

다중 접속 서버에 대한 이해 및 다중 프로세스 함수 실습

인하공업전문대학 컴퓨터정보과
최효현 교수

주요사항

- ❑ 다중 접속 서버 개념
- ❑ Fork를 이용한 멀티 프로세스 생성 이해
- ❑ 좀비 프로세스 이해
- ❑ 시그널 핸들링 방법 이해
- ❑ Fork를 이용한 다중 접속 서버 구현 실습

10.1 프로세스의 이해와 활용

1. 다중접속 서버란?

- 여러 클라이언트들이 동시에 접속할 수 있는 서버
(서비스를 동시에 받을 수 있는 서버)

2. 리눅스 기반의 다중접속 서버구현 방법

- 프로세스 생성을 통한 멀티태스킹(Multitasking) 서버의 구현
- **select** 함수에 의한 멀티플렉싱(Multiplexing) 서버의 구현
- 쓰레드를 기반으로 하는 멀티쓰레딩(Multithreading) 서버의 구현

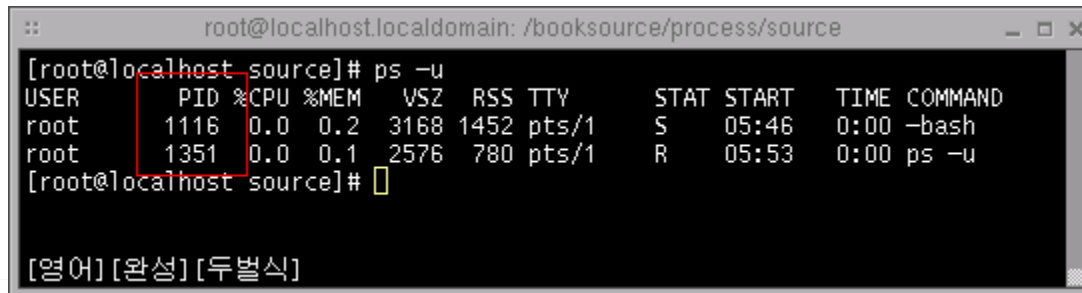
10.1 프로세스의 이해와 활용

1. 프로세스(Process)에 대한 이해

- 프로세스란 실행되고 있는 프로그램의 기본 단위이다 – OS 용어임
- 생성된 프로세스는 운영체제의 의해 할당된 고유한 ID를 지닌다.
- 하나의 프로그램 내에서 여러 개의 프로세스가 동시에 실행 될 수 있다.

2. 프로세스 ID

- OS로 부터 할당되는 유일한 프로세스 식별자로, 2 – 32768의 값을 가짐
- ID “1” : init 프로그램에 할당(운영체제가 시작되자마자 실행됨)
- PS 명령을 사용 프로세스 확인



```
root@localhost.localdomain: /booksource/process/source
[root@localhost source]# ps -u
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1116  0.0  0.2  3168  1452 pts/1    S    05:46   0:00 -bash
root      1351  0.0  0.1  2576   780 pts/1    R    05:53   0:00 ps -u
[root@localhost source]#
```

[영어] [완성] [두벌식]

[그림 10-1]

