UNIT
11

프로세스 제어

로봇SW 교육원
최상훈(shchoi82@gmail.com)

학습 목표

- 리눅스 프로세스 제어

init 프로세스

- init **프로세스**
  - **프로세스** ID : 1
  - **시스템 부팅 절차 마지막 단계에서 커널에 의해 실행됨**
  - **커널 부팅 이후** UNIX **시스템을 뛰우는 역할을 함**
  - **시스템 의존적 초기화 파일들 파일들을 읽고 시스템을 특정한 상태(예: 다중 사용자 상태)로 설정함**
    - /etc/rc*
    - /etc/inittab
    - /etc/init.d
  - **종료되지 않음**
    - **초기화를 수행 완료한 후 종료되지 않고 일반 사용자 프로세스로 남아 고아가 된 자식 프로세스의 부모 프로세스 역할을 수행함**
  - /sbin/init

```
$ ls -l /sbin/init
-rwxr-xr-x 1 root root 31328 Oct 13  2013 /sbin/init
```

프로세스 ID

- 프로세스 ID
  - 커널에 의해 관리
  - 프로세스를 구별하는 고유한 ID가 부여됨
  - 음이 아닌 정수 값

- 대부분의 UNIX System들은 방금 종료된 프로세스의 ID가 새로 만든 프로세스에 배정되는 일을 방지하기 위해 프로세스 ID의 재사용을 지연하는 알고리즘들을 구현하고 있음

실습1: 프로세스 ID


```
#include<unistd.h>
pid_t getpid(void);
                                        Returns : process ID of calling process
pid_t getppid(void);
                                    Returns : parent process ID of calling process
```

파일명 : pid.c

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

int main(void)
{
    printf("getpid:%d\n", getpid());
    printf("getppid:%d\n", getppid());
    return 0;
}
```

프로세스와 관련된 ID

- **프로세스는 여러 가지** ID**와 관련되어 있음**
  - **실제**(real) **사용자/그룹** ID
    - **로그인 시 패스워드 파일에서 읽어 오는** ID
    - **로그인 세션 동안에는 바뀌지 않음**
  - **유효**(effective) **사용자/그룹** ID, **추가 그룹** ID
    - **파일에 대한 접근 권한을 결정함**
  - **저장된** set-user-ID(saved SUID), **저장된** group-user-ID(saved SGID)
    - exec( ) **함수 수행 시 저장된 유효 사용자/그룹** ID

- **일반적인 프로그램을 실행**
  - **실제 사용자/그룹** ID = **유효 사용자/그룹** ID

- set-user-ID(SUID), set-group-ID(SGID)**비트가 설정된 파일을 실행**
  - **실행 파일의 소유 사용자/그룹** ID = **유효 사용자/그룹** ID

실습2: 프로세스와 관련된 ID


```
uid_t getuid(void);
                                        Returns : real user ID of calling process
uid_t geteuid(void);
                                    Returns : effective user ID of calling process
gid_t getgid(void);
                                        Returns : real group ID of calling process
gid_t getegid(void);
                                    Returns : effective group ID of calling process
```

파일명 : ids.c

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

int main(void)
{
    printf("getuid:%d\n", getuid());
    printf("geteuid:%d\n", geteuid());
    printf("getgid:%d\n", getgid());
    printf("getegid:%d\n", getegid());
    return 0;
}
```

fork 함수


```
#include<unistd.h>
pid_t fork(void);
            Returns : 0 in child, process ID of child in parent, -1 on error
```

- **기존 프로세스(부모 프로세스)가**
  새 프로세스(자식 프로세스)를 생성
- **한번 호출, 두번 반환**
- **자식 프로세스는 부모 프로세스의 복사본**
  - **자식, 부모 프로세스 모두 fork호출 이후의 명령들을 계속 실행함**
  - **부모의 자료구역, 힙, 스택의 복사본을 갖음**
    - COW(Copy-On-Write) **기법을 사용함**
  - **텍스트 구역은 공유함**

실습3: fork 함수(1/2)

