Unix Process

NETWORKING LAB

Department of Computer Engineering

Kyung Hee University.

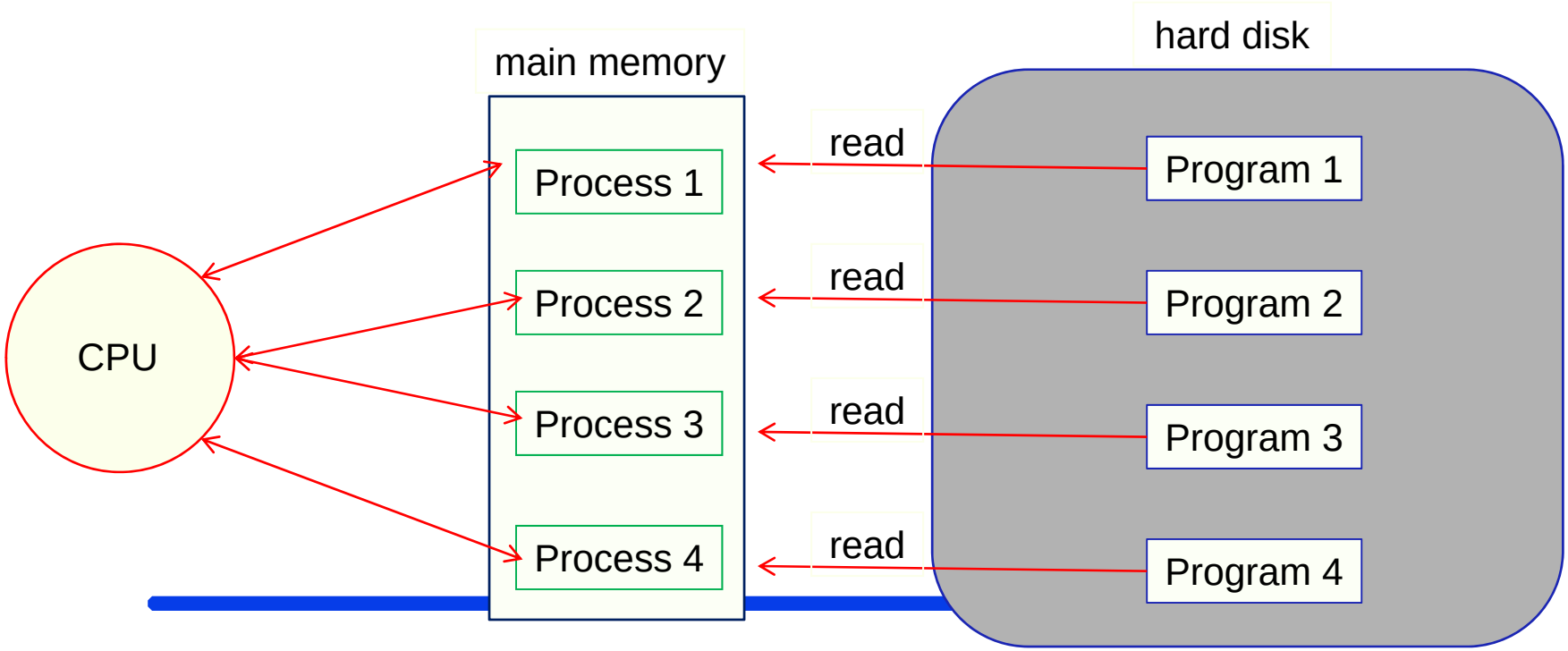
Choong Seon Hong

유닉스 기반 다중 서버 구현 방법

- 클라이언트들이 동시에 접속할 수 있는 서버
- 서비스를 동시에 처리할 수 있는 서버
  - 프로세스 생성을 통한 멀티태스킹 (Multitasking) 서버의 구현
  - select 함수에 의한 멀티플렉싱 (Multiplexing) 서버의 구현
  - 쓰레드를 기반으로 하는 멀티쓰레딩 (Multithreading) 서버의 구현

Process의 정의

- Program 이란
  - C, C++, Java와 같은 프로그램 언어로 작성된 소스코드
- Process 란
  - Running program
  - 생성된 프로세스는 운영체제의 의해 할당된 고유한 ID를 지닌다. (ps –u)
  - 하나의 프로그램 내에서 여러 개의 프로세스가 동시에 실행 될 수 있다.



