프로세스의 생성, 실행과 EXEC 시스템 콜

프로세스의 생성, 실행과 EXEC 시스템 콜

1.1) 프로세스 생성

커널의 사용자 레벨 프로세스 생성 과정

커널

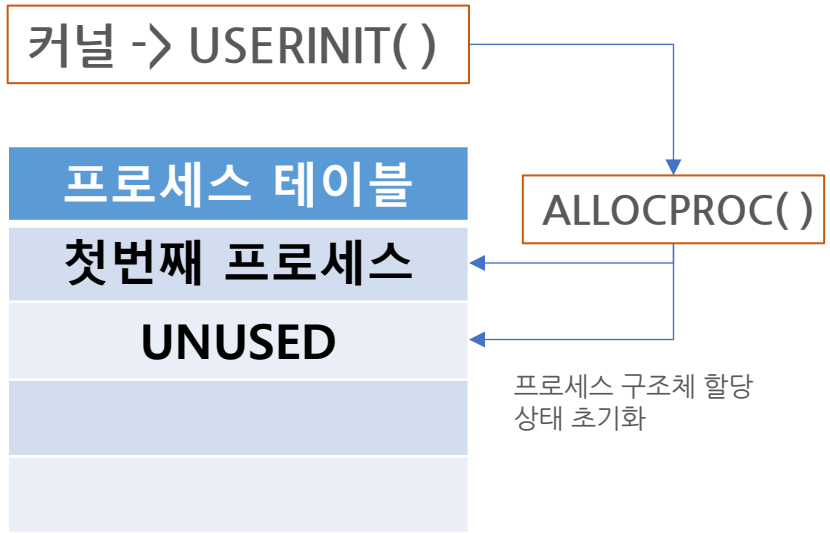
- ▶ USERINIT **함수**로 첫 프로세스 호출 (MAIN.C -> PROC.C)

USERINIT

- ▶ ALLOCPROC **함수**로 각각의 프로세스 호출 (PROC.C)

ALLOCPROC

- ▶ 프로세스 테이블의 **미사용 공간** 탐색
- ▶ **초기(EMBRYO) 상태**로 설정
- ▶ 프로세스 ID를 부여하고 커널 스택 할당



프로세스의 생성, 실행과 EXEC 시스템 콜

1.1) 프로세스 생성

커널의 사용자 레벨 프로세스 생성 과정

커널 스택 할당, 배치

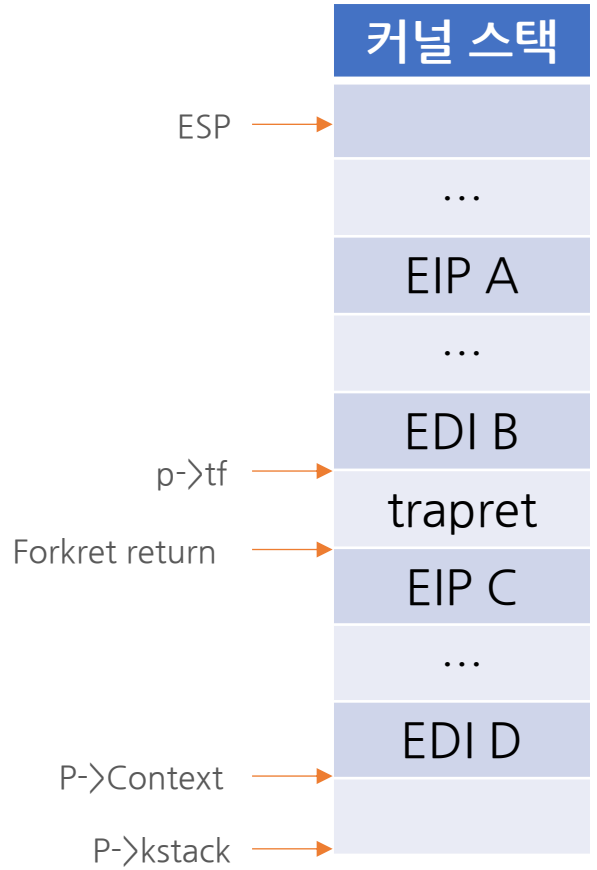
- 프로세스가 동작할 때 실행되는 함수, 레지스터 등을 미리 탑재한다.
- 커널 레지스터는 사용자 공간으로 복귀하기 위해 사용된다.

