3. 유닉스 프로세스

3.1 프로세스의 이해

프로세스의 정의

- 프로세스
  - 실행중인 프로그램
    - 프로그램이란 C, Java 등으로 작성된 소스코드이며
    - 프로세스는 이러한 프로그램이 샐행되는 것
- 프로세스의 모드
  - 사용자 모드
    - 사용자의 권한으로 명령이 실행
  - 커널 모드
    - 커널의 권한으로 실행
    - 하드디스크를 읽기 위한 read() 함수
- 프로세스의 상태
  - 실행(running) 상태
  - 블록(waiting) 상태
  - 중단(stopped) 상태
  - 좀비(zombie) 상태

프로세스의 메모리 배치

- **프로세스는 일정한 메모리를 배정 받아 사용 (이미지)**
  - **프로그램 실행에 필요한 어셈블러 코드, 변수가 저장**
  - **원칙적으로 한 프로세스는 다른 프로세스의 메모리 영역에 접근 불가**
- **C 프로그램과 이미지의 내용** (1)

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

extern char **environ;
int init_global_var = 3;
int uint_global_var;

int main(int argc, char **argv)
{
  int auto_var;                 static int static_var;
  register int reg_var;         char *aurto_ptr;
  auto_ptr = malloc(10);
  return 0
}
```

프로세스의 메모리 배치

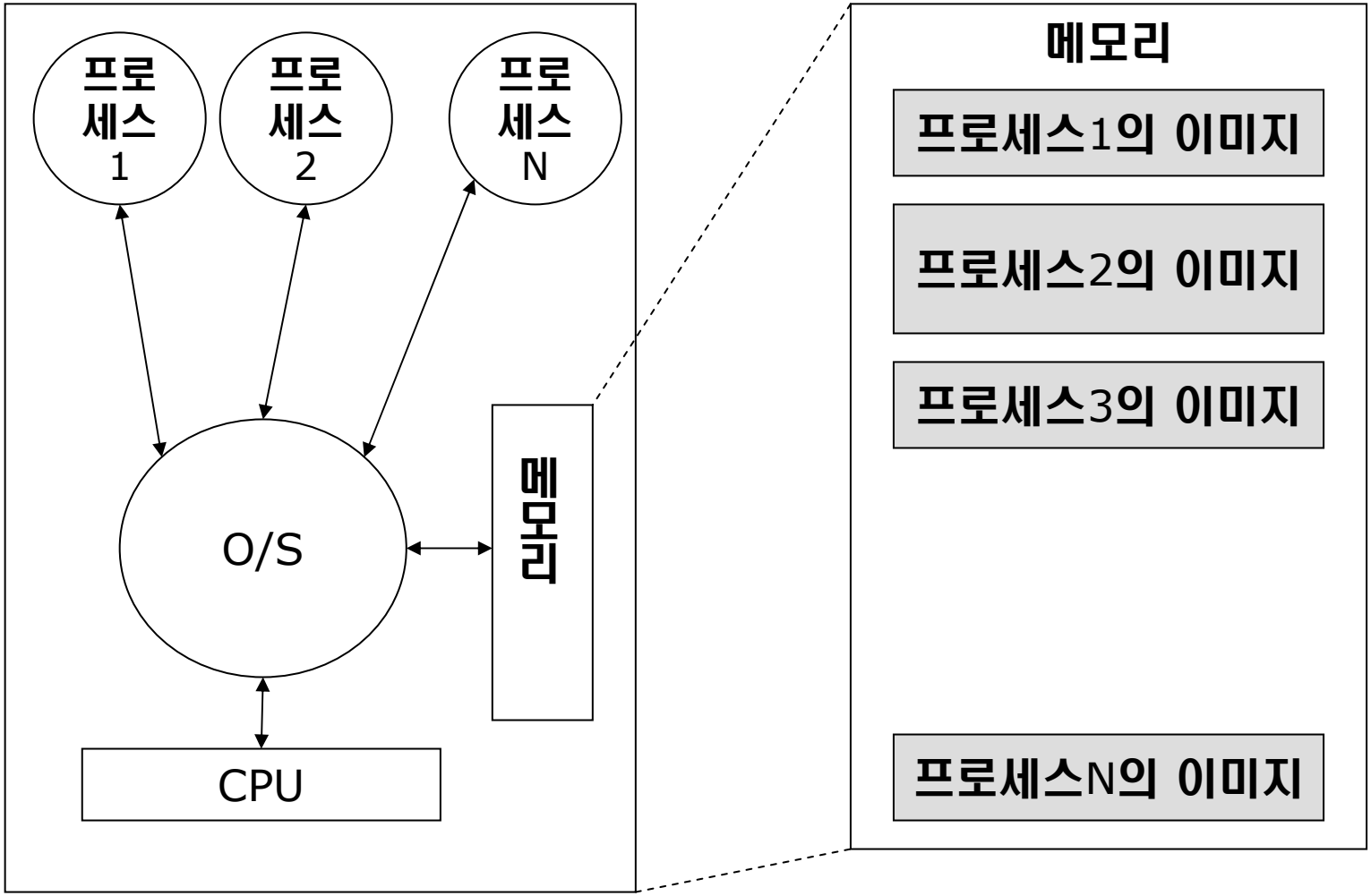
- C 프로그램과 이미지의 내용 (2)

메모리 영역		메모리 내용 및 변수
환경변수, 명령행 인자 영역		*environ 내용
스택		argc, argv, auto_var, reg_var, auto_ptr
힙		malloc()이 할당한 10바이트
데이터 영역	초기화 안된 영역	unint_global_var, static_var
	초기화된 영역	init_global_var = 3
코드 영역		어셈블된 프로그램 코드

스택과 힙

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

프로세스의 생성과 종료

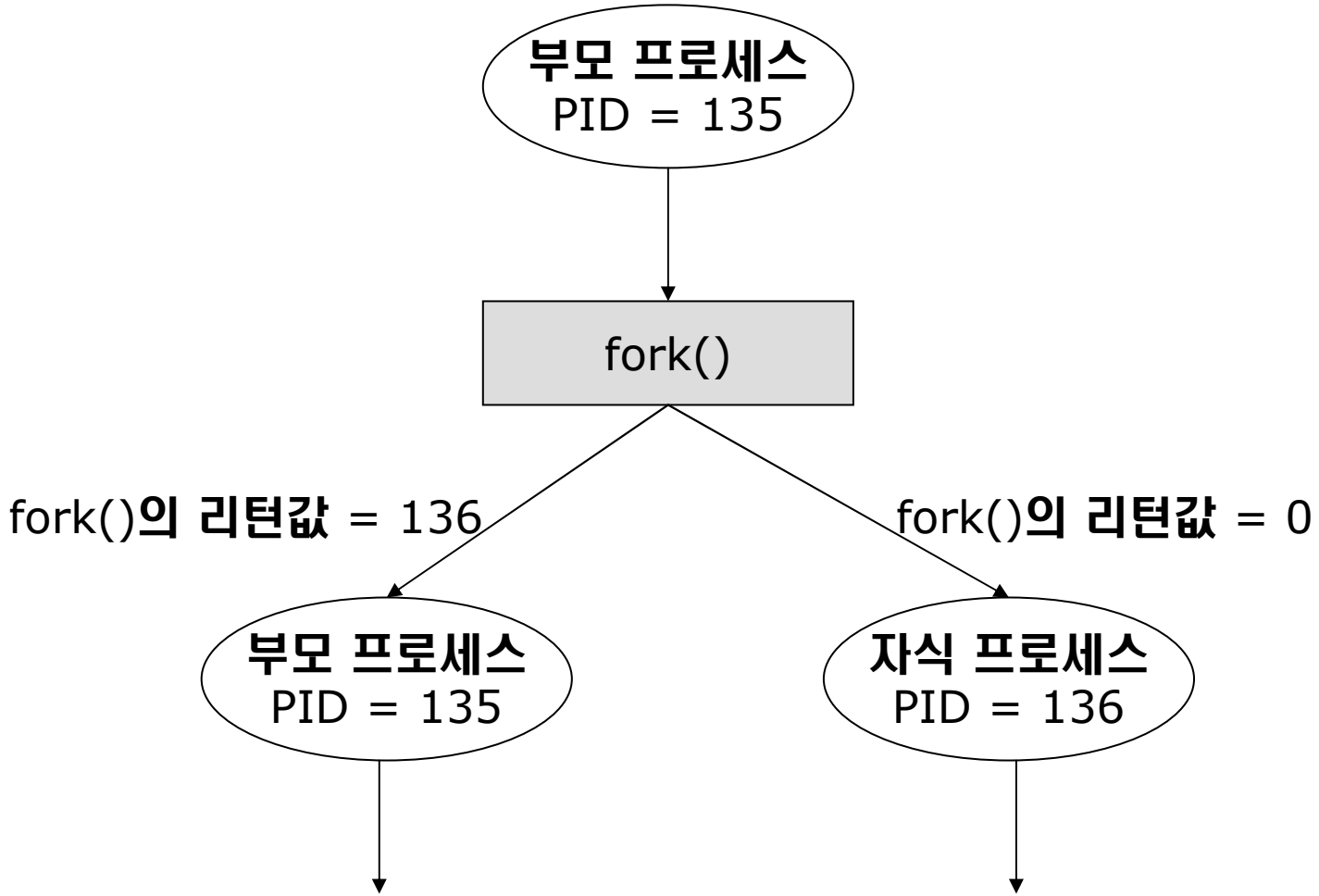

프로
세스
1
프로
세스
2
프로
세스
N
O/S
메모리
CPU
메모리
프로세스1의 이미지
프로세스2의 이미지
프로세스3의 이미지
프로세스N의 이미지

fork

- 새로운 프로세스를 만들기 위해 주료 사용
- fork()를 호출한 프로세스의 이미지를 복사하여 새로운 프로세스를 생성
- 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()

- **부로/자식 프로세스의 실행과 리턴값**



exec()

- **프로세스가 기존 작업을 종료하고 다른 작업으로 전환할 때 사용**
  - **현재 실행 중인 프로세스의 이미지를 다은 프로세스의 이미지로 교체**