10.1 프로세스의 이해와 활용

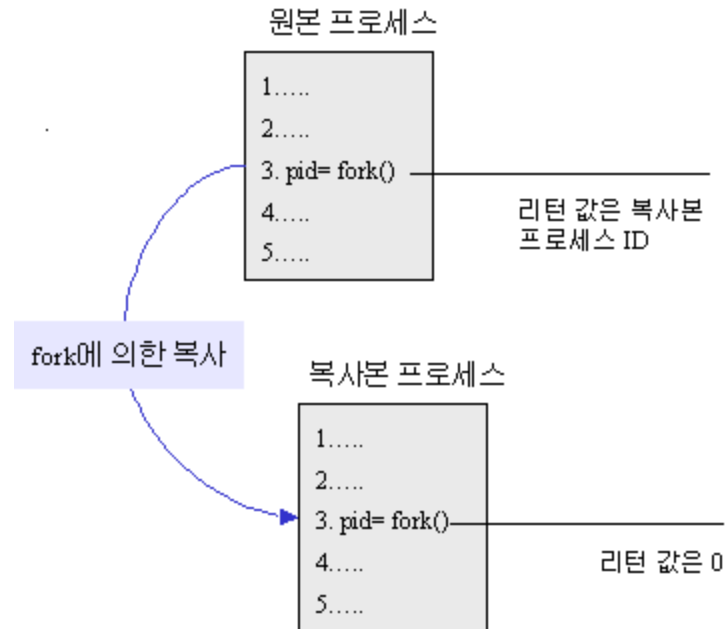
• fork 함수 호출을 통한 프로세스의 생성

1. fork 함수 호출을 통한 프로세스의 생성은 복사에 의한 생성이다.
즉, 복사본 프로세스를 생성함
2. 성공시에는 원본 프로세스와 복사본 프로세스에 리턴되는 값이 달라짐
이를 이용하여 프로세스의 흐름을 둘로 나눔

