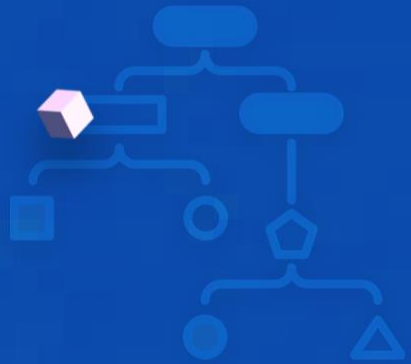
강원혁신플랫폼

리눅스프로그래밍

프로세스와 다중 처리 프로그래밍



Linux에서 현재 프로세스 이미지를 새 프로그램 이미지로 바꾸는 데 사용하는 함수는 무엇인가요?

exec() 함수

학습목표

학습 내용

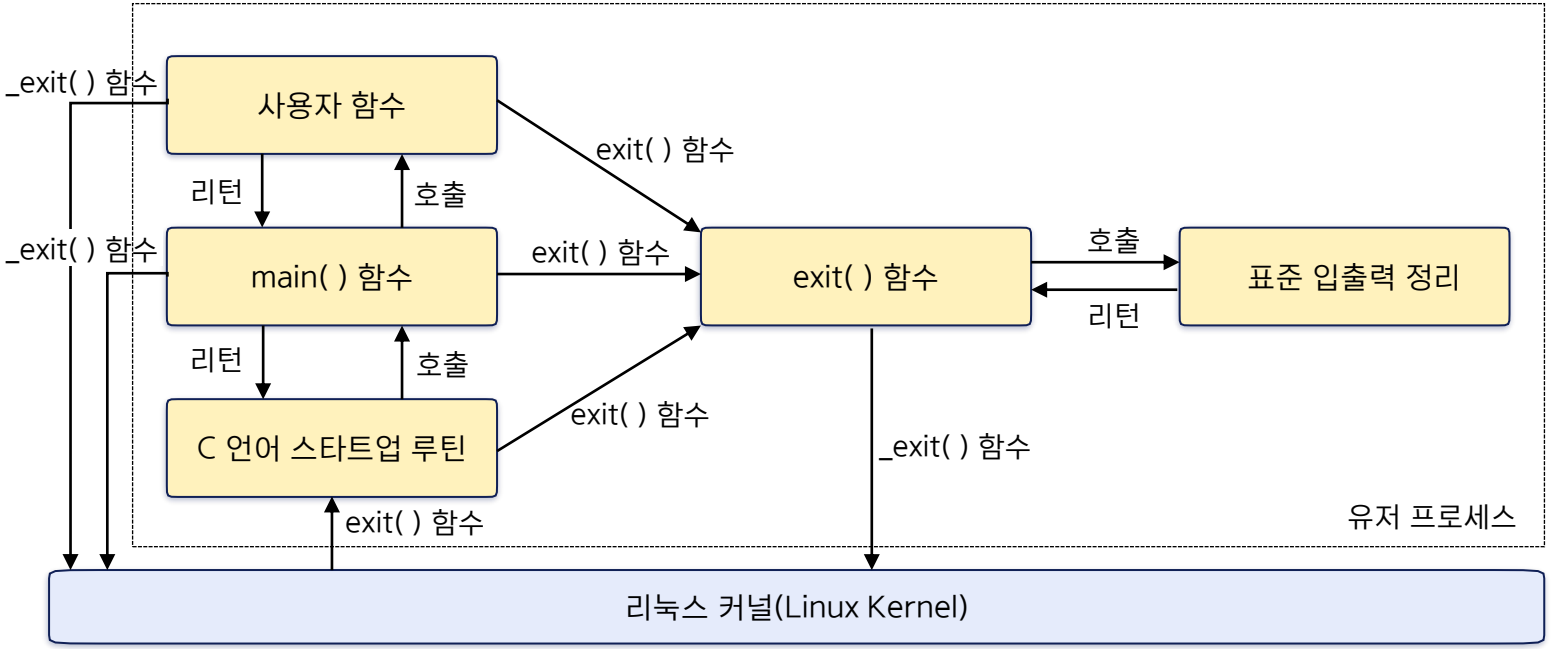
1. 프로세스와 다중 처리 프로그래밍
2. fork(), exec(), wait()

학습 목표

- 프로세스와 다중 처리 프로그래밍에 대해 설명할 수 있다.
- 다중 처리 프로그래밍 주요 함수와 처리 과정에 대해 설명할 수 있다.