파일명 : forkEx1.c

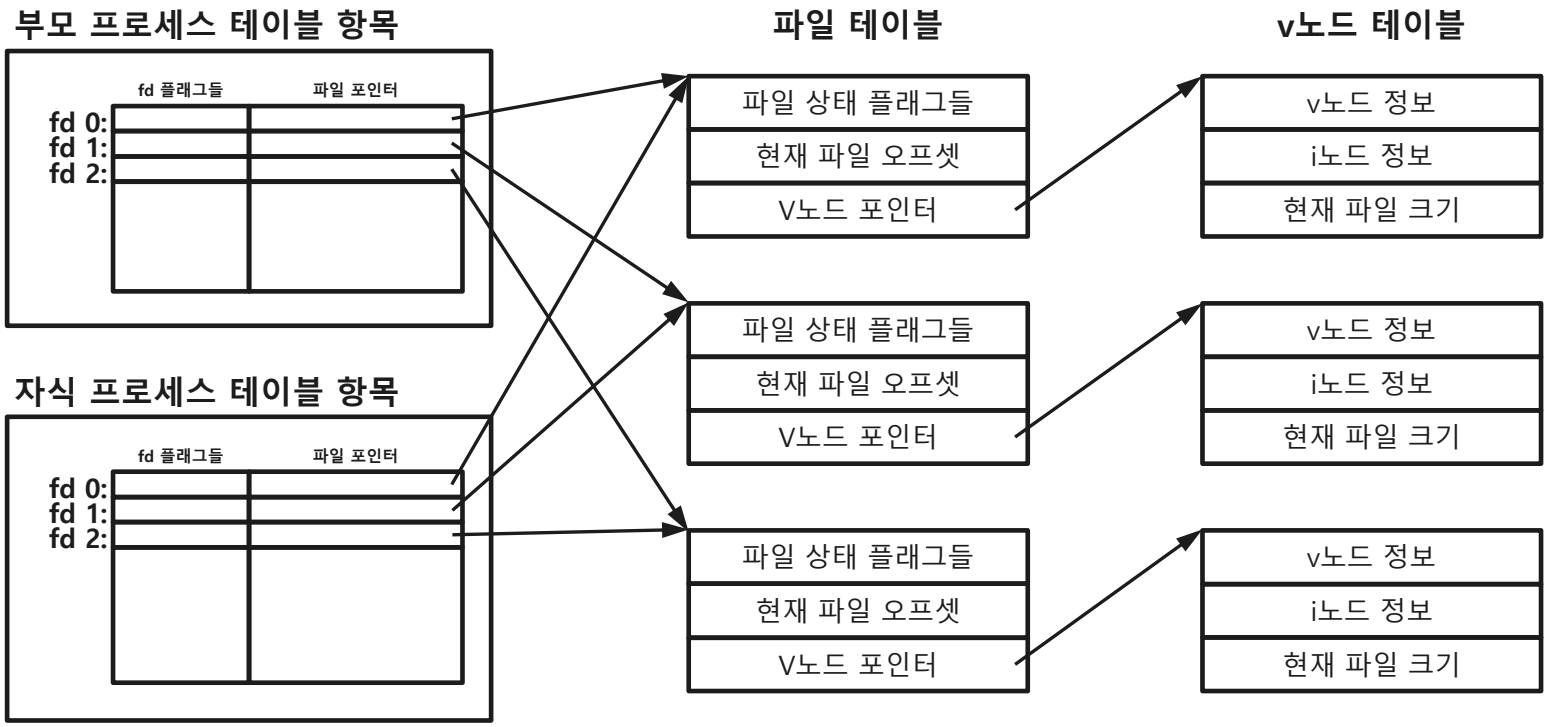
```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int glob = 6;
char buf[] = "a write to stdout\n";

int main(int argc, char *argv[])
{
     int var;
     pid_t pid;
     var = 88;
     if(write(STDOUT_FILENO, buf, sizeof(buf)-1) != sizeof(buf)-1){
          fprintf(stderr, "write error\n");
          exit(1);
     }
     printf("before fork\n");
     //fflush(stdout);
     if((pid = fork()) < 0){
          fprintf(stderr, "fork error\n");
          exit(1);
     }else if(pid == 0){
          glob++;
          var++;
     }else{
          sleep(2);
     }
     printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
     exit(0);
}
```

실습3: fork 함수(2/2)


```
$ ./forkEx1
a write to stdout
before fork
pid = 18052, glob = 7, var = 89
pid = 18051, glob = 6, var = 88
$
$
$
$ ./ forkEx1 > temp.out
$ cat temp.out
a write to stdout
before fork
pid = 18054, glob = 7, var = 89
before fork
pid = 18053, glob = 6, var = 88
$
```

fork 함수 : 파일 공유

- 부모의 열린 파일 서술자들이 모두 자식에게 복사됨
  - 해당 파일 서술자에 대해 dup함수가 호출된 것과 같은 효과
  - 모든 열린 파일 서술자에 대해 동일한 파일 테이블 항목을 공유
    - 동일한 파일 오프셋을 공유함



실습4-1: fork 함수(파일의 공유)

파일명 : forkEx2.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<string.h>
#define PERM    0644
#define MAXBUF  256
char msg[MAXBUF];
int main(int argc, char *argv[])
{
    char *szFile = "data";
    int fd;
    long offset;
    pid_t pid;

    if((fd = open(szFile, O_RDWR | O_CREAT, PERM)) == -1){
        fprintf(stderr, "open error\n");
        exit(1);
    }

    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }
```

실습4-2: fork 함수(파일의 공유)


```
    else if(pid == 0){                  /* 자식 프로세스 */
        printf("%d:I'm child\n", getpid());
        sleep(2);
        sprintf(msg, "pid:%d write\n", getpid());
        if(write(fd, msg, strlen(msg)) != strlen(msg)){
            fprintf(stderr, "write error\n");
            exit(1);
        }
        offset = lseek(fd, 0, SEEK_CUR);
        printf("%d:offset:%ld\n", getpid(), offset);
    }else{                               /* 부모 프로세스 */
        printf("%d:I'm parent\n", getpid());
        sprintf(msg, "pid:%d write\n", getpid());
        if(write(fd, msg, strlen(msg)) != strlen(msg)){
            fprintf(stderr, "write error\n");
            exit(1);
        }

        offset = lseek(fd, 0, SEEK_CUR);
        printf("%d:offset:%ld\n", getpid(), offset);
    }
    close(fd);
    exit(0);
}
```

실습4-3: fork 함수(파일의 공유)


```
$ ./forkEx2
2706:I'm parent
2706:offset:15
2707:I'm child
$ 2707:offset:30 [Enter]

$
$ cat data
pid:2706 write
pid:2707 write
$
```

fork 함수

- fork **시스템 콜에 의해 자식에게 상속되는 속성**
  - **열린 파일들(파일 디스크립터 테이블)**
  - **열린 파일 서술자들에 대한** exec**시 닫기**(close-on-exec) **플래그**
  - **실제 사용자/그룹** ID, **유효 사용자/그룹** ID
  - SUID**플래그와** SGID**플래그**
  - **현재 작업 디렉토리**
  - **파일 모드 생성 마스크**
  - **환경 목록**
  - **신호 마스크와 신호 처리 설정들**
  - **추가 그룹** ID**들**, **부착된 공유 메모리 영역들**, **메모리 매핑 들**
  - **자원 한계들**, **프로세스 그룹** ID, **세션** ID, **제어터미널**, **루트 디렉토리**

fork 함수

- **자식 프로세스와 부모 프로세스의 차이**
  - fork**의 반환 값**
  - **프로세스의** ID, **부모 프로세스의** ID
  - **부모가 잠근 파일 자물쇠들은 자식에게 상속되지 않음**
  - **아직 발동되지 않은 정보**(alarm)**들은 자식에서 모두 해제됨**
  - **자식의 유보 중인 신호 집합은 비워짐(빈 집합이 됨)**

- fork **함수가 실패하는 경우**
  - **시스템에 너무 많은 프로세스들이 있을 때**
  - **사용자** ID**당 프로세스의 최대 개수인** CHILD_MAX**값을 넘었을 때**

vfork 함수