Process의 모드

- Process는 User mode 또는 Kernel mode에서 실행
  - 사용자 모드
    - 사용자의 권한으로 명령이 실행
  - 커널 모드
    - 커널의 권한으로 실행
    - 하드디스크를 읽기 위한 read() 함수

```
void main()
{
        char buf[512];

        int n = read(0, buf, 512};

        n++;

        exit(0);
}
```

User mode에서 실행

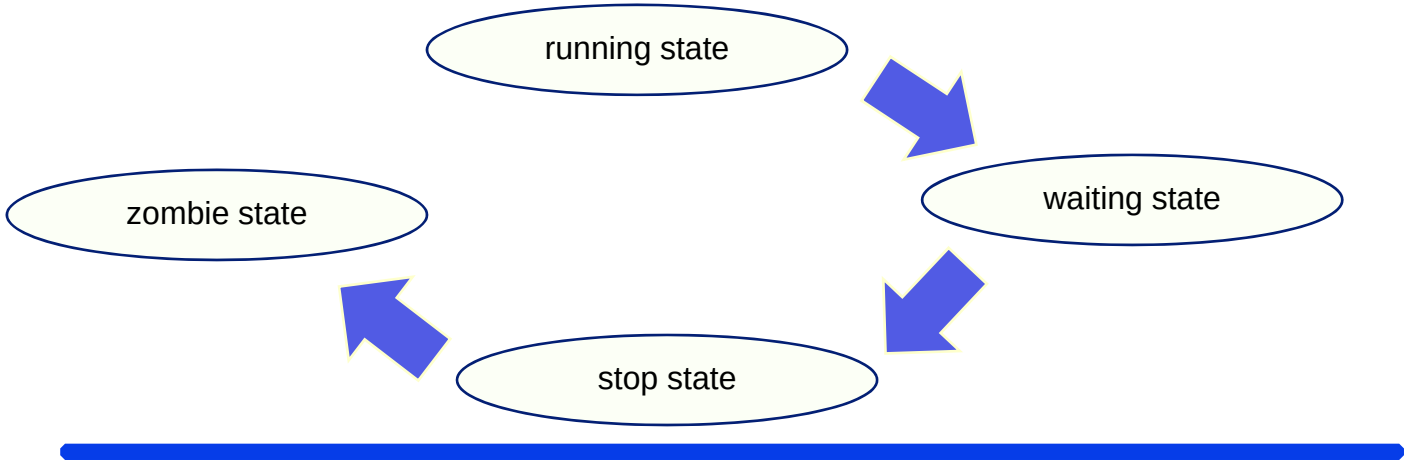
Kernel mode에서 실행

User mode에서 실행

Kernel mode에서 실행

Process의 상태

- Running state (실행 상태)
  - 프로세스가 CPU 서비스를 받을 수 있는 상태
- Waiting state (블록 상태)
  - 입출력 처리와 같은 어떤 조건을 기다리는 상태
- Stop state (중단 상태)
  - 특정 시그널을 커널로부터 받아 프로세스의 동작이 정지된 상태
- Zombie state (좀비 상태)
  - 프로세스의 실행은 끝났으나 여전히 메모리에 존재하는 상태



Process의 메모리 배치

- 프로세스는 일정한 메모리를 배정 받아 사용 (프로세스 이미지)
  - 프로그램 실행에 필요한 어셈블러 코드, 변수가 저장
  - 원칙적으로 한 프로세스는 다른 프로세스의 메모리 영역에 접근 불가
- C 프로그램과 이미지의 내용