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

pid_t fork(void);
```

성공시 : process ID 리턴
실패시 : -1을 리턴

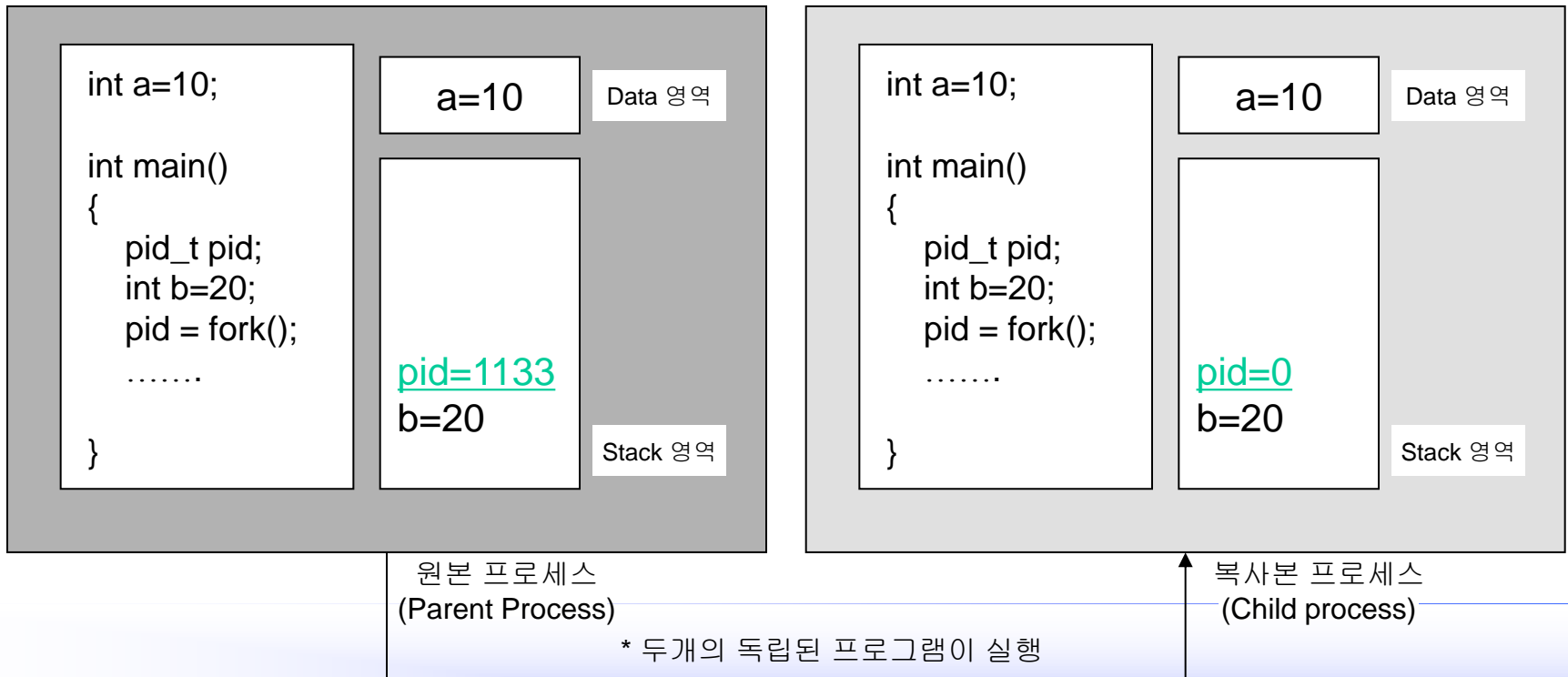


[그림 10-2]

10.1 프로세스의 이해와 활용

• fork 함수 호출을 통한 프로세스의 생성

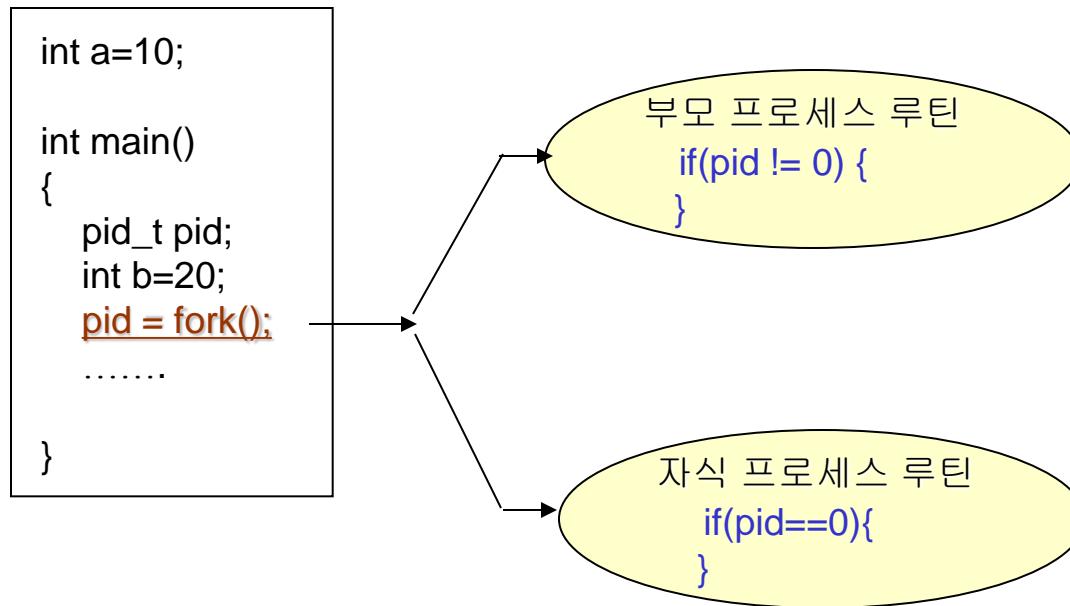
- 함수 호출이 성공하면, 복사본에 해당하는 프로세스는 원본 프로세스의 모든 메모리 공간의 영역(힙(heap), 스택(stack)) 등을 그대로 복사함
각각이 독립적인 프로그램으로 실행됨



10.1 프로세스의 이해와 활용

- fork 함수 호출을 통한 프로세스의 생성

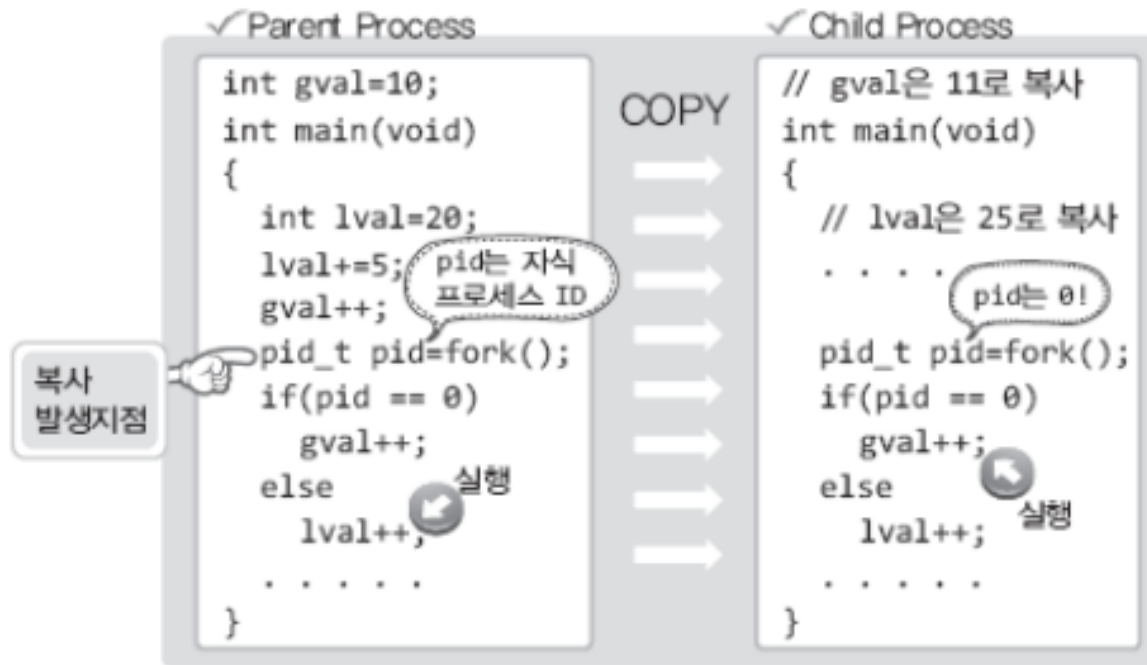
- fork() 함수 이루에 2개의 흐름이 존재



10.1 프로세스의 이해와 활용