```
#include<unistd.h>
pid_t vfork(void);
            Returns : 0 in child, process ID of child in parent, -1 on error
```

- **프로세스 생성이후 즉시** exec**를 실행하는 경우에 특화된 버전**
- **부모의 프로세스공간을 자식에게 복사하지 않음**
  - vfork **이후 즉시** exec(**또는** exit)**가 호출할 테고, 따라서 부모의 주소공간을 참조하는 일은 없을 것을 가정하기 때문임**
- **자식 프로세스와 부모 프로세스가 같은 주소공간에서 실행됨**
- **자식 프로세스가 먼저 실행되는것을 보장함**
  - **자식이** exec**나** exit**를 호출하기 전까지 부모 프로세스의 실행은 유보됨**

실습5:vfork 함수

파일명 : vfork.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int glob = 6;                   /* 초기화된 자료 구역에 있는 외부 변수 */
int main(void)
{
    int var;                    /* 스택에 저장되는 자동 변수 */
    pid_t pid;
    var = 88;
    printf("before vfork\n");
    //fflush(stdout);
    if((pid = vfork()) < 0){
        fprintf(stderr, "fork error\n");         exit(1);
    }else if(pid == 0){             /* 자식 프로세스 */
        glob++;                     /* 변수들을 수정한다 */
        var++;
        _exit(0);                   /* 자식이 종료됨 */
    }
    /* 부모는 여기에서부터 실행을 재개한다 */
    printf("pid = %d, glob = %d, var = %d\n", getpid(), glob, var);
    exit(0);
}
```

```
$ ./vfork
before vfork
pid = 18077, glob = 7, var = 89
$
```

exit 함수들

- **프로세스가 종료되는 상황**(8**가지**)
  - **정상적인 종료**(5)
    - main **의 반환**(return)
    - exit **호출**
    - _exit **또는** _Exit **호출**
    - **마지막 스레드를 시작한 스레드 시동 루틴의 반환**(return)
    - **마지막 스레드의** pthread_exit **호출**
  - **비정상적인 종료**(3)
    - abort **호출**(SIGABRT)
    - signal **수신**
    - **마지막 스레드의 실행 취소**
- **프로세스의 종료**
  - **프로세스가 어떻게 종료되든 결국에는 커널 안의 동일한 코드가 수행됨**
  - **열린 서술자들을 모두 닫음**
  - **프로세스가 사용한 메모리 해제**
  - **기타 마무리 작업**

exit 함수들

- **종료 상태**(exit status)
  - **정상적인 종료인** exit **함수들**(exit, _exit, _Exit)**의 호출 시 인자 값**
- **종지 상태**(termination status)
  - **최종적으로** _exit**가 호출될 때 커널이 프로세스의 종료상태를 종지상태로 변환**
  - **비정상적인 종료의 경우 커널이 종지상태를 결정함**
- **프로세스가 종료되면 커널은** SIGCHLD **신호를 부모 프로세스에게 보냄**

  ※ SIGCHLD **신호에 대한 기본 동작은 '무시'**

exit 함수들

- **고아 프로세스**
  - **자식보다 부모 프로세스가 먼저 종료 됐을 때**
  - **일반적으로 한 프로세스가 종료되면 커널은 모든 활성 프로세스를 훑으면서 종료된 프로세스의 자식 프로세스들이 남아 있는지 찾음, 만일 자식이 남아 있으면 그 자식 프로세스들의 부모 프로세스** ID**를** 1**로 설정함**
    (init **프로세스**)
    - **반드시 하나의 부모 프로세스를 갖게 됨**

- **좀비**(zombie) **프로세스**
  - **자식 프로세스가 종료되면 종지상태 등 자식 프로세스가 종료된 상태를 확인하기 위한 일부분의 자료를 남겨둠**
    - **자식의 프로세스** ID, **프로세스의 종지 상태, 프로세스가 사용한** CPU **시간 등**

실습6: 고아 프로세스

파일명 : orphan.c

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

int main(int argc, char *argv[])
{
    pid_t pid;

    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0){
        printf("I'am Child Porcess : %d\n", getpid());
        sleep(2);
        printf("My parent process : %d\n", getppid());
        exit(1);
    }else{
        printf("I'am Parent Porcess : %d\n", getpid());
        exit(1);
    }
    return 1;
}
```

```
$ ./orphan
I'am Parent Porcess : 2817
$ I'am Child Porcess : 2818
My parent process : 1
[Enter]
$
```

실습7-1: 좀비 프로세스

파일명 : zombie.c

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

int main(int argc, char *argv[])
{
    pid_t pid;

    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0){
        printf("I'am Child Porcess : %d\n", getpid());
        printf("Child Porcess end : %d\n", getpid());
        exit(1);
    }else{
        printf("I'am Parent Porcess : %d\n", getpid());
        sleep(10);
        exit(1);
    }
    return 1;
}
```