프로세스와 다중 처리 프로그래밍

exit( ) 함수와 _exit( ) 함수의 처리



출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

따라하기 fork.c


```
static int g_var = 1;   /* data 영역의 초기화된 변수 */
char str[ ] = "PID";
int main(int argc, char **argv) {
    int var;            /* stack 영역의 지역 변수 */
    pid_t pid;
    var = 92;

    if((pid = fork()) < 0) { /* fork( ) 함수의 에러 시 처리 */
        perror("[ERROR] : fork()");
    } else if(pid == 0) {       /* 자식 프로세스에 대한 처리 */
        g_var++;                /* 변수의 값 변경 */
        var++;
```

```
printf("Parent %s from Child Process(%d) : %d\n",
                                str, getpid(), getppid());
    } else {                    /* 부모 프로세스에 대한 처리 */
        printf("Child %s from Parent Process(%d) : %d\n", str, getpid(), pid);
        sleep(1);
    }
    /* 변수의 내용을 출력 */
    printf("pid = %d, Global var = %d, var = %d\n", getpid(), g_var, var);

    return 0;
}
```

따라하기 fork.c

실행결과

```
$ gcc -o fork fork.c
$ ./fork
Child PID from Parent Process(2596) : 2597
Parent PID from Child Process(2597) : 2596
pid = 2597, Global var = 2, var = 93
pid = 2596, Global var = 1, var = 92
```

fork( ) 함수의 실행


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

static int g_var = 1; char str[] = "PID";

int main(int argc, char** argv)
{
    int var; pid_t pid; var = 92;

    if((pid = fork()) < 0) {
```

fork( ) 함수 : 2개의 프로세스로 분화

부모 프로세스 pid = 2436

```
} else {
printf("Child %s from Parent Process(%d) :
%d\n", str, getpid(), pid);
sleep(1);
}

printf("pid = %d, Global var = %d, var =
%d\n", getpid(), g_var, var);

return 0;
}
```

부모 프로세스에서 출력된 자식의 PID : 2437 pid = 2436, Global var = 1, var = 92

자식 프로세스 pid = 2437

```
} else if (pid == 0) { g_var++;
var++;
printf("Parent %s from Child Process(%d) :
%d\n", str, getpid(), getppid());
}

printf("pid = %d, Global var = %d, var =
%d\n", getpid(), g_var, var);

return 0;
}
```

자식 프로세스에서 출력한 부모의 PID : 2436 pid = 2437, Global var = 2, var = 93

따라하기 fork2.c


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

int main() {
    pid_t child_pid, grandchild_pid;

    printf("Original parent process (PID: %d) : %d\n", getpid(), getppid());

    // First fork
    child_pid = fork();
```

따라하기 fork2.c


```
if (child_pid == 0) {
    // This block will be executed by the child process
    printf("First child process (PID: %d) : %d\n", getpid(), getppid());

    // Second fork inside the child process
    grandchild_pid = fork();

    if (grandchild_pid == 0) {
        // This block will be executed by the grandchild process
        printf("Second child process (PID: %d) : %d\n", getpid(), getppid());
        // Additional processing can be done in the grandchild process.
    } else if (grandchild_pid > 0) {
```

```
    } else if (grandchild_pid > 0) {
        // This block will be executed by the child process
        // Wait for the grandchild to finish
        wait(NULL);
    } else {
        // An error occurred during the second fork
        perror("Error in grandchild fork");
    }
    // Additional processing can be done in the child process.
} else if (child_pid > 0) {
```

```
    } else if (child_pid > 0) {
        // This block will be executed by the original parent process
        // Wait for the first child to finish
        wait(NULL);
    } else {
        // An error occurred during the first fork
        perror("Error in first child fork");
    }

    // Additional processing can be done in the original parent process.
    return 0;
}
```

