본적으로 1은 표준출력(stdout)의 파일 디스크립터인데, 이제 1은 fp를 가리키게 되었음.

```
cju@cju-VirtualBox:~/chap9$ gcc -o redirect1 redirect1.c
cju@cju-VirtualBox:~/chap9$ ./redirect1 out
Hello stderr !
cju@cju-VirtualBox:~/chap9$ cat out
Hello stdout !
cju@cju-VirtualBox:~/chap9$
```

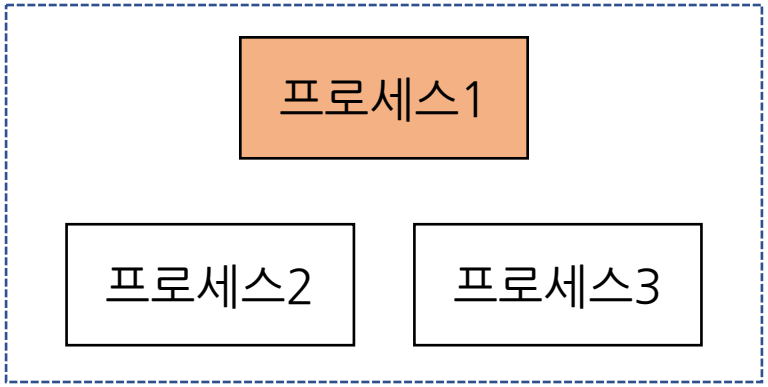
redirect2.c

```
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
    int pid=fork();
    if(pid==0){
        FILE *fp=fopen(argv[1], "w");
        dup2(fileno(fp), 1);
        fclose(fp);
        execvp(argv[2], &argv[2]);
        fprintf(stderr, "%s: 실행 불가₩n", argv[1]);
    } else {
        int status=0;
        int child=wait(&status);
        printf("[%d] 자식 프로세스 %d 종료₩n", getpid(), child);
    }
}
```

```
cju@cju-VirtualBox:~/chap9$ gcc -o redirect2 redirect2.c
cju@cju-VirtualBox:~/chap9$ man date > you.txt
cju@cju-VirtualBox:~/chap9$ ./redirect2 out wc you.txt
[22559] 자식 프로세스 22560 종료
cju@cju-VirtualBox:~/chap9$ cat out
 209  820 6516 you.txt
cju@cju-VirtualBox:~/chap9$
```

프로세스 그룹

- 프로세스 그룹(process group)은 여러 프로세스들의 집합을 의미함.
  - 각 프로세스는 하나의 프로세스 그룹에 속하게 됨.
  - 각 프로세스는 PID 뿐만 아니라 자신이 속한 **프로세스 그룹 ID(PGID)**도 가짐.
- 부모 프로세스가 fork를 통해 생성한 자식 프로세스들은 부모와 하나의 프로세스 그룹에 속하게 됨.
  - 각 프로세스 그룹에는 그 그룹을 만든 프로세스 그룹 리더가 있음.
  - 프로세스 그룹 리더의 PID는 **프로세스 그룹 ID(PGID)**와 같음.



프로세스 그룹

getpgrp() 시스템 호출

- 프로세스 그룹 ID(PGID)는 getpgrp() 시스템 호출을 통해 알 수 있음.
- 함수 프로토타입
  - pid_t getpgrp(void);
    - 호출한 프로세스의 프로세스 그룹 ID를 반환함.

pgrp1.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("[PARENT] PID=%d GID=%d₩n", getpid(), getpgrp());
    int pid=fork();
    if(pid==0) {
        printf("CHILD: PID=%d GID=%d₩n", getpid(), getpgrp());
    }
}
```