실습7-2: 좀비 프로세스


```
$ ./zombie &
[1] 2877
$ I'am Parent Porcess : 2877
I'am Child Porcess : 2878
Child Porcess end : 2878
[Enter]
$ ps
  PID TTY          TIME CMD
 2842 pts/1    00:00:00 bash
 2877 pts/1    00:00:00 zombie
 2878 pts/1    00:00:00 zombie <defunct>
 2879 pts/1    00:00:00 ps
$
[1]+  Exit 1                  ./zombie
$
```

```
$ ./zombie
I'am Parent Porcess : 2880
I'am Child Porcess : 2881
Child Porcess end : 2881
$
```

```
$ ps u
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pi        2615  0.0  1.0   6272  4704 pts/0    Ss   18:48   0:02 -bash
pi        2850  0.2  0.9   5688  4028 pts/2    Ss   20:46   0:00 /bin/bash
pi        2880  0.0  0.2   1548   932 pts/1    S+   20:50   0:00 ./zombie
pi        2881  0.0  0.0      0     0 pts/1    Z+   20:50   0:00 [zombie] <defunct>
pi        2882  0.0  0.4   4464  2116 pts/2    R+   20:50   0:00 ps u
$
```

wait 함수와 waitpid 함수


```
#include<sys/wait.h>
pid_t wait(int *statloc);
pid_t waitpid(pid_t pid, int *statloc, int options);
                Both return : process ID if OK, 0(see later), or -1 on error
```

- **부모가 자식 프로세스의 종료를 기다리게 함**
- **자식 프로세스의 종지 상태를 회수함**
- **반환 값** : **종료된 프로세스의** ID
- **종지상태를 알 필요가 없을 경우** statloc**인수** NULL

wait 함수와 waitpid 함수

- wait함수와 waitpid함수 중 하나를 부모에서 호출할 때 호출이 반환되는 방식은 상황에 따라 다름
  - 부모의 자식 프로세스들이 모두 아직 실행 중이면 호출이 반환되지 않음 (부모의 실행이 차단됨)
  - 한 자식이 프로세스가 종료되어 해당 종지 상태의 회수를 기다리고 있는 상황이라면 호출이 즉시 반환됨
  - 자식 프로세스가 하나도 없으면 즉시 오류가 반환됨
- 두 함수의 차이점
  - wait함수는 하나의 자식 프로세스가 종료될 때까지 차단되나 waitpid함수는 인자(WNOHANG)를 설정하면 차단을 방지할 수 있음
  - wait함수는 임의의 자식 프로세스를 기다리지만 waitpid함수는 특정한 자식 프로세스를 지정할 수 있음

wait 함수와 waitpid 함수

- waitpid는 wait에 비해 좀더 유연한 기능을 갖음
- pid 인자 값에 따른 기능

pid == -1	임의의 자식 프로세스를 기다림
pid > 0	프로세스 ID가 pid인 한 자식 프로세스를 기다림
pid == 0	프로세스 그룹 ID가 호출한 프로세스의 것과 동일한 임의의 자식 프로세스를 기다림
pid < -1	프로세스 그룹 ID가 pid의 절대값과 같은 임의의 자식 프로세스를 기다림

wait 함수와 waitpid 함수

- **종지 상태** (statloc **정수 포인터**)
  - **상태 값내에 저장되는 내용**
    - **종료상태(정상적인 종료의 경우)**
    - **신호의 번호(비정상적인 종료의 경우)**
    - **코어파일 생성여부**

- **종지 상태를 확인하기 위한 매크로 정의** (POSIX.1)
  - **헤더파일** <sys/wait.h>
  - **네 게의 매크로 중 하나만 참(상호 배타적임)**
    - WIFEXITED(status)
    - WIFSIGNALED(status)
    - WIFSTOPPED(status)
    - WIFCONTINUED(status)

wait 함수와 waitpid 함수

- WIFEXITED(status)
  - **자식 프로세스가 정상적으로 종료되었으면 참**
  - WEXITSTATUS(status)**를 이용해 자식의 종료 상태를 알아낼 수 있음**
  - **자식이** exit, _exit, _Exit**로 넘겨준 인수의 하위** 8**비트**
- WIFSIGNALED(status)
  - **자식 프로세스가 신호를 받았으나 그것을 처리하지 않아서 비정상적으로 종료되었으면 참**
  - WTERMSIG(status)**를 이용해서 종료를 유발한 신호의 번호를 알 수 있음**
  - **코어파일 생성여부 확인 매크로** WCOREDUMP(status)
- WIFSTOPPED(status)
  - **자식 프로세스가 현재 중지 중이면 참**
  - WSTOPSIG(status)**를 이용해 중지를 유발한 신호의 번호를 알수 있음**
- WIFCONTINUED(status)
  - **자식 프로세스가 작업 제어 중지 이후 실행이 재개되었으면 참**

실습8-1: 종지상태

파일명 : termstatus.c

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

void pr_exit(int status)
{
    if(WIFEXITED(status))
        printf("normal termination, exit status = %d\n", WEXITSTATUS(status));
    else if(WIFSIGNALED(status))
        printf("abnormal termination, signal number = %d%s\n", WTERMSIG(status),
#ifdef WCOREDUMP
                WCOREDUMP(status) ? "(core file generated)" : "");
#else
    "");
#endif
    else if(WIFSTOPPED(status))
        printf("child stopped, signal number = %d\n", WSTOPSIG(status));
    else if(WIFCONTINUED(status))
        printf("child continued\n");
}
```

실습8-2: 종지상태


```
int main(void)
{
    pid_t pid;
    int status;
    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0)
        exit(7);        //정상 종료
    if(wait(&status) != pid){
        fprintf(stderr,"wait error\n");
        exit(1);
    }
    pr_exit(status);

    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0)
        abort();        //비정상 종료
    if(wait(&status) != pid){
        fprintf(stderr, "wait error\n");
        exit(1);
    }
    pr_exit(status);
```