따라하기 fork2.c

실행결과

```
$ gcc -o fork2 fork2.c
$ ./fork2
Original parent process (PID: 660) : 455
First child process (PID: 661) : 660
Second child process (PID: 662) : 661
$
```

fork( ) 함수의 호출 후 관련 사항

- 두 프로세스가 공유하는 값
  - 실제/유효 사용자 ID, 실제/유효 그룹 ID, 보조 그룹 ID
  - 프로세스 그룹 ID, 세션 ID
  - set-user-ID 플래그, set-group-ID 플래그
  - 현재 작업 디렉터리, 루트 디렉터리
  - 파일 생성 마스크와 시그널 마스크
  - 열린 파일 디스크립터와 close-on-exec 플래그
  - 제어 터미널
  - 환경 변수와 자원 제약

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

fork( ) 함수의 호출 후 관련 사항

- 자식 프로세스에서 변경되는 값
  - fork( ) 함수의 반환 값 : 프로세스 ID, 부모 프로세스 ID
  - 부모 프로세스의 파일 잠금은 초기화
  - 처리되지 않은(pending) 알람과 시그널은 자식 프로세스에서 초기화
  - 자식 프로세스의 프로세스 진행 시간(tms_utime, tms_stime, tms_cutime, tms_cstime)의 값은 0으로 초기화

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

waitpid( ) 함수의 PID의 값


```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
pid_t waitpid(pid_t pd, int *status, int options);
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

wait() 및 waitpid() 함수는 자식 프로세스가 종료될 때까지 대기하고 종료 상태에 대한 정보를 얻는 데 사용

waitpid( ) 함수의 PID의 값

wait() 함수의 특징

01 • 하위 프로세스 중 하나가 종료될 때까지 호출 프로세스를 일시 중단

02 • 종료 상태와 같은 종료된 자식 프로세스에 대한 정보를 수집하고 종료된 자식 프로세스의 프로세스 ID(PID)를 반환

03 • 하위 프로세스가 여러 개인 경우 wait()는 종료된 하위 프로세스의 PID를 반환

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

waitpid( ) 함수의 PID의 값

waitpid() 함수의 특징

- 자식 프로세스를 기다리는 추가 옵션을 지정할 수 있음

옵션	내용	비고
pid == -1	모든 자식 프로세스를 대기함	wait( ) 함수와 같음
pid > 0	프로세스 ID가 pid인 자식 프로세스를 대기함	
pid == 0	호출한 프로세스 같은 프로세스 그룹 ID를 가진 자식 프로세스들을 대기함	
pid < -1	pid의 절댓값에 해당하는 프로세스 그룹 ID를 가진 자식 프로세스들을 대기함	

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

waitpid( ) 함수의 옵션(options) 값

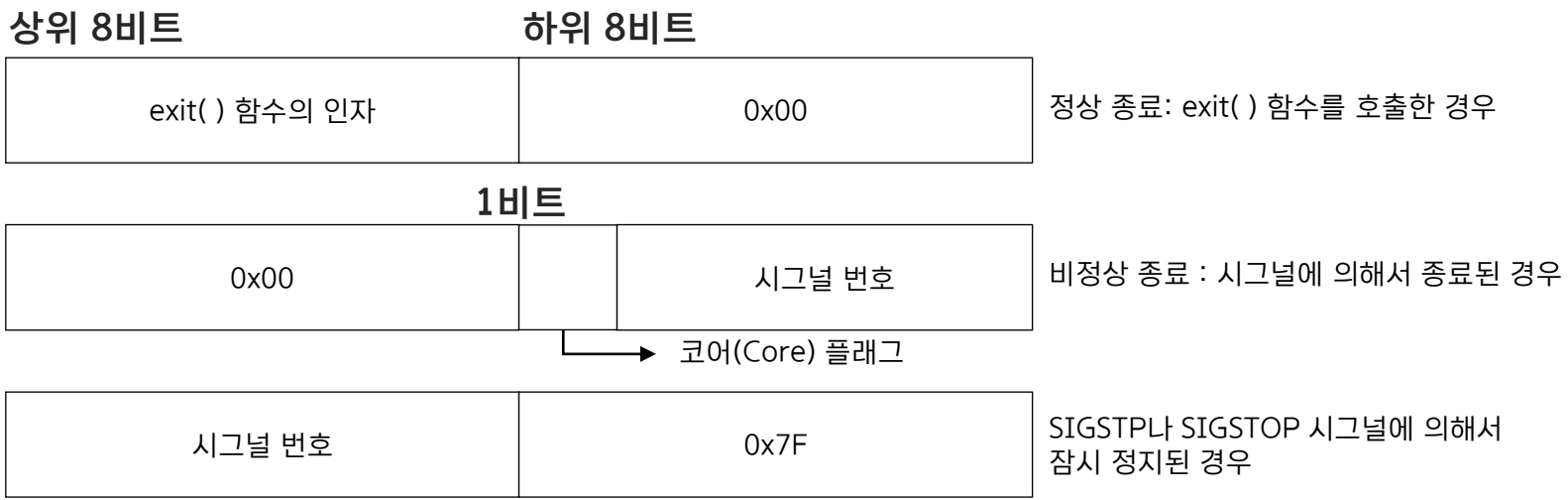