```
cju@cju-VirtualBox:~/chap9$ gcc -o pgrp1 pgrp1.c
cju@cju-VirtualBox:~/chap9$ ./pgrp1
[PARENT] PID=22582 GID=22582
CHILD: PID=22583 GID=22582
cju@cju-VirtualBox:~/chap9$
```

프로세스 그룹의 활용 1

- 프로세스 그룹을 이용하면 그룹 내의 모든 프로세스들을 대상으로 액션을 취할 수 있음.
  - 주로 프로세스 그룹 내의 모든 프로세스들에게 특정 시그널(signal)을 보내어 그룹 내의 모든 프로세스들을 제어하거나 종료 시킬 때 사용함.
  - 예
    - kill -9 pid // kill pid와 같음.
    - kill -9 0 // 현재 속한 프로세스 그룹 내의 모든 프로세스들에게 9번 시그널(SIGKILL)을 보냄.
    - kill -9 -pid // -는 프로세스 그룹을 의미. 프로세스 그룹 ID가 pid인 모든 프로세스들에게 9번 시그널(SIGKILL)을 보냄.

프로세스 그룹의 활용 2

- waitpid() 시스템 호출도 프로세스 그룹과 관련이 있음.
  - pid_t waitpid(pid_t pid, int *status, int options);
- 첫번째 인수 pid의 값에 따라 다양하게 사용될 수 있음.
  - pid == -1
    - 임의의 자식 프로세스가 종료하기를 기다림. (wait() 시스템 호출과 같음.)
  - pid > 0
    - PID가 pid인 자식 프로세스가 종료하기를 기다림.
  - pid == 0
    - 호출자와 같은 **프로세스 그룹** 내의 어떤 자식 프로세스가 종료하기를 기다림.
  - pid < -1
    - pid의 절대값과 같은 **프로세스 그룹** 내의 어떤 자식 프로세스가 종료하기를 기다림.

새로운 프로세스 그룹 만들기

- setpgid() 시스템 호출을 통해 새로운 프로세스 그룹을 생성하거나 다른 프로세스 그룹에 멤버로 들어갈 수 있음.
  - 프로세스가 이 호출을 통해 새로운 프로세스 그룹을 생성한 경우 새로운 그룹 리더가 됨.
- 함수 프로토타입
  - int setpgid(pid_t pid, pid_t pgid);
    - PID가 pid인 프로세스의 프로세스 그룹 ID를 pgid로 설정함.
    - 성공하면 0을, 실패하면 -1을 반환함.
  - 예
    - pid == pgid
      - PID가 pid인 프로세스가 새로운 프로세스 그룹의 리더가 됨.
    - pid != pgid
      - PID가 pid인 프로세스가 프로세스 그룹 ID가 pgid인 그룹의 멤버가 됨.
    - pid == 0
      - 호출자의 PID를 사용함.
    - pgid == 0
      - PID가 pid인 프로세스가 새로운 프로세스 그룹의 리더가 됨.

사용 예

- int setpgid(pid_t pid, pid_t pgid);
  - 예
    - pid == pgid
      - PID가 pid인 프로세스가 새로운 프로세스 그룹의 리더가 됨.
    - pid != pgid
      - PID가 pid인 프로세스가 프로세스 그룹 ID가 pgid인 그룹의 멤버가 됨.
    - pid == 0
      - 호출자의 PID를 사용함.
    - pgid == 0
      - PID가 pid인 프로세스가 새로운 프로세스 그룹의 리더가 됨.
  - setpgid(getpid(), getpid());
  - setpgid(0, 0);

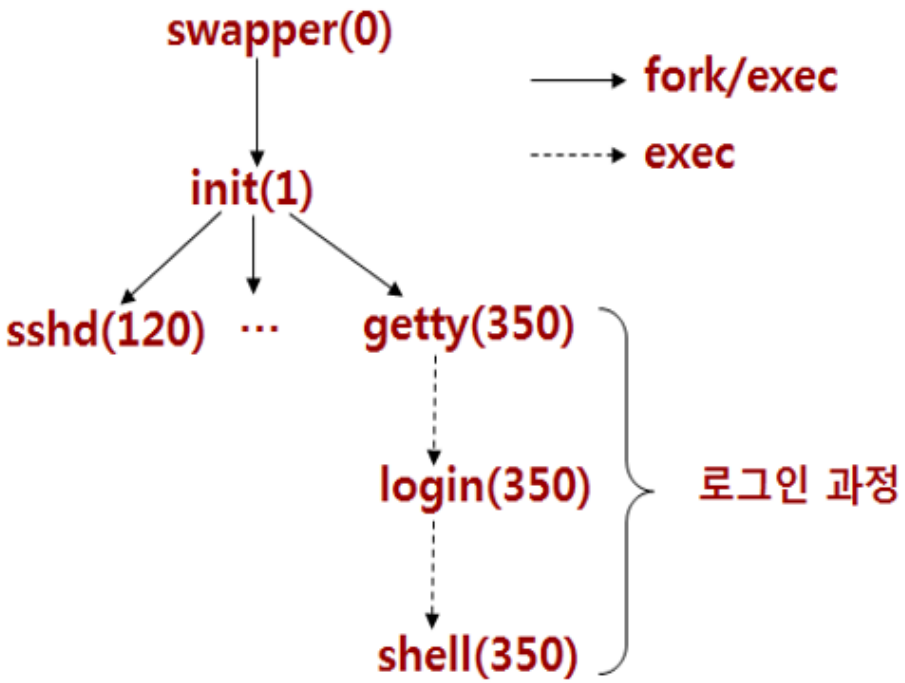
pgrp2.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    printf("[PARENT] PID=%d GID=%d₩n", getpid(), getpgrp());
    int pid=fork();
    if(pid==0) {
        setpgid(0, 0);
        printf("CHILD: PID=%d GID=%d₩n", getpid(), getpgrp());
    }
    return 0;
}
```

```
cju@cju-VirtualBox:~/chap9$ gcc -o pgrp2 pgrp2.c
cju@cju-VirtualBox:~/chap9$ ./pgrp2
[PARENT] PID=22598 GID=22598
CHILD: PID=22599 GID=22599
cju@cju-VirtualBox:~/chap9$
```