```
 if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0)
        status /= 0;    // 비정상 종료
    if(wait(&status) != pid){
        fprintf(stderr, "wait error\n");
    }
    pr_exit(status);
    exit(0);
}
```

```
$ ./termstatus
normal termination, exit status = 7
abnormal termination, signal number = 6
abnormal termination, signal number = 8
$
```

실습9-1: waitpid

파일명 : waitpid.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<wait.h>
int main(int argc, char *argv[])
{
    pid_t pid,  ret_pid;
    int status;
    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0){
        printf("Child Porcess is started : %d\n", getpid());
        sleep(5);
        printf("Child Porcess is finished : %d\n", getpid());
        exit(7);
    } else{
        printf("Parent Porcess is started : %d\n", getpid());
        while((ret_pid = waitpid(-1, &status, WNOHANG)) == 0){
            printf("  waitpid WNOHANG\n");
            sleep(1);
        }
        printf("  waitpid return %d\n", ret_pid);
        printf("  child exit status:%d\n", WEXITSTATUS(status));
        printf("Parent Porcess is finished : %d\n", getpid());
        exit(1);
    }
    return 1;
}
```

실습9-2: waitpid


```
$ ./waitpid
Parent Porcess is started : 2937
  waitpid WNOHANG
Child Porcess is started : 2938
  waitpid WNOHANG
  waitpid WNOHANG
  waitpid WNOHANG
  waitpid WNOHANG
Child Porcess is finished : 2938
  waitpid WNOHANG
  waitpid return 2938
  child exit status:7
Parent Porcess is finished : 2937
$
```

exec류 함수들


```
#include<unistd.h>
int execl(const char *pathname, const char *arg0, … /* (char *)0 */ );
int execv(const char *pathname, const *const argv[]);
int execle(const char *pathname, const char *arg0, …
              /* (char *)0 , char *const envp[] */ );
int execve(const char *pathname, char *const argv[], char *const envp[]);
int execlp(const char *filename, const char *arg0, … /* (char *)0 */ );
int execvp(const char *filename, char *const argv[]);
                          All six return : -1 on error, no return on success
```

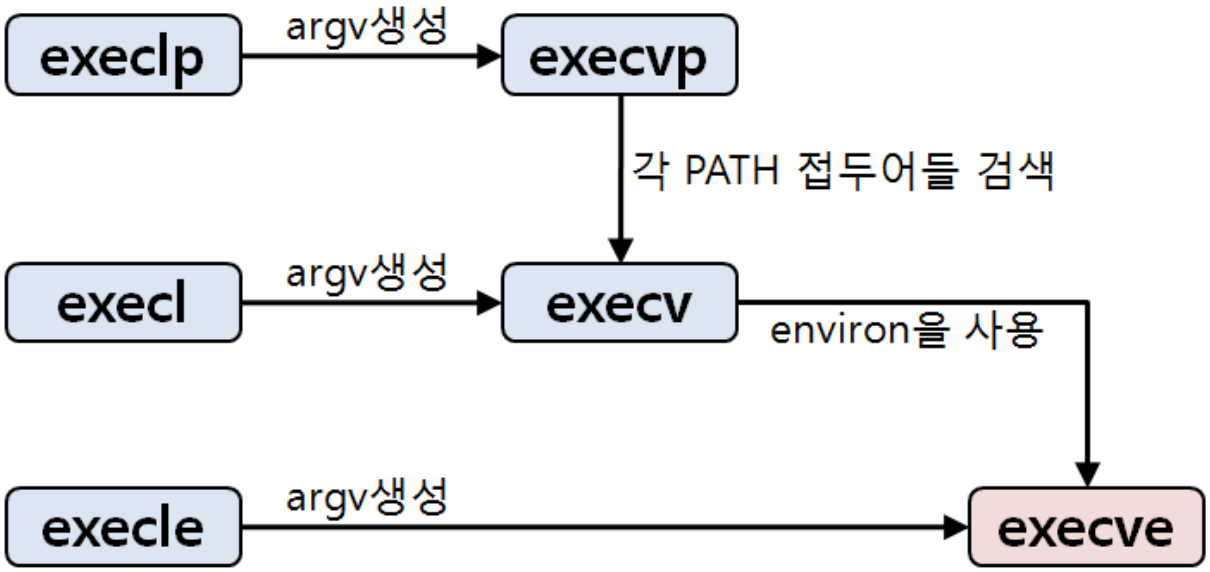
- **새 프로그램을 시동**
- **새 프로그램으로 완전히 대체되어 main함수에서 실행이 다시 시작됨**
- **실행파일로부터 이미지(텍스트, 데이터, 힙, 스택구역)를 로딩하여 현재 프로세스를 새 이미지로 대체**

exec류 함수들

- exec를 호출한 프로세스의 새 프로그램으로 상속되는 특성들
  - 프로세스ID와 부모 프로세스ID
  - 실제 사용자 ID와 실제 그룹 ID
  - 현재 작업 디렉토리
  - 파일 모드 생성 마스크
  - 추가 그룹 ID
  - 프로세스 그룹 ID
  - 세션 ID
  - 제어 터미널
  - 경보(alarm) 발동까지 남은 시간
  - 루트 디렉토리
  - 파일 자물쇠
  - 프로세스 신호 마스크
  - 아직 처리되지 않은 신호들
  - 자원의 한계들
  - tms_utime, tms_stime, tms_cutime, tms_cstime 값들

exec류 함수들

- **열린 파일들이 어떻게 처리되는 가는 각 서술자의** FD_CLOEXEC (exec **호출시 닫기** close_on_exec) **플래그에 따라 결정됨**
- **새 프로그램 파일의** SUID, SGID **비트의 설정 여부에 따라 유효 사용자** ID**와 유효 그룹** ID**는 변할 수 있음**
- **시스템 호출** execve, **나머지는 라이브러리 함수**