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

extern char **environ;          //extern 변수
int init_global_var = 3;        //초기화된 global 변수
int unint_global_var;           //초기화되지 않은 global 변수

int main(int argc, char **argv)
{
        int auto_var;           //자동 변수
        static int static_var;  //static 변수
        register int reg_var;   //register 변수
        char *auto_ptr;         //자동 변수
        auto_ptr = malloc(10);  //메모리 10byte 할당
        return 0;
}
```

메모리 영역		메모리 내용 및 변수들
환경변수와 명령행 인자 영역		*environ 내용
stack		argc, argv, auto_var, reg_var, auto_ptr
heap		malloc()이 할당한 10 byte
데이터 영역	초기화 안 된 영역	unint_global_var, static_var
	초기화 된 영역	init_global_var = 3
코드영역		어셈블된 프로그램 코드

Stack and Heap

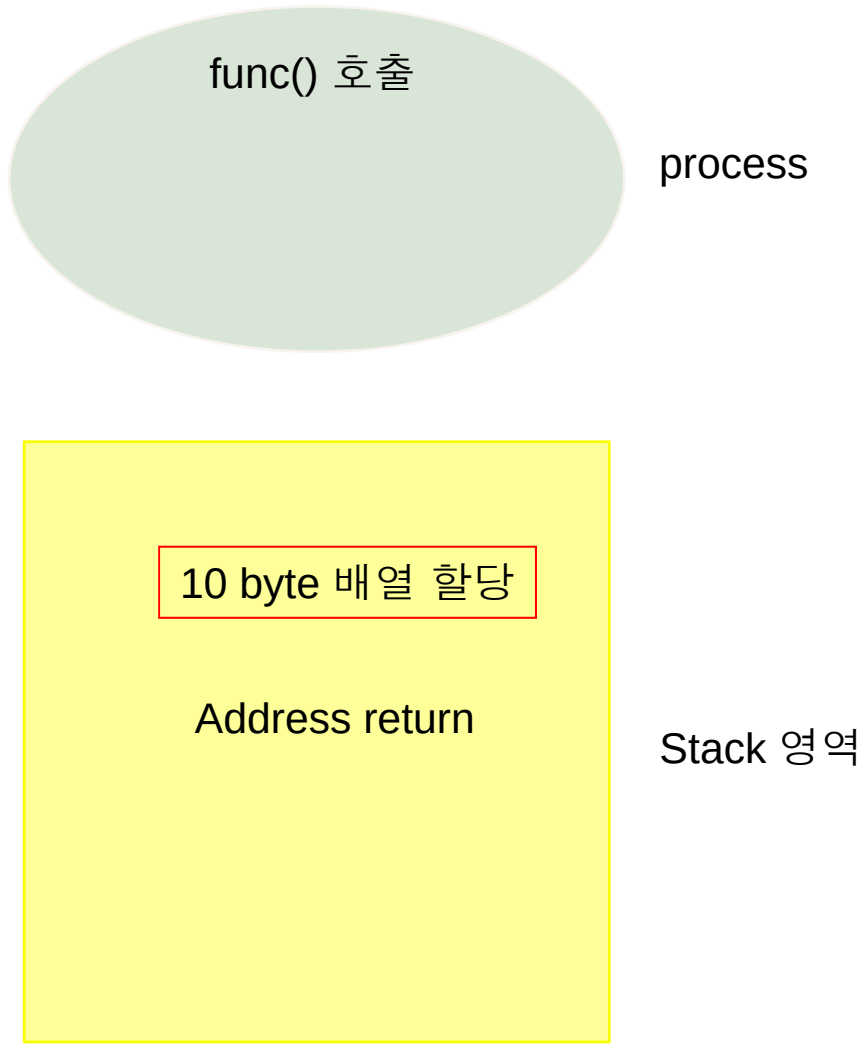
- Stack
  - 현재 호출되어 실행중인 함수의 코드와 환경 정보를 저장
    - main()에서 printf()를 호출하였다면 main()이 차지하는 영역위에 printf()를 처리하기 위한 메모리가 할당되고 printf()가 수행
    - printf()가 리턴되면 printf()와 관련된 영역은 삭제되고 프로세스의 수행은 main()으로 돌아감
  - 함수 내부에서 임시로 사용되는 자동 변수도 스택 영역에 할당
- Heap
  - 스택은 사용하던 메모리가 함수의 종료와 함께 사라짐
  - 이 문제를 해결하기 위해 리턴 되어도 사라지지 않도록 한 영역이 힙
  - malloc() 함수를 사용한 영역은 힙에 저장

Stack