P->tf

- 프로세스 구조체의 트랩 프레임
- 사용자 프로세스의 레지스터 저장

P-> context

- 프로세스 구조체의 현재 프로세스 문맥
- 문맥 교환(context switch)를 위한 저장 공간



프로세스의 생성, 실행과 EXEC 시스템 콜

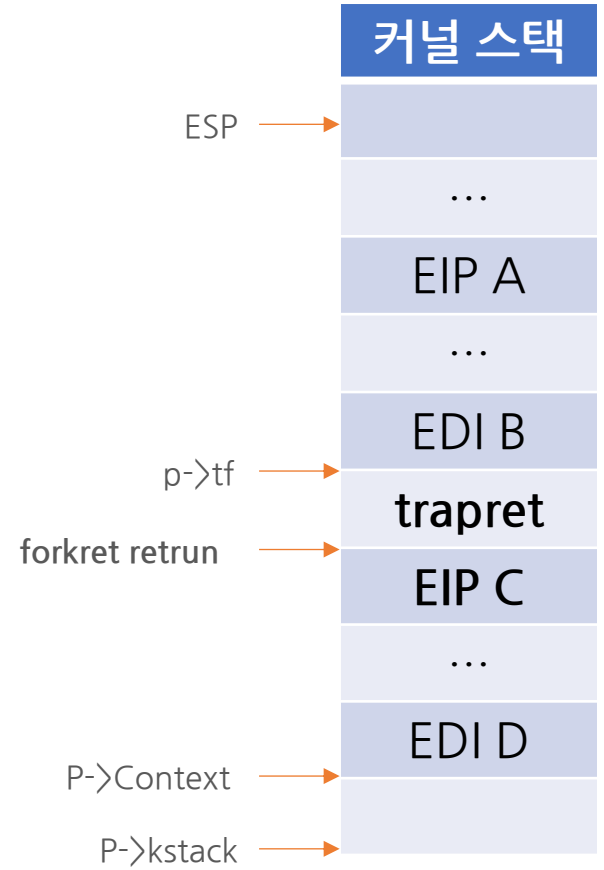
1.1) 프로세스 생성

커널의 사용자 레벨 프로세스 생성 과정

EIP
다음 실행될 명령어의 주소 저장

forkret( )
문맥의 EIP가 forkret의 위치로 설정되고, fork가 종료되면 forkret 실행

trapret( )
커널 모드에서 사용자 모드로 이동하기 위하여 커널 스택의 메모리 복구
RETURN FROM TRAP



프로세스의 생성, 실행과 EXEC 시스템 콜

1.1) 프로세스 생성

커널의 사용자 레벨 프로세스 생성 과정

USERINIT( )

▶ SETUPKVM( ) 호출

SETUPKVM( )

▶ 최초 프로세스와 커널 메모리를 연결할 페이지 테이블 생성

INITUVM( )