실습10-1: exec 함수

파일명 : exec.c

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

char *env_init[] = {"USER=unknown", "PATH=/tmp", NULL};

int main(void)
{
    pid_t pid;
    if((pid = fork()) < 0){
        fprintf(stderr, "fork error\n");
        exit(1);
    } else if(pid == 0){          /* 경로이름 설정, 환경을 지정 */
        if(execle("./echoall", "echoall", "myarg1", "ARG2", (char *)0, env_init) < 0){
            fprintf(stderr, "execle error\n");
            exit(1);
        }
    }
    if(waitpid(pid, NULL, 0) < 0){
        fprintf(stderr, "wait error\n");
        exit(1);
    }
```

실습10-2: exec 함수

파일명 : exec.c

```
    if((pid = fork()) < 0) {
        fprintf(stderr, "fork error\n");
        exit(1);
    }else if(pid == 0){
        if(execlp("./echoall", "echoall", "only 1 arg", (char *)0) < 0){
            fprintf(stderr, "execlp error\n");
            exit(1);
        }
    }
    if(waitpid(pid, NULL, 0) < 0){
        fprintf(stderr, "wait error\n");
        exit(1);
    }
    exit(0);
}
```

실습10-3: exec 함수


```
// echoall 프로그램
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
    int i;
    char **ptr;
    extern char **environ;

    /* 명령줄 인수들을 모두 출력한다. */

    for(i = 0 ; i < argc ; i++)
        printf("argv[%d]: %s\n", i, argv[i]);

    for(ptr = environ; *ptr != 0 ; ptr++)
        printf("%s\n", *ptr);
    exit(0);
}
```

실습10-4: exec 함수


```
$ ./exec
argv[0]: echoall
argv[1]: myarg1
argv[2]: ARG2
USER=unknown
PATH=/tmp
argv[0]: echoall
argv[1]: only 1 arg
SHELL=/bin/bash
TERM=screen
SSH_CLIENT=192.168.1.8 55201 22
SSH_TTY=/dev/pts/0
USER=pi
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;
43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01
;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*
.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31
:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01
;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.sv
g=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;3
5:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01
;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=
01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m
id=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.axa=00;36:*.oga=00;36:
*.spx=00;36:*.xspf=00;36:
TERMCAP=SC|screen|VT 100/ANSI X3.64 virtual terminal:\
        :DO=\E[%dB:LE=\E[%dD:RI=\E[%dC:UP=\E[%dA:bs:bt=\E[Z:\
        :cd=\E[J:ce=\E[K:cl=\E[H\E[J:cm=\E[%i%d;%dH:ct=\E[3g:\
        :do=^J:nd=\E[C:pt:rc=\E8:rs=\Ec:sc=\E7:st=\EH:up=\EM:\
        :le=^H:bl=^G:cr=^M:it#8:ho=\E[H:nw=\EE:ta=^I:is=\E)0:\
        :li#60:co#120:am:xn:xv:LP:sr=\EM:al=\E[L:AL=\E[%dL:\
        :cs=\E[%i%d;%dr:dl=\E[M:DL=\E[%dM:dc=\E[P:DC=\E[%dP:\
        :im=\E[4h:ei=\E[4l:mi:IC=\E[%d@:ks=\E[?1h\E=:\
        :ke=\E[?1l\E>:vi=\E[?25l:ve=\E[34h\E[?25h:vs=\E[34l:\
        :ti=\E[?1049h:te=\E[?1049l:us=\E[4m:ue=\E[24m:so=\E[3m:\
        :se=\E[23m:mb=\E[5m:md=\E[1m:mr=\E[7m:me=\E[m:ms:\
        :Co#8:pa#64:AF=\E[3%dm:AB=\E[4%dm:op=\E[39;49m:AX:\
```

실습10-4: exec 함수


```
        :vb=\Eg:G0:as=\E(0:ae=\E(B:\
        :ac=\140\140aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~..--++,,hhII00:\
        :po=\E[5i:pf=\E[4i:Km=\E[M:k0=\E[10~:k1=\EOP:k2=\EOQ:\
        :k3=\EOR:k4=\EOS:k5=\E[15~:k6=\E[17~:k7=\E[18~:\
        :k8=\E[19~:k9=\E[20~:k;=\E[21~:F1=\E[23~:F2=\E[24~:\
        :F3=\E[1;2P:F4=\E[1;2Q:F5=\E[1;2R:F6=\E[1;2S:\
        :F7=\E[15;2~:F8=\E[17;2~:F9=\E[18;2~:FA=\E[19;2~:kb=:\
        :K2=\EOE:kB=\E[Z:kF=\E[1;2B:kR=\E[1;2A:*4=\E[3;2~:\
        :*7=\E[1;2F:#2=\E[1;2H:#3=\E[2;2~:#4=\E[1;2D:%c=\E[6;2~:\
        :%e=\E[5;2~:%i=\E[1;2C:kh=\E[1~:@1=\E[1~:kH=\E[4~:\
        :@7=\E[4~:kN=\E[6~:kP=\E[5~:kI=\E[2~:kD=\E[3~:ku=\EOA:\
        :kd=\EOB:kr=\EOC:kl=\EOD:km:
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games
MAIL=/var/mail/pi
STY=2841.pts-0.raspberrypi-robotcode77
PWD=/home/pi/12
LANG=en_GB.UTF-8
HOME=/home/pi
SHLVL=2
LOGNAME=pi
WINDOW=0
SSH_CONNECTION=192.168.1.8 55201 192.168.1.10 22
_=./exec
$
```

실습11-1: exec 함수

파일명 : execEx2.c

