Signals

System Programming

실습

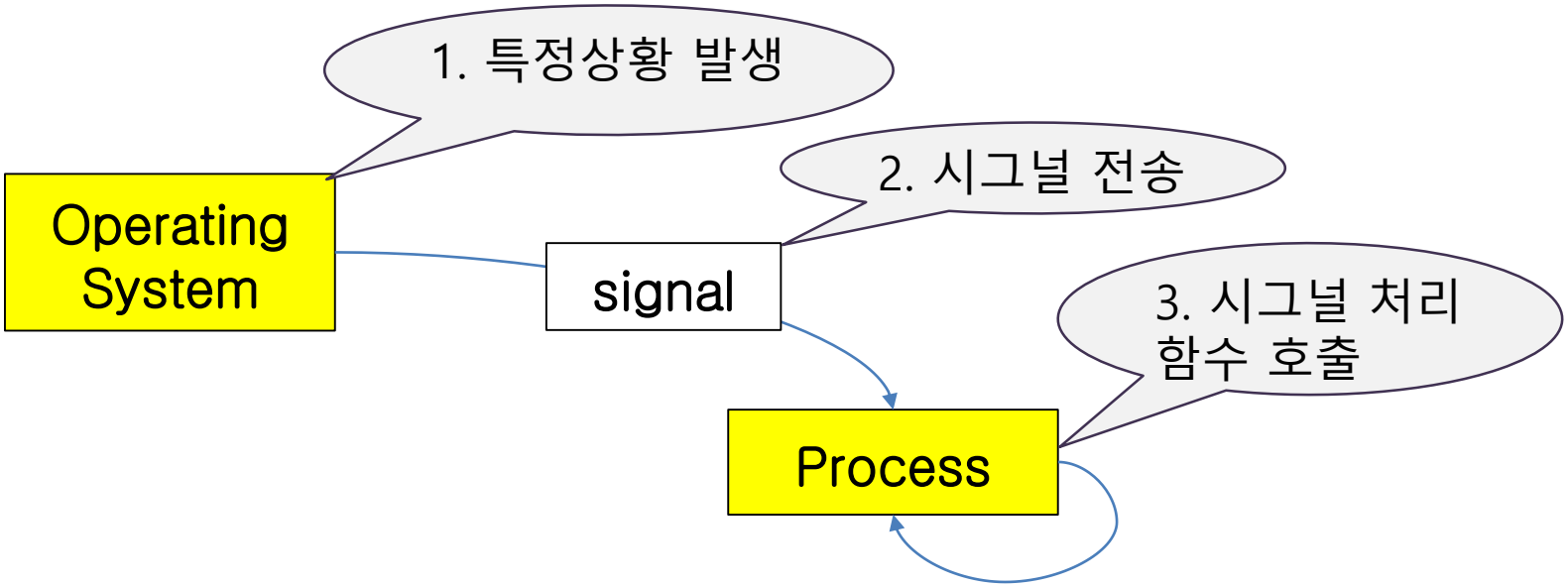
gcc 사용하기

- **예제 코드 컴파일 및 실행 해보기 (다음 슬라이드들 참고)**
  - sigint.c 코드 작성 및 실행해보기
  - sigalarm.c 코드 작성 및 실행해보기
  - z_handler.c 코드 작성 및 실행해보기
    - ✓ 슬라이드 설명 참고로 z_handler2.c로 수정 및 실행해보기

시그널 핸들링

❑ 시그널(signal)핸들링

– ‘시그널’이란 시스템에 특정 상황이 발생했을때, 이를 알리기 위해 운영체제가 전달하는 메시지이며 ‘시그널 핸들러’란 그 시그널을 처리하는 함수 모듈을 의미한다. 또한, ‘시그널 핸들링’이란 시그널이 발생했을 때, 이에 대한 시그널 핸들러를 실행시키는 행위를 의미한다.