• 예제 확인 1

프로그램 예제
- fork.c



10.2 프로세스 & 좀비(Zombie) 프로세스

• 좀비 프로세스(zombie process)

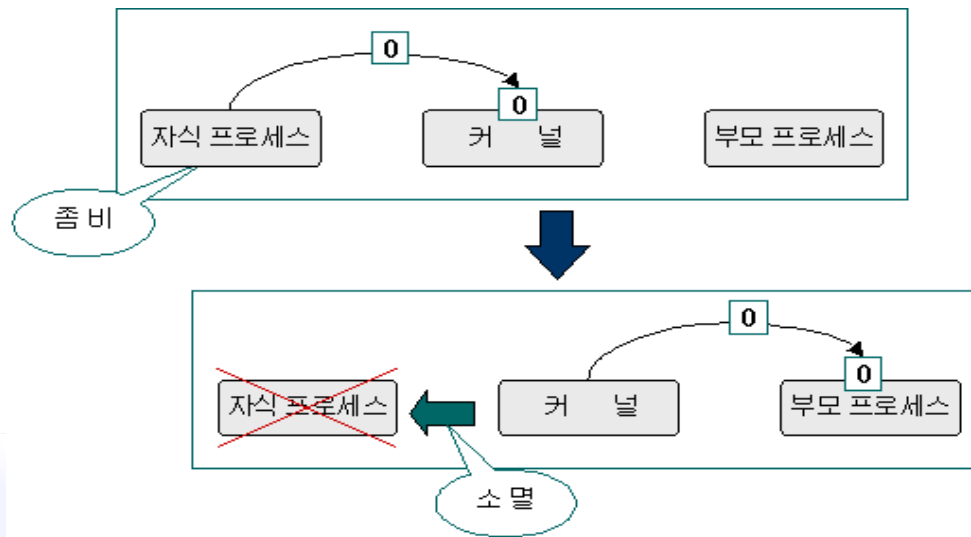
1. 좀비 프로세스란?

- 프로세스 종료 후 메모리상에서 사라지지 않는 프로세스
- 시스템의 리소스를 점유하여 성능을 저하시킴

2. 좀비 프로세스의 생성 이유.

- 커널은 비록 자식 프로세스가 종료되었더라도 리턴값을 부모 프로세스에 넘겨줄 때까지 자식 프로세스를 소멸시키지 않음
- 자식 프로세스는 부모 프로세스에게 실행 결과에 대한 값을 반환해야 한다.

(부모 프로세스가 임의의 함수 호출을 통해서 커널에 리턴 값을 전달해 달라고 요청해야 함)



10.2 프로세스 & 좀비(Zombie) 프로세스

- 좀비 프로세스의 생성 예

프로그램 예제 : 부모 프로세스가 자식의 리턴값을 읽어들이지 않음, 자식은 좀비가 됨
- zombie.c