```
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>

int system(const char *cmdstring);

int
main(int argc, char *argv[])
{
    if(argc != 2){
        fprintf(stderr,"command-line argument required");
        return 1;
    }

    system(argv[1]);
    return 0;
}
```

실습11-2: exec 함수


```
int
system(const char *cmdstring)
{
    pid_t pid;
    int status;
    if (cmdstring == NULL)
        return(1);
    if ((pid = fork()) < 0) {
        status = -1;
    } else if (pid == 0) {          // child process
        execl("/bin/bash", "bash", "-c", cmdstring, (char *)0);
        _exit(127);                 // execl error
    } else {                        // parent process
        while (waitpid(pid, &status, 0) < 0) {
            if (errno != EINTR) {
                status = -1;        // error other than EINTR from waitpid()
                break;
            }
        }
    }
    return(status);
}
```

실습11-3: exec 함수


```
pi@shchoi82 ~ $ ./execEx2 "ls -al > filelist"
pi@shchoi82 ~ $ ./ execEx2 "date > curDate"
pi@shchoi82 ~ $ cat curDate
2015. 11. 14. (토) 16:09:00 UTC
pi@shchoi82 ~ $ ./ execEx2 "cal > curCal"
pi@shchoi82 ~ $ cat curCal
      11월 2015
일 월 화 수 목 금 토
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

pi@shchoi82 ~ $
```

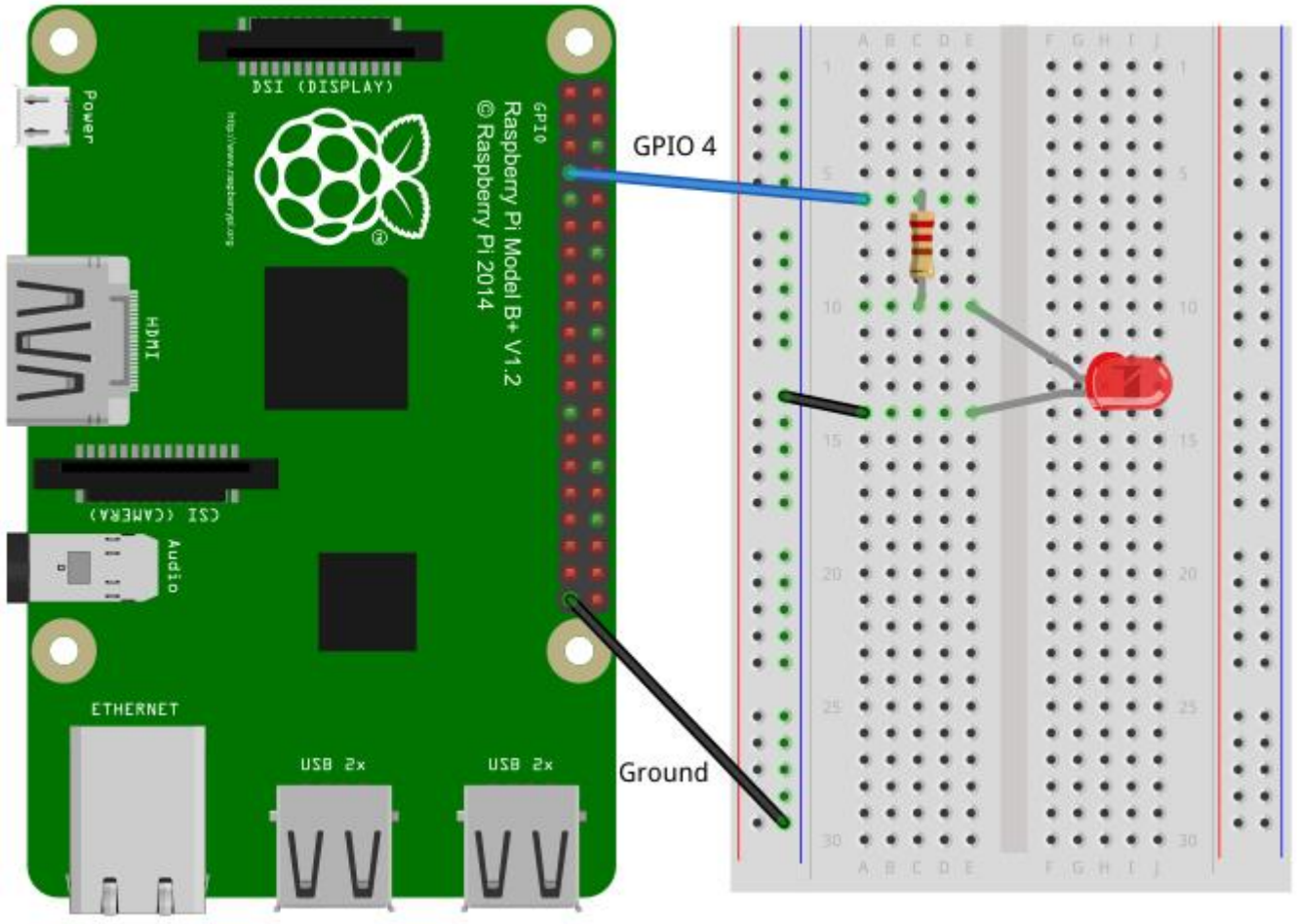
system 함수


```
#include <stdlib.h>
int system(const char *cmdstring);
```

- **프로그램 내에서 명령을 실행할 때 편리함**
- **내부적으로** fork, exec, waitpid**를 호출하여 실행됨**
- sh –c “command string”

실습12-1: system으로 gpio명령제어

- 구성
  - LED(GPIO 4)



실습12-2

파일명 : system.c

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

int
main(int argc, char *argv[])
{
    int i;
    int pinNo;
    int repeat;
    char szCmdString1[1024] = {0};
    char szCmdString2[1024] = {0};
    char szCmdString3[1024] = {0};

    if(argc != 3){
        fprintf(stderr,"usage : a.out [pinNo] [repeat]\n");
        return 1;
    }

    pinNo = atoi(argv[1]);
    repeat = atoi(argv[2]);

    sprintf(szCmdString1, "gpio -g mode %d out", pinNo);
    system(szCmdString1);
```