```
pid_t wait(int *status);
pid_t waitpid(pid_t pd, int *status, int options);
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

waitpid( ) 함수의 옵션(options) 값

옵션	내용
WNOHANG	pid로 지정한 프로세스가 아직 종료하지 않았더라도 대기하지 않고 바로 0의 값을 반환함
WUNTRACED	작업 제어가 지원되는 시스템에서 pid로 지정한 프로세스가 잠시 중단된 경우에도 바로 반환함

waitpid( ) 함수의 옵션(options) 값

status의 인자의 값



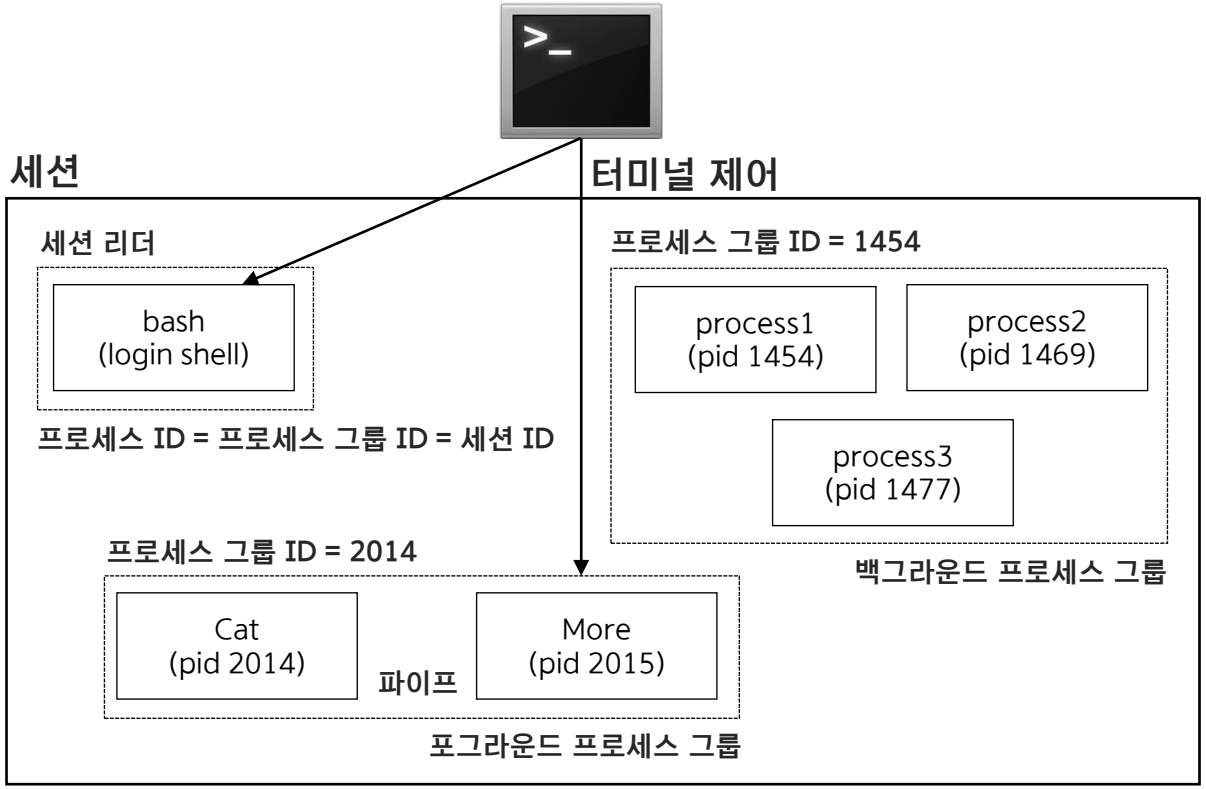
출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