시그널 핸들링

❑ **특정 상황과 시그널**

시그널	
SIGALRM	시간 예약(alarm함수)해놓고 그 시간이 되었을 경우 발생
SIGINT	인터럽트 발생을 알린다. 여기서 인터럽트는 Ctrl-C를 누른 경우에 발생
SIGCHLD	자식 프로세스가 종료된 경우에 발생

❑ **시그널이 발생했을 때 미리 준비해 놓은 시그널 함수(시그널 핸들러)가 호출되도록 연결해주는 작업이 필요하다**

- 이러한 일을 하는 함수는 signal함수과 sigaction함수 두 가지가 있다.
- sigaction함수를 사용하는 것이 더 명확하고 안정된 시스템의 구현을 위해 권장됨

시그널 핸들링

❑ sigaction함수

```
#include <signal.h>

int sigaction(int signum, const struct sigaction *act,  struct sigaction *oldact)
```

- Return값은 성공시 0, 실패시 -1을 리턴
- signum : signal 함수와 마찬가지로 가로채고자 하는 시그널 종류를 인자로 전달
- act : 새로 등록할 시그널 핸들러 정보로 초기화된 sigaction 구조체 변수의 포인터를 인자로 전달
- oldact : 이전에 등록되었던 시그널 핸들러의 포인터를 얻고자 할때 사용하게 되는 인자

시그널 핸들링

❑ sigaction 구조체 정의(signal.h)

```
struct sigaction
{
        void (*sa_handler)  (int)
        sigset_t   sa_mask;
        int sa_flags;
}
```

- **sa_handler** : 함수 포인터. 이곳에 시그널을 처리하는 시그널 핸들러의 포인터를 대입해준다.
- sa_mask : 시그널 핸들러 함수가 실행되는 동안에 블로킹될 시그널들을 설정하는 요소이다. 주로 모든 비트를 0로 masking한다.
- sa_flags : 시그널을 핸들링하는데 있어서 필요한 옵션을 설정하는 경우에 사용된다. 기본적으로 0으로 설정한다.

시그널 핸들링

❑ sigaction을 이용한 프로그램 작성

– 아래 그림과 같이 gedit sigint.c로 프로그램을 작성

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

void handler(int sig);

int main()
{
        int state;
        int num=0;


        struct sigaction act;
        act.sa_handler = handler;
        sigemptyset(&act.sa_mask);
        act.sa_flags=0;

        state = sigaction(SIGINT, &act, 0);
        if(state != 0){
                puts("sigaction() error");
                exit(1);
        }
```

(계속)

시그널 핸들링

❑ sigaction을 이용한 프로그램 작성

– 아래 그림과 같이 gedit sigint.c로 프로그램을 작성

```
        while(1)
        {
                printf("%d : wait \n", num++);
                sleep(2);
                if(num>5) break;
        }

        return 0;
}

void handler(int sig)
{
        printf("Signal : %d \n", sig);
}
```

(끝)

시그널 핸들링 프로그램 컴파일 및 실행

❑ gcc –o sigint sigint.c로 컴파일 및 실행

- 프로그램이 실행되면 2초 간격으로 “wait”를 출력하면서 5번 출력 후 종료
- 이때 중간에 Ctrl-C를 입력하게 되면 프로그램에서 정의한 시그널 핸들러가 호출됨

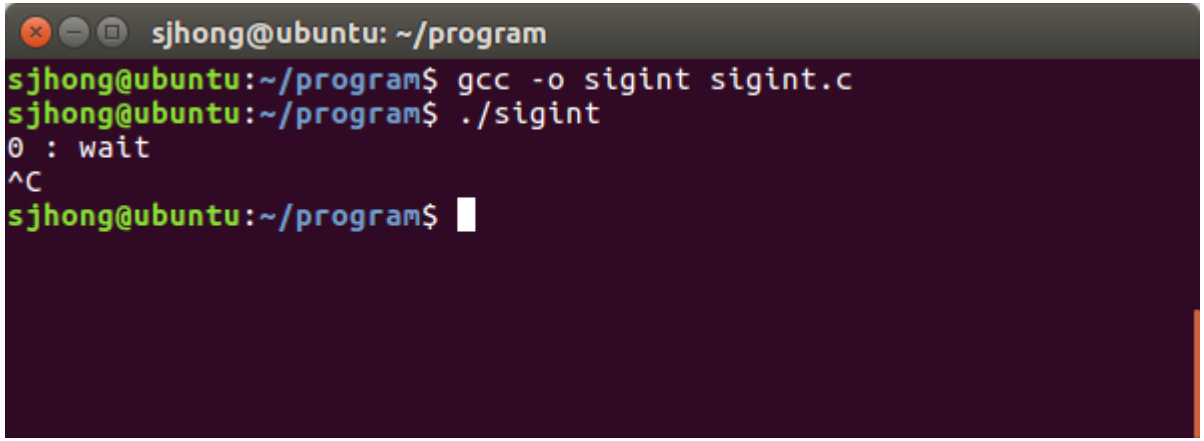
```
sjhong@ubuntu: ~/program
3 : wait
^CSignal : 2
4 : wait
5 : wait
sjhong@ubuntu:~/program$
sjhong@ubuntu:~/program$ gedit sigint.c
sjhong@ubuntu:~/program$ gedit sigint.c
sjhong@ubuntu:~/program$ gedit sigint.c
^C
sjhong@ubuntu:~/program$ gcc -o sigint sigint.c
sjhong@ubuntu:~/program$ ./sigint
0 : wait
1 : wait
^CSignal : 2
2 : wait
3 : wait
^CSignal : 2
4 : wait
5 : wait
^CSignal : 2
sjhong@ubuntu:~/program$
```

시그널 핸들링 프로그램 컴파일 및 실행

❑ sigaction함수 호출을 주석 처리 후 재실행해보기

- 이때 중간에 Ctrl-C를 입력하게 되면, 프로그램이 종료
- 핸들러를 설정해주지 않았다고 해서 시그널이 발생되지 않는 것은 아님. 시그널은 발생되지만, 우리가 직접 핸들러를 설정해주지 않으면, 운영체제 차원에서 지니고 있는 기본적인 핸들러가 동작됨.

✓ 운영체제에서 기본적인 SIGINT 시그널에 대한 핸들러는 프로세스를 그냥 종료시킴



시그널을 통한 ALARM기능 구현

❑ alarm함수

