주요 내용 요약:
1. **프로세스의 정의**: 프로세스는 실행 중인 프로그램으로, 디스크에 저장된 프로그램 코드와 데이터를 운영체제가 메모리에 로드하고 실행함으로써 생명력을 얻습니다.

2. **CPU 가상화**: 운영체제는 **시분할(time sharing)** 기법을 사용해 CPU를 여러 프로세스가 동시에 사용하는 것처럼 보이게 만듭니다. 이는 실제로 하나 또는 적은 수의 CPU를 여러 프로세스가 나눠 사용하는 방식입니다.

3. **프로세스 상태**:
   - **실행(Running)**: 프로세스가 CPU에서 실행 중인 상태.
   - **준비(Ready)**: 실행될 준비가 되었으나 CPU가 사용 중이라 대기 중인 상태.
   - **대기(Blocked)**: I/O 작업 같은 외부 작업을 기다리는 중이라 실행을 중단한 상태.

4. **프로세스 제어**: 프로세스는 생성(Create), 제거(Destroy), 대기(Wait), 일시정지(Stop) 등의 API를 통해 제어됩니다.

5. **좀비 상태**: 프로세스가 종료되었으나, 아직 부모 프로세스가 종료 상태를 수집하지 않아 프로세스의 정보가 남아있는 상태입니다. 이 상태는 종료된 프로세스가 제대로 정리되지 않았을 때 발생하며, 보통 부모 프로세스가 `wait()` 시스템 호출을 통해 이 정보를 수집하면 좀비 상태가 해제됩니다.

좀비 상태는 CPU 자원이나 메모리를 사용하지 않지만, 프로세스 테이블에 정보가 남아 있어 시스템에 많은 좀비 프로세스가 생기면 시스템 자원 관리를 방해할 수 있습니다.

```
// 프로세스 상태
enum proc_state { UNUSED , EMBRYO , SLEEPING , RUNNABLE , RUNNING ,
ZOMBIE };
// 프로세스 구조체
struct proc {
    char *mem;              // 프로세스 메모리
    uint sz;                // 프로세스 크기
    char *kstack;           // 커널 스택
    enum proc_state state;  // 프로세스 상태
    int pid;                // 프로세스 ID
    struct proc *parent;    // 부모 프로세스
    void *chan;             // 차단 상태에서 대기할 채널
    int killed;             // 프로세스가 종료되었는지 여부
    struct file *ofile[NOFILE]; // 열린 파일들
    struct inode *cwd;      // 현재 작업 디렉터리
    struct context context;  // 프로세스 문맥 저장
    struct trapframe *tf;    // 인터럽트 처리 정보
};
```

코드 설명:
enum proc_state: 프로세스의 상태를 나타내는 열거형
UNUSED: 사용되지 않는 상태
EMBRYO: 초기화 중인 상태
SLEEPING: 대기 중인 상태 (I/O 같은 작업을 기다림)
RUNNABLE: 실행 가능한 상태
RUNNING: 실행 중인 상태
ZOMBIE: 좀비 상태 (프로세스가 종료되었으나 자원이 회수되지 않음)
struct proc: 프로세스의 상태와 관련된 여러 정보를 저장하는 구조체입니다.
mem: 프로세스가 사용하는 메모리의 포인터
sz: 프로세스의 메모리 크기
kstack: 커널 스택 포인터
state: 프로세스의 현재 상태 (proc_state 타입)
pid: 프로세스
IDparent: 부모 프로세스에 대한 포인터
chan: 프로세스가 대기할 채널 (차단 상태일 때 사용)
killed: 프로세스가 강제 종료되었는지 여부
ofile: 프로세스가 열고 있는 파일들
cwd: 프로세스의 현재 작업 디렉토리
context: 프로세스 문맥 교환에 필요한 레지스터 상태
tf: 인터럽트 처리 정보를 저장하는 구조체

예상되는 결과:이 구조체는 운영체제의 커널이 각 프로세스의 상태를 추적하고 관리하는 데 사용됩니다. 특히 프로세스의 **상태(state)**가 변화하는 것을 관리하여, 프로세스가 실행 중인지, 대기 중인지, 종료되었는지를 추적합니다. 예를 들어, 프로세스가 종료된 후에도 좀비 상태로 남아 있다가 부모 프로세스가 종료 상태를 수집하면 그제서야 완전히 제거됩니다. 이 코드 자체는 프로세스의 상태를 정의한 것이므로, 실행 결과라기보다는 프로세스의 상태 전이를 위한 기반이 되는 구조체라고 볼 수 있습니다.