프로세스 그룹과 세션


```
#include <unistd.h>
int setpgid(pid_t pid,        pid_t pgrp);
pid_t getpgid(pid_t pid);

pid_t getpgrp(void);      /* POSIX.1 버전 */
pid_t getpgrp(pid_t pid); /* BSD 버전 */

int setpgrp(void); /* System V 버전 */
int setpgrp(pid_t pid,        pid_t pgrp); /* BSD 버전 */
```

getpgrp() 및 setpgrp() 함수는 각각 프로세스의 프로세스 그룹 ID(PGID)를 가져오고 설정하는 데 사용

프로세스 그룹과 세션



출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

다른 프로그램의 실행 exec() 함수

- exec() 함수는 현재 프로세스 이미지를 새 프로그램 이미지로 바꾸는 데 사용

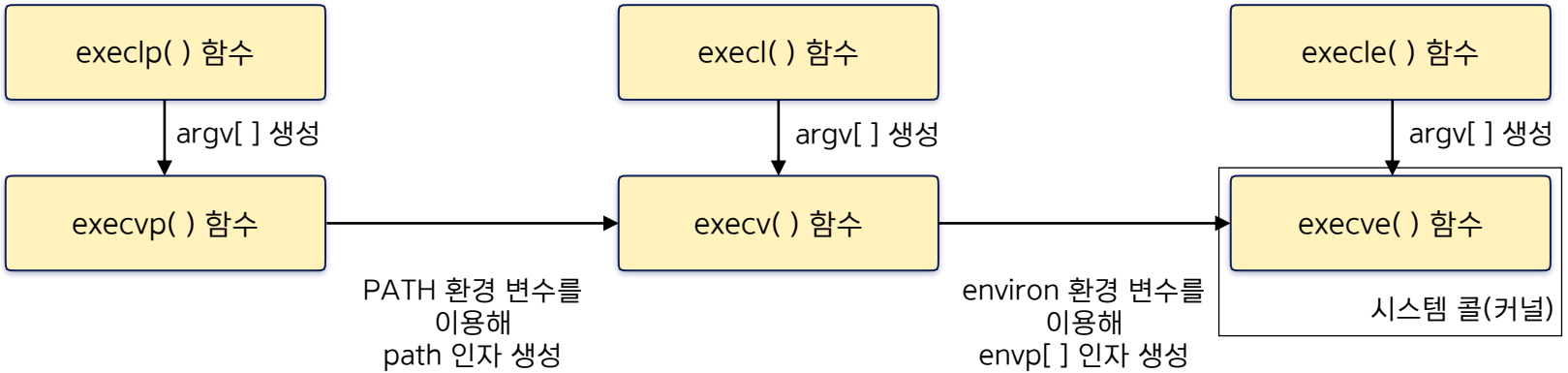
```
#include <unistd.h>

extern char **environ;
int execl(const char *pathname, const char *arg, ...
                        /*, (char *) NULL */);
int execlp(const char *file, const char *arg, ...
                        /*, (char *) NULL */);
int execle(const char *pathname, const char *arg, ...
                        /*, (char *) NULL, char *const envp[] */);
int execv(const char *pathname, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[], char *const envp[]);
```