```
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
```

– seconds : SIGALRM 시그널 발생을 초단위로 예약. 이전에 이미 예약한 알람을 취소하기 위해 0을 전달할 수 있음.

시그널을 통한 ALARM기능 구현

❑ gedit sigalarm.c로 프로그램을 작성 (그림 참조)

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

void timer(int sig);

int main()
{
        int state;

        struct sigaction act;
        act.sa_handler = timer;
        sigemptyset(&act.sa_mask);
        act.sa_flags=0;

        state = sigaction(SIGALRM, &act, 0);
        if(state != 0){
                puts("sigaction() error");
                exit(1);
        }

        alarm(5);
```

(계속)

시그널을 통한 ALARM기능 구현

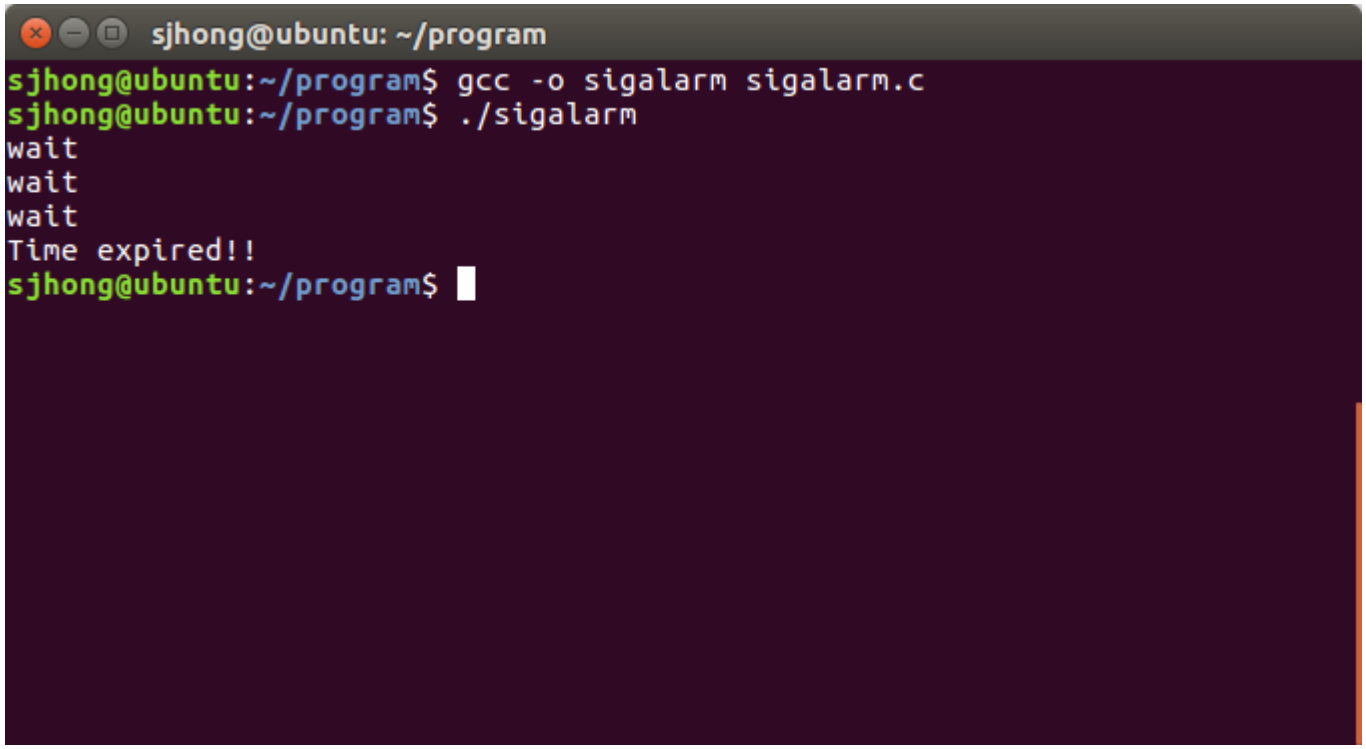
❑ gedit sigalarm.c로 프로그램을 작성 (그림 참조)

```
        while(1)
        {
                puts("wait");
                sleep(2);
        }

        return 0;
}

void timer(int sig)
{
        printf("Time expired!!\n");
        exit(0);
}
```