```
root@my_linux:/tcpip# ps au
USER      PID   %CPU %MEM    VSZ   RSS  TTY      STAT START   TIME COMMAND
. . . . .
root      10976  0.0   0.0   1628    368 pts/0    S+   20:26   0:00 ./zombie
root      10977  0.0   0.0      0      0 pts/0    Z+   20:26   0:00 [zom] <defunct>
. . . . .
```

10.2 프로세스 & 좀비(Zombie) 프로세스

• 좀비 프로세스의 소멸1

1. 소멸 방법

- 부모 프로세스에서 자식 프로세스의 반환 값을 요구한다.

2. wait() 함수의 사용

- 장점 : 사용하기 간단하다.
- 단점 : 무한 대기 상태에 빠질 수 있다.

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

pid_t wait(int * status)
```

성공시 : 종료된 자식 프로세스 ID

실패시 : -1

종료상태를 확인할 수 있는 매크로 함수

WIFEXITED(status) 정상 종료시 0을 리턴

WEXITSTATUS(status) 종료시 리턴 하거나 **exit**함수의 인자로 넘겨진 값을 반환

10.2 프로세스 & 좀비(Zombie) 프로세스

- 좀비 프로세스의 소멸 예제 1

프로그램 예제

- wait.c

```
root@my_linux:/tcpip# gcc wait.c -o wait
root@my_linux:/tcpip# ./wait
Child PID: 12337
Child PID: 12338
Child send one: 3
Child send two: 7
```

실행 결과

- WIFEXITED

자식 프로세스가 정상 종료한 경우 '참(true)'을 반환한다.

- WEXITSTATUS

자식 프로세스의 전달 값을 반환한다.

10.2 프로세스 & 좀비(Zombie) 프로세스

• 좀비 프로세스의 소멸2

1. 소멸 방법

- 부모 프로세스에서 자식 프로세스의 반환 값을 요구한다.

2. waitpid 함수의 사용

- wait 함수가 지니고 있는 무한 대기 상태의 문제점을 해결.

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

```
pid_t waitpid(pid_t pid, int * status, int options)
```

성공시 : 종료된 자식 프로세스 ID

실패시 : -1

pid : 종료하기를 원하는 자식프로세스의 id

status : 리턴되는 정보를 얻음

options : 종료한 자식 프로세스가 없는 경우, 대기 상태 없이 즉시 리턴(WNOHANG 상수를 사용)

10.2 프로세스 & 좀비(Zombie) 프로세스

• 좀비 프로세스의 소멸2

부모 프로세스에서

```
do {  
    sleep(3);  
    puts("3초 대기");  
    child = waitpid(-1, &state, WNOHANG);  
}while(child == 0)
```

이미 종료한 자식 프로세스가 없으면 대기없이 즉시 0을 리턴

리턴되는 여러정보중에서 종료 리턴값을 확인

임의의 자식 프로세스 종료를 기다림

child가 가질 수 있는 값:

pid : 종료한 자식 프로세스 pid
0 : 이미 종료한 자식 프로세스가 없을 경우
-1 : 비정상적인 종료

10.2 프로세스 & 좀비(Zombie) 프로세스

- 좀비 프로세스 소멸의 예제 2

1. 프로그램 예제

- waitpid.c

```
root@my_linux:/tcpip# gcc waitpid.c -o waitpid
root@my_linux:/tcpip# ./waitpid
sleep 3sec.
sleep 3sec.
sleep 3sec.
sleep 3sec.
sleep 3sec.
Child send 24
```