실습12-3


```
    sprintf(szCmdString2, "gpio -g write %d 1", pinNo);
    sprintf(szCmdString3, "gpio -g write %d 0", pinNo);

    for(i = 0 ; i < repeat ; i++){
        printf("%s\n", szCmdString2);
        system(szCmdString2);
        sleep(1);
        printf("%s\n", szCmdString3);
        system(szCmdString3);
        sleep(1);
    }

    return 0;
}
```

```
pi@robotcode ~ $ ./system 4 3
gpio -g write 4 1
gpio -g write 4 0
gpio -g write 4 1
gpio -g write 4 0
gpio -g write 4 1
gpio -g write 4 0
pi@robotcode ~ $
```

부팅 시 프로세스 자동 시작

- rc.local **파일**
  - **부팅 시 자동으로 실행됨**

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

자동으로 실행할 명령어

exit 0
```

부팅 시 프로세스 자동 시작

- **다른 사용자 계정**(UID, GID **환경)으로 명령 실행**
  - su **명령어**
    - **예**)

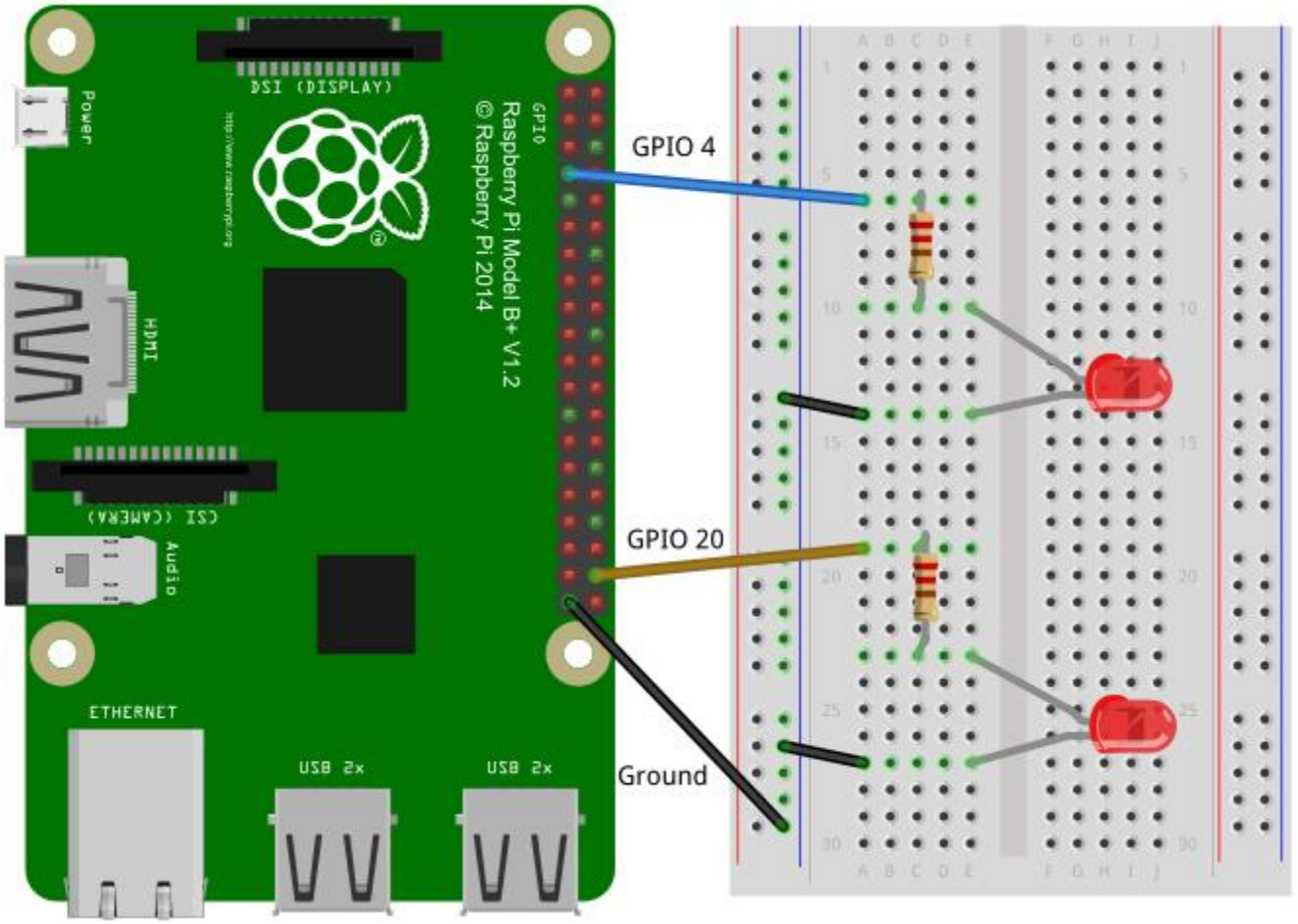
```
$su -l pi -c "/home/pi/system 4 3"
$su pi -c "/home/pi/system 4 3"
```

  - sudo **명령어**
    - **예**)

```
$sudo -u pi /home/pi/system 4 3
```

실습13-1 : 부팅 시 프로세스 자동 시작

- 구성
  - LED(GPIO 4,20)



실습13-2

- rc.local **파일 수정**

  $ sudo vim /etc/rc.local

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

su -l pi -c "/home/pi/system 4 3"

sudo -u pi /home/pi/system 20 3

exit 0
```

실습13-3

- **재부팅**

  $ sudo reboot