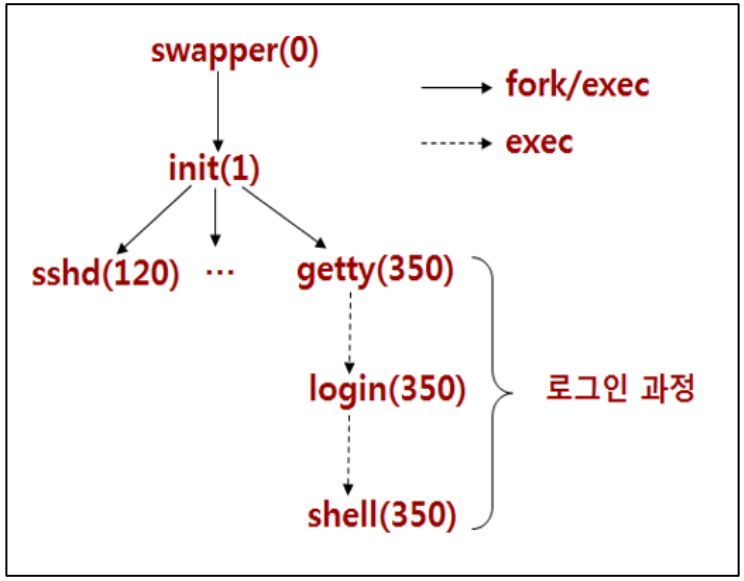
시스템 부팅

- 시스템의 부팅 과정에서 여러 개의 프로세스가 생성되는데 이 과정은 어떻게 이루어질까?
  - 앞에서 배운 fork()와 exec() 시스템 호출을 통해 시스템 부팅이 이루어진다.



시스템 부팅 관련 프로세스들

- swapper (스케줄러 프로세스)
  - 커널 내부에서 만들어진 프로세스로 프로세스 스케줄링을 함.
- init (초기화 프로세스)
  - /etc/inittab 파일에 기술된 대로 시스템을 초기화함.
- getty 프로세스
  - 로그인 프롬프트를 내고 키보드 입력을 감지함.
- login 프로세스
  - 사용자의 로그인 아이디 및 패스워드를 검사함.



- shell 프로세스
  - 시작 파일을 읽고 실행한 후에 쉘 프롬프트를 내고 사용자로부터 명령어를 기다림.

프로세스 리스트 내용 확인

- $ ps -ef | less

```
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0  5월31 ?      00:00:01 /sbin/init splash
root           2       0  0  5월31 ?      00:00:00 [kthreadd]
root           3       2  0  5월31 ?      00:00:00 [rcu_gp]
root           4       2  0  5월31 ?      00:00:00 [rcu_par_gp]
root           5       2  0  5월31 ?      00:00:00 [slub_flushwq]
root           6       2  0  5월31 ?      00:00:00 [netns]
root           8       2  0  5월31 ?      00:00:00 [kworker/0:0H-events_highpri]
root          10       2  0  5월31 ?      00:00:00 [mm_percpu_wq]
root          11       2  0  5월31 ?      00:00:00 [rcu_tasks_kthread]
root          12       2  0  5월31 ?      00:00:00 [rcu_tasks_rude_kthread]
root          13       2  0  5월31 ?      00:00:00 [rcu_tasks_trace_kthread]
root          14       2  0  5월31 ?      00:00:00 [ksoftirqd/0]
root          15       2  0  5월31 ?      00:00:01 [rcu_preempt]
root          16       2  0  5월31 ?      00:00:00 [migration/0]
root          17       2  0  5월31 ?      00:00:00 [idle_inject/0]
root          19       2  0  5월31 ?      00:00:00 [cpuhp/0]
root          20       2  0  5월31 ?      00:00:00 [cpuhp/1]
```

질문

Q&A