실행 결과

10.3 시그널 핸들링

1. 좀비 프로세스 소멸 방법을 이해
2. 자식 프로세스가 언제 종료될 줄 알고 `waitpid` 함수를 호출할 것인가?
3. 당장 블록킹 문제는 해결, 그러나 계속 루프로 확인은 좋지 않은 방법임

=> 이상적인 방법

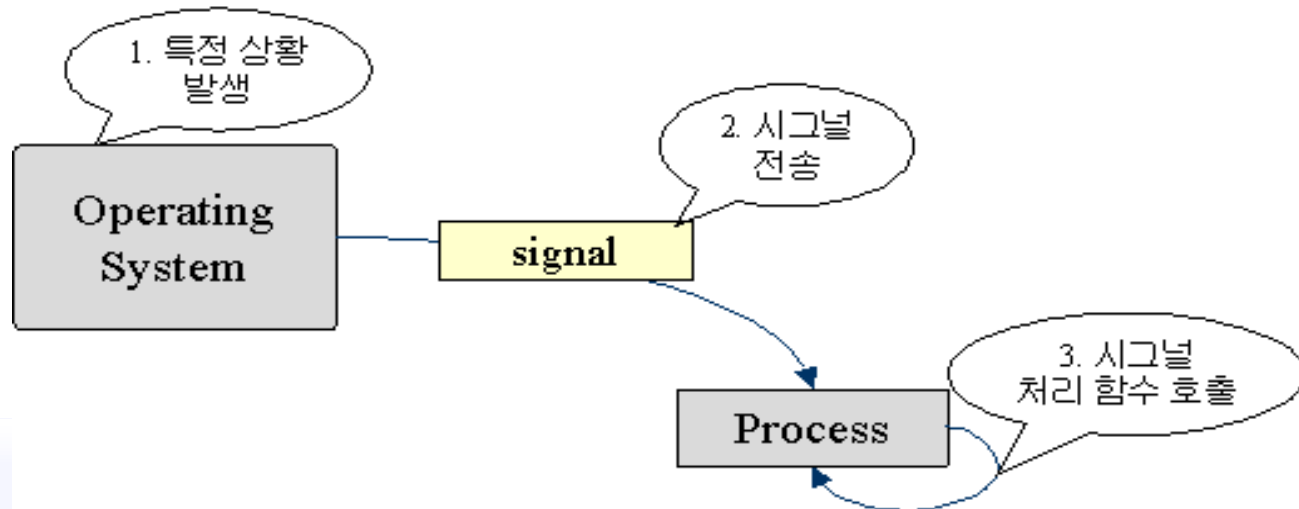
커널이 부모에게 자식이 종료되었음을 알려줘서
이에 따른 처리를 구현함

“시그널 핸들링” 방법으로 이를 구현함

10.3 시그널 핸들링

• 시그널(Signal) 핸들링

1. 시그널이란?
 - 시스템 내의 특정상황 발생을 알리기 위해서 커널이 전달하는 신호
2. 시그널 핸들러
 - 적절한 처리를 해 주는 함수.
3. 시그널 핸들링
 - 시그널이 발생 함에 따라 이에 대한 적절한 처리를 해 주는 것.



10.3 시그널 핸들링

- 시그널(Signal)의 종류

시그널	발생 상황
SIGALRM (14)	시간을 예약(alarm 함수 사용)해 놓고 그 시간이 되었을 경우 발생.
SIGINT (2)	인터럽트(interrupt) 발생을 알린다. 여기서 인터럽트는 Ctrl-C를 누른 경우 발생한다.
SIGCHLD	자식 프로세스가 종료된 경우 발생한다.