▶ 프로그램과 메모리 페이지 테이블을 연결

프로세스의 생성, 실행과 EXEC 시스템 콜

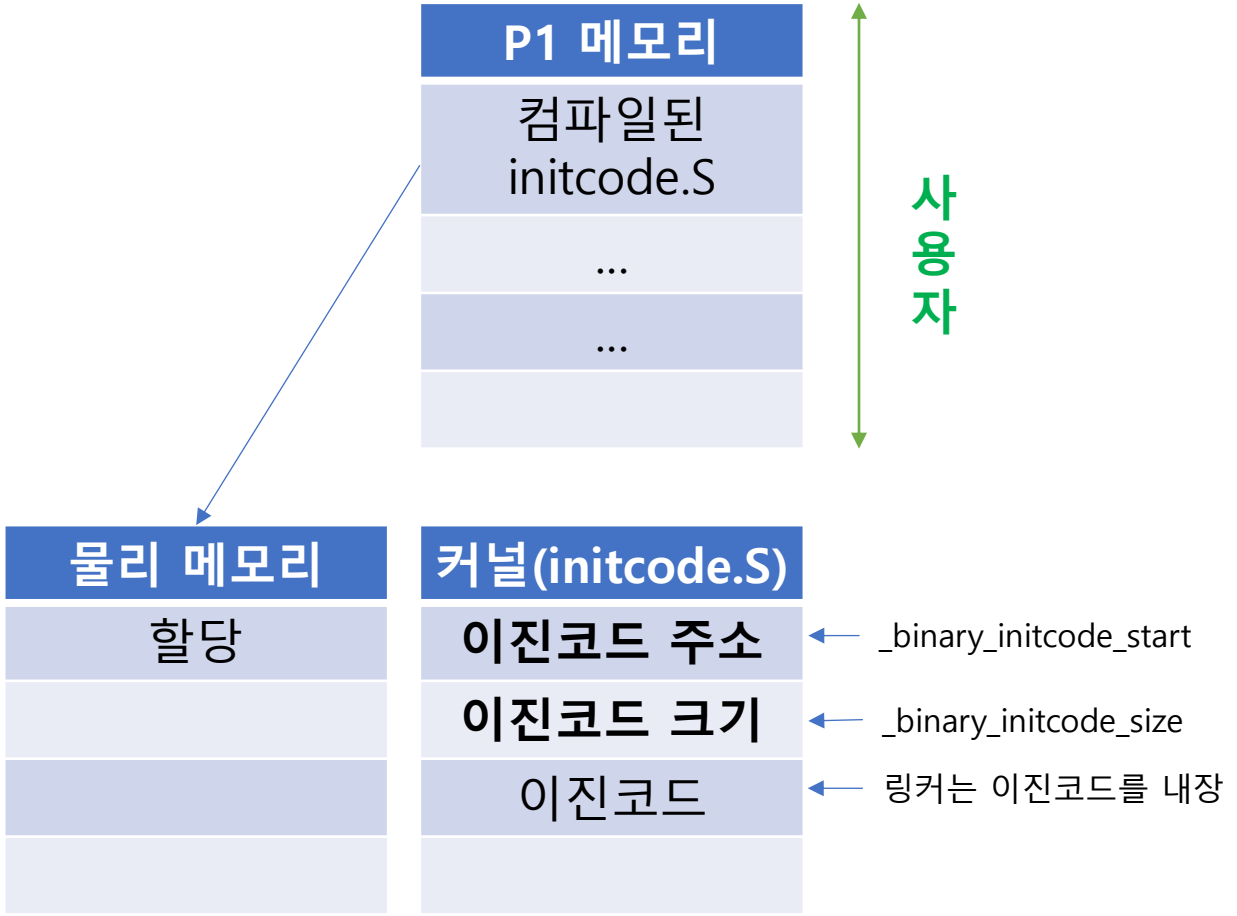
1.1) 프로세스 생성

커널의 사용자 레벨 프로세스 생성 과정

INITUVM( )
커널 내부의 이진코드를 프로세스 메모리로 복사한다.

이후, 물리 메모리를 할당하고 가상 메모리 주소와 연결한다.

가상 주소 내부의 이진코드 값을 물리 메모리에 복사한다.



프로세스의 생성, 실행과 EXEC 시스템 콜

1.1) 프로세스 생성

커널의 사용자 레벨 프로세스 생성 과정

USERINIT( )

trapframe을 사용자 모드 상태의 초기값으로 설정
프로세스 상태를 실행 가능(Runnable) 상태로 설정하고 스케줄링 준비

ESP는 프로세스 구조체의 SZ로 설정
EIP는 초기화된 코드의 시작점으로 설정

DPL(Decriptor Privilege Level)

- ▶DPL_USER는 0x3으로 정의 (mmu.h)
- ▶ cs, ds, es, ss 레지스터는 사용자 코드와 사용자 데이터, 스택을 DPL 사용자 권한으로 사용

▶ 커널은 최초 프로세스를 생성하고 실행하기위해 스택을 생성하고 메모리를 할당하여 실행을 준비한다.

프로세스의 생성, 실행과 EXEC 시스템 콜

1.2) 프로세스 실행

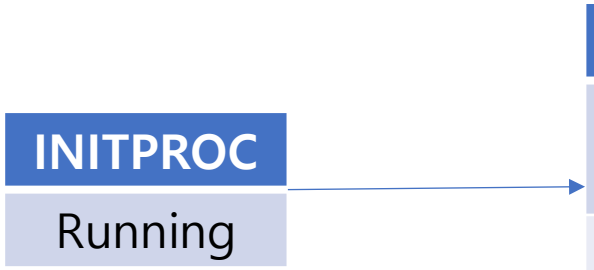
첫 사용자 프로세스의 실행 과정

mpmain( )

USERINIT 이후, 스케줄러를 호출하여 Runnable 상태의 프로세스를 탐색 하게한다.