```
int main() {
        ch
        '
}

char '
        arr[10];
        rn arr;
}
```

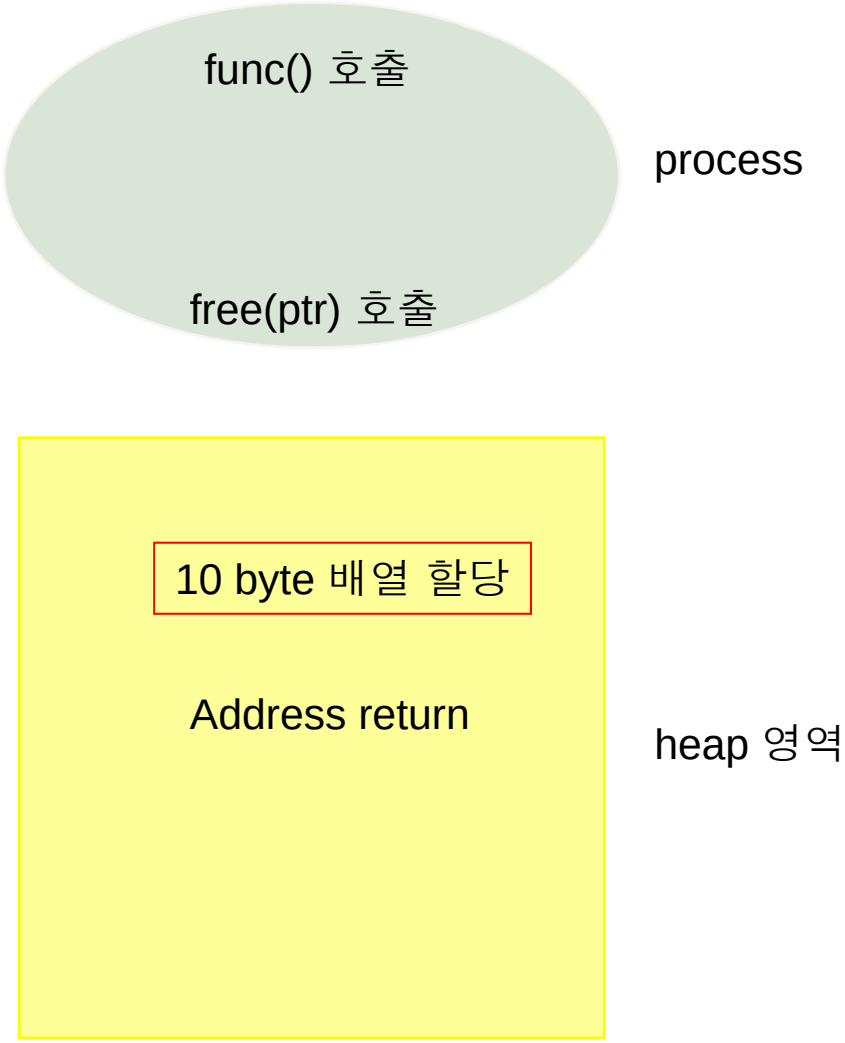
Error



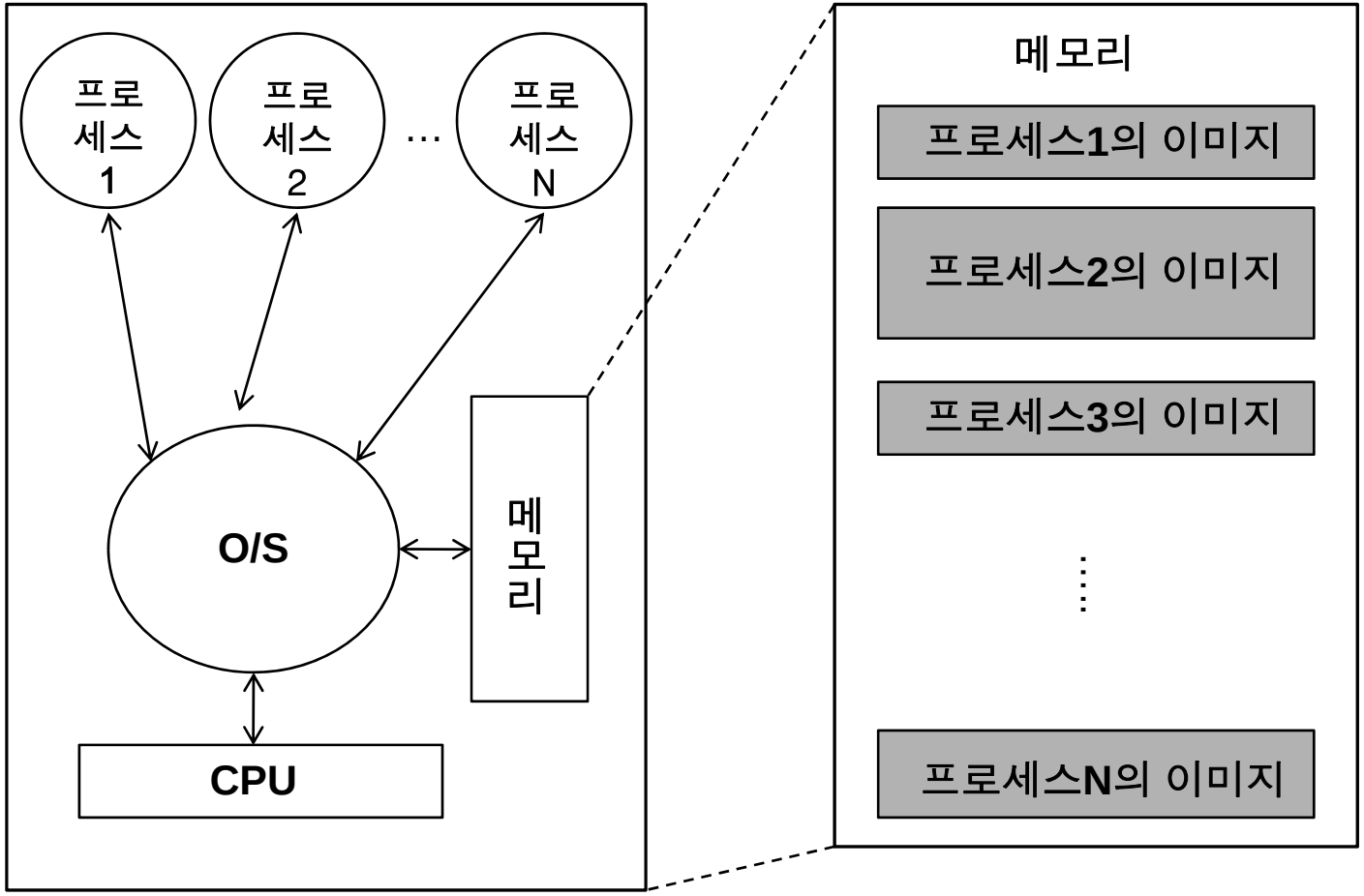
Heap