[표 10-2]

10.3 시그널 핸들링

- signal 함수를 이용한 시그널 핸들링

1. signal 함수

- 시그널과 시그널 핸들러를 연결해 주는 기능을 한다.

```
#include <signal.h>
```

```
void (*signal(int signum, void (*func)(int)))(int);
```

signum : 프로세스가 가로채고자 하는 시그널 상수

func : 시그널을 처리할 함수의 포인터

기능 : **signum**인자로 전달되는 시그널이 발생하면, **func**인자로 전달된 포인터가 가리키는 함수가 호출되도록 설정하는 것을 담당

10.3 시그널 핸들링

- 예제 확인 1

프로그램 예제
- signal.c

```
root@my_linux:/tcpip# gcc signal.c -o signal
root@my_linux:/tcpip# ./signal
wait...
Time out!
wait...
Time out!
wait...
Time out!
```

실행 결과

10.3 시그널 핸들링

• sigaction 함수를 이용한 시그널 핸들링

1. sigaction 함수

- 더 명확하고 안정된 시스템 구현을 위한 방법임
- 시그널과 시그널 핸들러를 연결해 주는 기능을 한다.

```
#include <signal.h>
```

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

성공시 : 0, 실패시 : -1을 리턴

signum : 프로세스가 가로채고자 하는 시그널 상수

act : 새로 등록할 시그널 핸들러 정보로 초기화된 sigaction 구조체 변수

oldact : 이전에 등록되었던 시그널 핸들러의 => 포인터를 얻고자 할때 사용

```
struct sigaction
```

```
{
```

```
    void (*sa_handler)(int) // 함수 포인터
```

```
    sigset_t sa_mask;        // 블로킹될 시그널 요소설정, 보통은 0으로 마스킹함
```

```
    int sa_flags;            // 필요한 옵션
```

```
}
```

10.3 시그널 핸들링

- 예제 확인 2

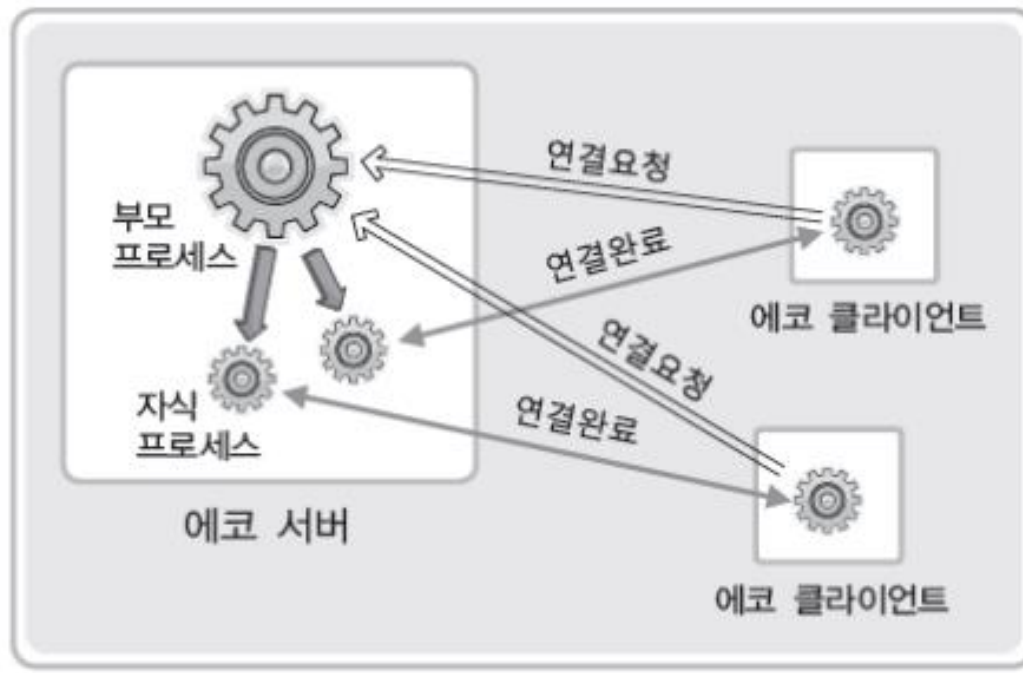
프로그램 예제
- sigaction.c

```
root@my_linux:/tcpip# gcc sigaction.c -o sigaction
root@my_linux:/tcpip# ./sigaction
wait...
Time out!
wait...
Time out!
wait...
Time out!
```