(끝)

시그널을 통한 ALARM기능 구현

- **gcc –o sigalarm sigalarm.c로 컴파일 및 실행**
  - 프로그램이 실행되면 5초후 알람이 발생되는 것을 확인



시그널을 통한 좀비 프로세스의 소멸

❑ gedit z_handler.c로 프로그램을 작성 (그림 참조)

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

void z_handler(int sig);

int main()
{
        int state;
        int num=0;
        pid_t pid;

        struct sigaction act;
        act.sa_handler = z_handler;
        sigemptyset(&act.sa_mask);
        act.sa_flags=0;

        state = sigaction(SIGCHLD, &act, 0);
        if(state != 0){
                puts("sigaction() error");
                exit(1);
        }
```

(계속)

시그널을 통한 좀비 프로세스의 소멸

❑ gedit z_handler.c로 프로그램을 작성 (그림 참조)

```
        pid=fork();
        if(pid == 0){
                printf("Generate child process : %d\n", getpid());
                exit(3);
        }
        else
        {
                sleep(3);
        }

        return 0;
}

void z_handler(int sig)
{
        pid_t pid;
        int rtn;

        while((pid= waitpid(-1, &rtn, WNOHANG)) > 0)    {
                printf("Destroyed zombie process ID : %d \n", pid);
                printf("Returned data : %d\n\n", WEXITSTATUS(rtn));
        }

}
```

(끝)

시그널을 통한 좀비 프로세스의 소멸

❑ gcc -o z_handler z_handler.c로 컴파일 후 실행

```
sjhong@ubuntu: ~/program
sjhong@ubuntu:~/program$ gedit z_handler1.c
sjhong@ubuntu:~/program$ gcc -o z_handler1 z_handler1.c
sjhong@ubuntu:~/program$ ./z_handler1
Generate child process : 4066
Destroyed zombie process ID : 4066
Returned data : 3

sjhong@ubuntu:~/program$
```

시그널을 통한 좀비 프로세스의 소멸2

❑ **gedit z_handler2.c로 프로그램을 작성 (그림 참조)**

- 이 프로그램의 경우 시그널 핸들러를 SIG_IGN (시그널 무시)로 해놓아 자식 프로세스가 종료되더라도 좀비 프로세스가 생성되지 않음
- 이 방식은 자식 프로세스의 상태를 가져올 수 없는 단점이 있지만, 깔끔한 처리가능

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

int main()
{
        int state;
        int num=0;
        pid_t pid;

        struct sigaction act;
        act.sa_handler = SIG_IGN;
        sigemptyset(&act.sa_mask);
        act.sa_flags=0;

        state = sigaction(SIGCHLD, &act, 0);
        if(state != 0){
                puts("sigaction() error");
                exit(1);
        }
```