```
int main() {
        char *ptr;
        ptr = func();
        free(ptr)
}

char *func() {
        char *arr;
        arr = malloc(10);
        return arr;
}
```



Process의 생성과 종료

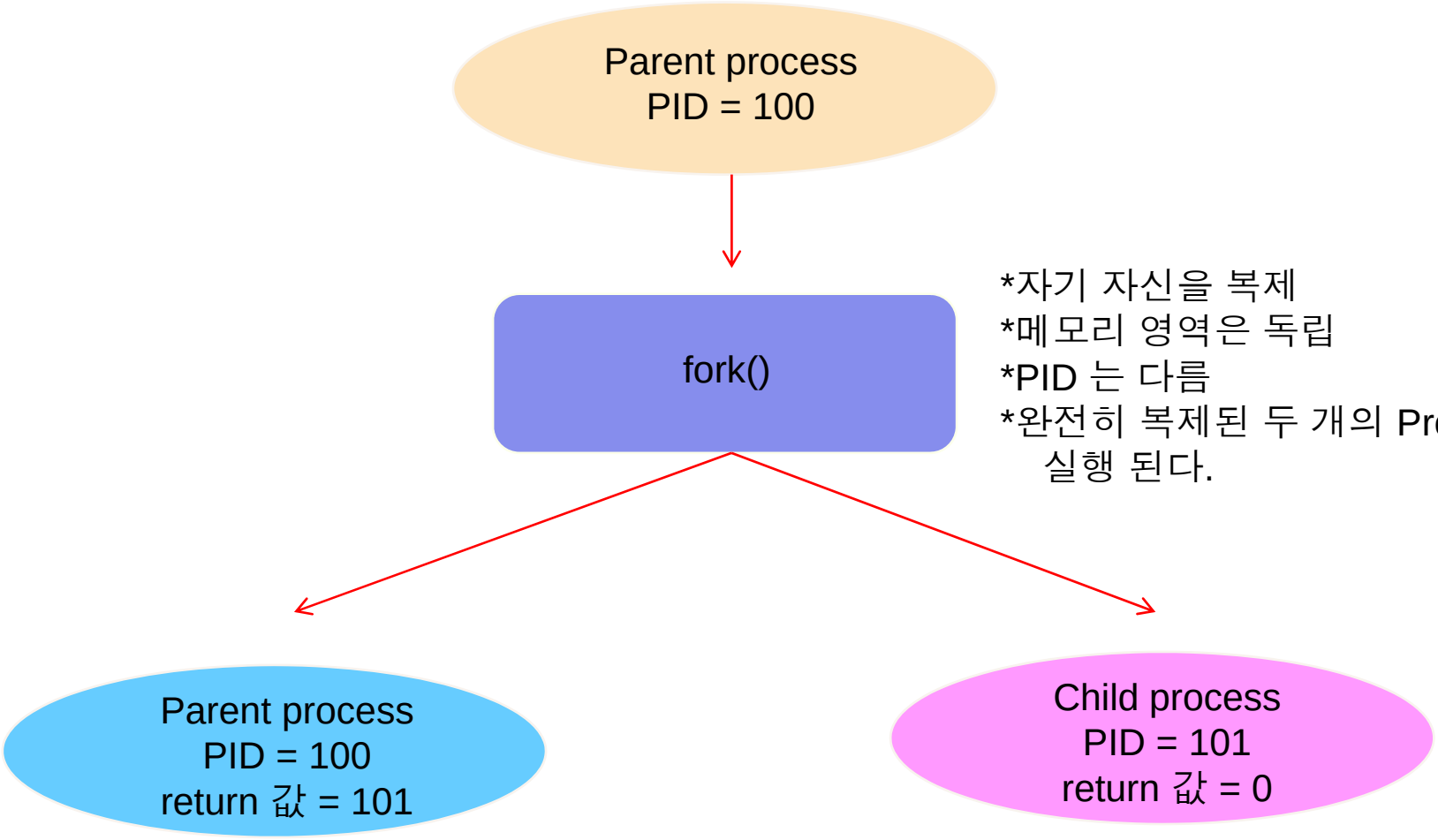


fork()

- 새로운 프로세스를 만들기 위해 주로 사용
- fork()를 호출한 프로세스의 이미지를 복사하여 새로운 프로세스를 생성
  - 원본 프로세스의 모든 메모리 공간 (데이터 영역, heap, stack)을 그대로 복사
- 부모/자식 프로세스
  - 부모 프로세스 : fork()를 호출한 프로세스
    - fork()의 리턴값 : 자식프로세스의 PID