execXXX() 함수의 접미어 의미

접미어	내용	비고
l	리스트(list) 형태의 명령 라인 인자	arg_0, arg_1, ..., arg_n
v	벡터/배열(vector) 형태의 명령 라인 인자	argv[ ]
e	환경 변수 인자	envp[ ]
p	경로에 대한 정보가 없는 실행 파일명	file

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

execve( ) 함수로의 변환



출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

exec( ) 함수의 호출 후 관련 사항

- 호출 후 변경되지 않는 값
  - 프로세스 ID, 부모 프로세스 ID, 실제 사용자 ID, 실제 그룹 ID, 보조 그룹 ID
  - 프로세스 그룹 ID, 세션 ID, 제어 터미널
  - 알람 시그널까지 남은 시간
  - 현재 작업 디렉터리, root 디렉터리
  - 파일 생성 마스크, 파일 잠금
  - 시그널 마스크, 처리되지 않은 시그널
  - 자원 제약
  - CPU 사용 시간(tms_utime, tms_stime, tms_cutime, tms_cstime)

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

exec( ) 함수의 호출 후 관련 사항

- 호출 후 변경되는 값
  - 새로 실행되는 프로그램의 set-user-ID가 세트되어 있는 경우에는 유효 사용자 ID가 프로그램 파일의 소유자 ID로 변경됨
  - 유효 그룹 ID도 새 프로그램의 set-group-ID 값에 따라 변경됨
  - 각 시그널에 대한 처리는 초기(default) 설정으로 복원되는데, 무시(ignore)되고 있던 시그널은 계속 무시됨

출처 서영진, 사물인터넷을 위한 리눅스 프로그래밍 with 라즈베리 파이, 제이펍, 2020

system() 함수

- Linux에서 system() 함수는 C 프로그램 내에서 쉘 명령을 실행

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

#include <stdlib.h>

int main() {
    int status = system(“cal");
    if (status == -1) {
        // Failed to execute the command
        // Handle the error
    } else if (WIFEXITED(status)) {
```

system() 함수

- Linux에서 system() 함수는 C 프로그램 내에서 쉘 명령을 실행

```
        int exit_status = WEXITSTATUS(status);
        // Command executed successfully
        // Process exit_status
    } else if (WIFSIGNALED(status)) {
        int signal_number = WTERMSIG(status);
        // Command terminated by a signal
        // Process signal_number
    }
    return 0;
}
```

따라하기 system.c


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

int system(const char *cmd)     /* fork(), exec(), waitpid() 함수를 사용 */
{
    pid_t pid;     int status;

    if((pid = fork()) < 0) {    /* fork( ) 함수 수행 시 에러가 발생했을 때의 처리 */
        status = -1;
    } else if(pid == 0) {       /* 자식 프로세스의 처리 */
```

```
        execl("/bin/sh", "sh", "-c", cmd, (char *)0);
        _exit(127);             /* execl( ) 함수의 에러 사항 */
    } else { /* 부모 프로세스의 처리 */
        while(waitpid(pid, &status, 0) < 0)     /* 자식 프로세스의 종료 대기 */
        if(errno != EINTR) {    /* waitpid( ) 함수에서 EINTR이 아닌 경우의 처리 */
            status = -1;
            break;
        }
    }
    return status;
}
```

따라하기 system.c

실행결과

```
$ sudo apt install ncal
$ gcc -o system system.c
$ ./system
July 2023
Su Mo Tu We Th Fr Sa
                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 31
$
```

요약정리

01 • 프로세스와 다중 처리 프로그래밍

02 • 주요 함수와 처리 과정