(계속)

시그널을 통한 좀비 프로세스의 소멸2

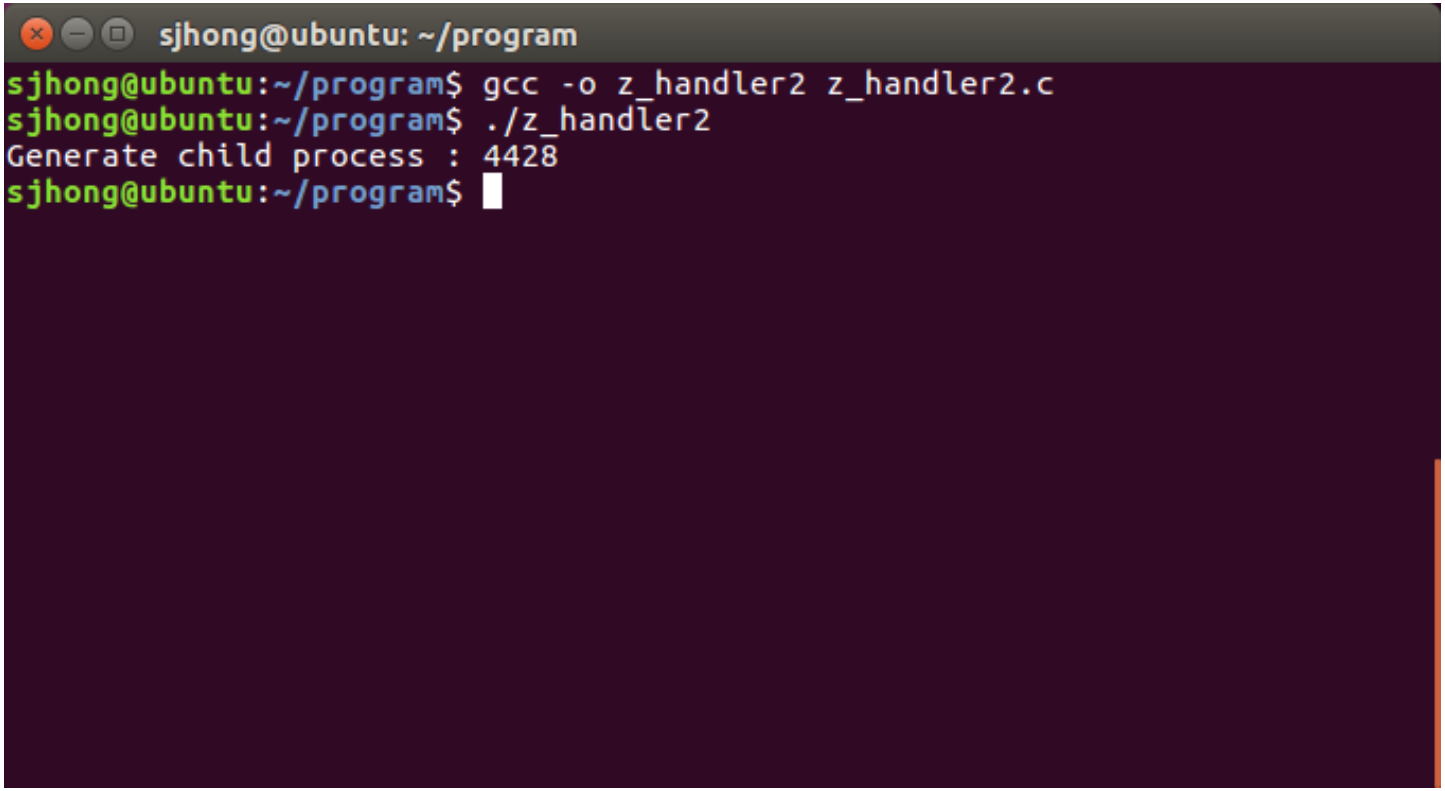
❑ gedit z_handler2.c로 프로그램을 작성 (그림 참조)

```
        pid=fork();
        if(pid == 0){
                printf("Generate child process : %d\n", getpid());
                exit(3);
        }
        else
        {
                sleep(3);
        }

        return 0;
}
```

(끝)

시그널을 통한 좀비 프로세스의 소멸

❑ gcc -o z_handler2 z_handler2.c로 컴파일 후 실행



Q and A