  - 자식 프로세스 : fork()에 의해 새로 생성된 프로세스
    - fork()의 리턴값 : 0
- 프로세스의 공유
  - 부모와 자식 프로세스는 변수를 서로 공유하지 않음
  - 개설한 파일이나 소켓은 프로세스 이미지 외부에 존재하므로 공유

fork()

- 부모/자식 프로세스의 실행과 리턴값



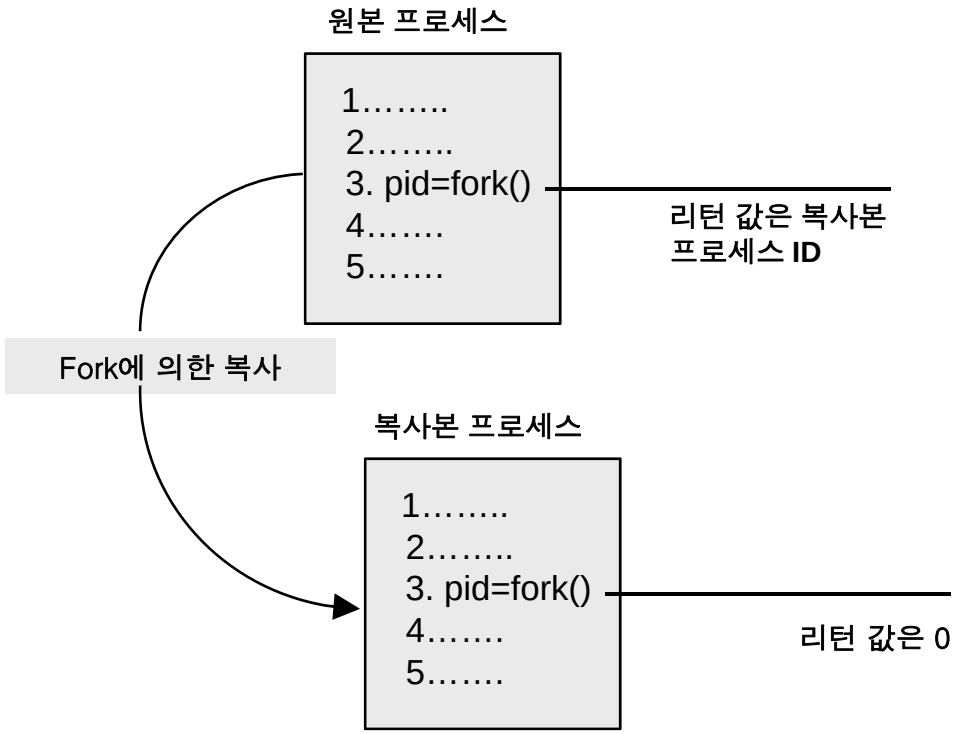
*자기 자신을 복제
*메모리 영역은 독립
*PID 는 다름
*완전히 복제된 두 개의 Process 가 실행 된다.

fork 함수 호출을 통한 프로세스의 생성

- fork 함수 호출을 통한 프로세스의 생성은 복사에 의한 생성이다.

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

pid_t  fork(void);
```