스케줄러

Runnable 상태의 최초 생성 프로세스를 스케줄링 한다. (INITPROC)
프로세스의 상태를 Running으로 변경하고 **문맥 교환**을 수행한다.
스케줄러는 반환되지 않고 **지속적으로** 스케줄링을 수행한다.



커널 모드	하드웨어	사용자 모드
프로세스 스케줄링 레지스터 저장		
	ESP, EIP 복원	
		프로세스 실행

프로세스의 생성, 실행과 EXEC 시스템 콜

1.2) 프로세스 실행

첫 사용자 프로세스의 실행 과정

ALLOCPROC( )

프로세스 실행 전 문맥의 EIP를 forkret으로 설정했다

.

forkret이 실행되면 초기화를 진행한다.

이는 커널이 아닌 프로세스 자신의 문맥에서만 실행된다.

이후, 스택의 ESP(최상위)를 프로세스 구조체의 TF로 설정한다.

트랩 프레임

trapret을 통해 프로세스의 트랩프레임 내부에 접근하여 명령을 수행한다.

- ▶POPl - gs, fs, es, ds 레지스터를 복구
- ▶addl - trapno, errcode 영역을 건너 뛴다.
- ▶iret - cs, eip, flags, esp, ss 레지스터를 스택에서 빼낸다.

```
sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;

  sp -= 4;
  *(uint*)sp = (uint)trapret;

  sp -= sizeof *p->context;
  p->context = (struct context*)sp;
  memset(p->context, 0, sizeof *p->context);
  p->context->eip = (uint)forkret;
```

proc.c 내부 allocproc 함수 일부

```
popl %gs
popl %fs
popl %es
popl %ds
...
addl $0x8, %esp # trapno and errcode
```

kernel.asm 일부

프로세스의 생성, 실행과 EXEC 시스템 콜

1.2) 프로세스 실행

첫 사용자 프로세스의 실행 과정

EIP와 ESP

프로세스를 생성하면서 할당한 가상 주소로 고정된다.
페이징 하드웨어가 가상 주소를 물리 주소로 연결한다.

allocuvm( )

가상 주소의 시작점을 물리 주소로 할당한다.
사용자 권한이 메모리에게 접근하기 위한 플래그(PTE_U)를 설정한다.

이때, userinit( )은 사용자 권한을 **현재 실행 권한(CPL)**을 3으로 설정하여
PTE가 사용자 권한을 벗어날 수 없게 한다.

▶ 프로세스는 DPL, CPL, PTE 플래그가 사용자로 설정되어 프로세스 자신의 메모리만 사용할 수 있다.

프로세스의 생성, 실행과 EXEC 시스템 콜

1.3) EXEC 시스템 콜

사용자 레벨의 프로세스가 커널 권한 실행을 요청하는 방법

initcode.S

EXEC 시스템 콜을 호출한다.
ARGV, **init**, 0(인자의 끝)을 스택에 넣고 eax 레지스터를 SYS_EXEC로 설정한다.

init은 exec를 통해 실행할 프로세스를 의미한다.
argv는 프로세스(init 인자)로 전달할 인자를 의미한다.

이후, T_SYSCALL을 실행하여 EXEC를 실행한다.

EXEC 결과

- ▶ 성공 시, 콘솔 쉘을 실행하고 종료될 때까지 고아 프로세스를 처리한다.
- ▶ 종료 시, 프로세스는 반환되지 않는다.
- ▶ 실패 시, EXIT을 호출한다.

```
#exec(init, argv)
.globl start
start:
  pushl $argv
  pushl $init
  pushl $0
  movl $SYS_exec, %eax
  int $T_SYSCALL
```

initcode.S 내부

참고 자료

[1] Russ Cox, Frans Kaashoek, Robert Morris, “xv6, a simple, Unix-like teaching operating system” , https://pdos.csail.mit.edu/6.828/2017/xv6/book-rev10.pdf , Chapter 1, P23~27