```
exec("/bin/ls", "ls", "test.txt", NULL)
```

- exec **계열의 함수**

```
#include <unistd.h>

int execl(const char *path, const char *arg, …);
int execv(const char *path, char *const argv[]);
int execle(const char *path, const char *arg, …, char *const envp[]);
int execve(const char *path, char *const argv[], char * const envp[]);
int execlp(const char *file, const char *arg, …);
int execvp(const char *file, char *const argv[]);
```

l : list **형태의 인자를 취한다는 의미**
v : vector **형태의 임자를 취한다는 의미**
p : **명령을** path**에서 검색하겠다는 의미**
e : **환경변수 인자를 받는 함수임을 의미**

프로세스의 환경변수

- **파일을** open**하는 코드**
  - **현재 디렉토리에** temp **파일이 존재하지 않으면 새로운** temp **파일 생성**
  - **생성되는 디렉토리를 지정하지 않았어도 위치는 "현재 디렉토리"**
  - **이러한 현재 작업 디렉토리명과 같은 정보가 환경변수**
  - **모든 프로세스는 실행을 위한 환경변수를 가짐**
  - fork()**의 경우**
    - **부모 프로세스의 환경변수 영역이 자식 프로세스에 복사**
    - **이를 이용하여 부모 프로세스는 자식 프로세스에게 필요한 정보 전달이 가능**
- getevn() **함수**

```
char *getenv(const char *name)
```

  - **외부 전역변수인** char **environ**을 **이용하여 모든 환경변수 출력이 가능**

프로세스의 종료

- **종료 조건**
  - main() **함수에서** return **되는 경우**
  - exit() **함수를 호출할 경우**
  - **프로세스 종료** signal**을 받은 경우**
- **종료 값**
  - main()**의** return**값 또는** exit() **호출시 인자**
  - **종료시 프로세스의 종료값 이외 프로세스의 정보들이 커널에 저장됨**
  - **자식 프로세스는 부모 프로세스에게** SIGCHLD **시그널을 전송하고 종료값 및 상태정보를 전달**

프로세스의 종료

- exit()
  - 프로세스가 자신을 종료시키는데 사용
  - 열려 있던 모든 파일을 닫기 위해 자동으로 close() 함수를 호출
    - fread(), fwrite() 함수는 시스템 콜이 아니며 내부적으로 read(), write() 시스템 콜을 호출
    - fwrite()가 write()를 호출하는 것을 flush라 함
    - 프로세스 종료시 flush되지 않은 데이커가 있으며 exit() 함수는 flush를 실행
- atexit()
  - 프로세스 종료 시에 반드시 수행해야 할 작업을 지정할 때 사용
  - 프로세스 종료시 atexit() 함수로 등록된 함수를 먼저 호출한 후에 flush를 실행

```
atexit(func1);
atexit(func2);
// 종료시 func2(), func1()의 순서로 실행
```

토크 서버/클라이언트 프로그램

- **서버**
  - listen()
  - accept() & fork()
  - send(), recv()
- **클라이언트**
  - connect();
  - send(), recv()