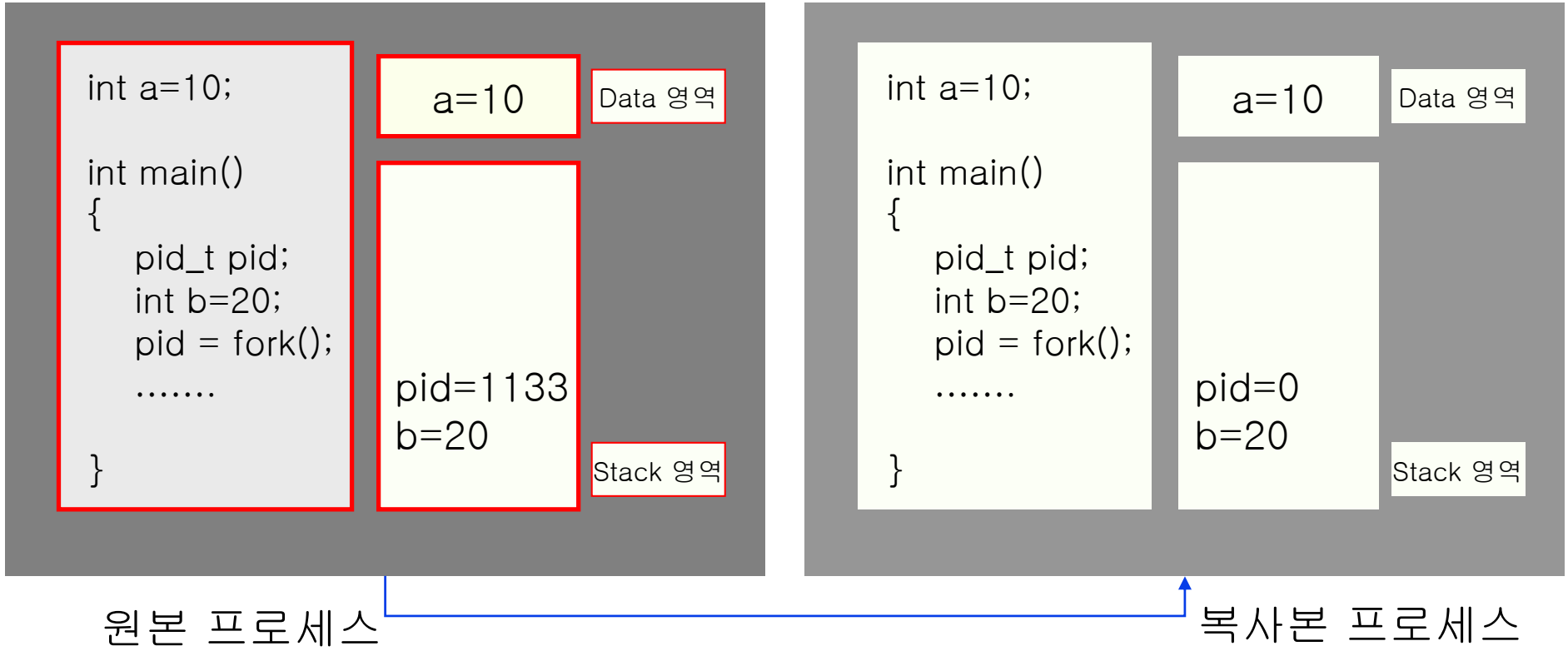



```
PID = fork();

if (PID == 0)
        child_work();        // child process에서 처리

else
        parent_work();       // parent process에서 처리
```