10.4 멀티태스킹 기반의 다중접속 서버

• 프로세스 기반 다중 접속 서버의 구현 모델

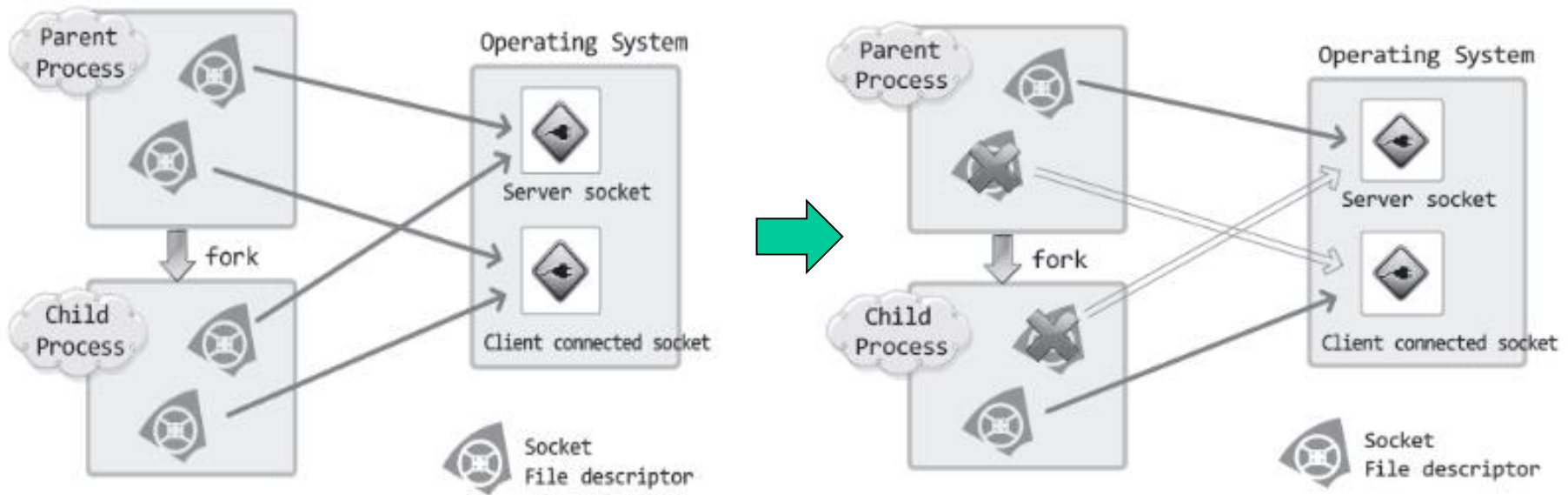
1. 여러 클라이언트를 동시에 접속 가능하도록 **echo** 서버를 변경
2. 새로운 연결 요청을 수락 할 때마다 프로세스의 생성.
3. 이 때 생성된 파일 기술자를 새로운 자식 프로세스에 전달, 데이터 송수신을 담당케 함



10.4 멀티태스킹 기반의 다중접속 서버

• 파일 디스크립터의 복사

1. 하나의 소켓(파일)에 대한 파일 디스크립터가 둘 이상 존재하는 경우, 모든 파일 디스크립터를 종료 해 줘야 해당 소켓(파일)이 종료 된다.



10.4 멀티태스킹 기반의 다중접속 서버

- 예제 확인

프로그램 예제

- `echo_mpserv.c` (서버 프로그램, 이전의 에코 클라이언트 사용)

Q&A