fork 함수 호출을 통한 프로세스의 생성



Process의 종료

- 종료 조건
  - main() 함수에서 return 되는 경우
  - exit() 함수를 호출할 경우
  - 프로세스 종료 signal을 받은 경우
- exit()
  - 프로세스가 자신을 종료시키는데 사용
  - 열려 있던 모든 파일을 닫기 위해 자동으로 close() 함수를 호출

예제 확인 #1

❑ 실행 파일

- fork.c

❑ 실행 결과

root@Network: /home/ejcho/Lecture_7_code

```
root@Network:/home/ejcho/Lecture_7_code# gcc fork.c -o fork
root@Network:/home/ejcho/Lecture_7_code# ./fork
fork 성공, 프로세스 id : 32210
data : 0
fork 성공, 프로세스 id : 0
data : 20
root@Network:/home/ejcho/Lecture_7_code#
```

예제 확인 #2

❑ 실행 파일

- fork_test.c

pid_t getpid(void)	//자신의 **PID**를 얻음
pid_t getppid(void)	//부모 프로세스의 **PID**를 얻음

```
root@Network: /home/ejcho/Lecture_7_code
root@Network:/home/ejcho/Lecture_7_code# gcc fork_test.c -o fork_test
root@Network:/home/ejcho/Lecture_7_code# ./fork_test
CHILD - my pid is 32540 and parent's pid is 32539
        global var : 1
        local var : 1
PARENT - my pid is 32539, child's pid is 32540
        global var : 5
        local var : 5
root@Network:/home/ejcho/Lecture_7_code#
```

예제 확인 #3

- 파일명
  - tcp_talkserv.c
  - tcp_talkcli.c
- 프로그램 설명
  - 토크서버에서는 먼저 listen()을 호출하고 accept()를 호출하여 클라이언트와 연결하고 연결이 이루어지면 fork()를 호출
  - 부모 프로세스는 사용자의 키보드 입력을 받아 클라이언트에게 전송
  - 자식 프로세스는 클라이언트가 보내온 메시지를 화면에 출력
  - strstr
    - 문자열 안에 특정 문자열이 존재하는지 여부를 체크하는 함수이다.
    - char * strstr(const char * string, const char * strCharSet);

      → string에서 strCharSet을 검색해 해당 문자열 위치를 포인터로 리턴한다.

예제 확인 #3

```
root@Network: /home/ejcho/Lecture_7_code
root@Network:/home/ejcho/Lecture_7_code# gcc tcp_talkserv.c -o talkserv
root@Network:/home/ejcho/Lecture_7_code# ./talkserv 6022
서버가 클라이언트를 기다리고 있습니다
클라이언트가 연결되었습니다.
hello
nice to mee you
bye
```

```
root@Network: /home/ejcho/Lecture_7_code
root@Network:/home/ejcho/Lecture_7_code# gcc tcp_talkcli.c -o talkcli
root@Network:/home/ejcho/Lecture_7_code# ./talkcli 127.0.0.1 6022
hello
nice to mee you
bye
exit
Good bye.
root@Network:/home/ejcho/Lecture_7_code#
```

실습과제 #4

- 새로운 프로세스를 생성하고 이 프로세스에서 10초 간격으로 현재의 시간을 클라이언트에게 전송 하는 프로그램 작성
  - TCP 이용
  - UDP 이용