Pintos Project #1 - Pintos 환경 구축

4/17/2019

1 개요

- ✓ 이번 과제는 Pintos의 소스 코드를 분석, 전체적인 구조를 이해하는 것을 목표로 한다.

2 내용

- ✓ Pintos 운영체제가 부팅을 시작하여, alarm-multiple 테스트를 수행하기까지의 과정에 대해 (1) 함수 분석, (2) 자료구조 분석, (3) 프로그램 실행 경로 분석 등을 수행한다.
- ✓ 배포한 가상 머신 상의 ~/pintos/src/threads 디렉토리 내의 소스 코드를 대상으로 분석한다.
- ✓ 대상 소스 코드 중 조건부 컴파일을 위한 매크로 USERPROG와 FILESYS 해당 부분은 분석 대상에서 제외한다.

3 제출물

- ✓ "pintos –v -- run alarm-multiple" 명령을 실행시켰을 경우의 프로그램 수행에 대한 소스 코드를 분석하여 제출한다.
- ✓ 보고서 양식/분량 등은 자유롭게 하되 (별첨 보고서 예시 참조), 프로그램 수행 경로를 따라 주요 함수를 분석하고, 관련 주요 자료구조에 대한 분석을 포함하여야 한다.

4 기타

- ✓ 과제 제출 기한은 5/4(토)이다.
- ✓ 개인별 과제로 진행한다.
- ✓ 이번 소스코드분석 과제의 배점은 전체 Pintos 과제 총점 100% 중 20%에 해당한다.
- ✓ TAGS (또는 ETAGS) 파일을 만들고, vi (또는 emacs)를 활용하거나, 배포한 html 형태로 변환한 소스코드를 활용한다.
- ✓ E-campus의 관련 자료 참고,

5 참고문헌

- ✓ Ben Pfaff, Pintos
- ✓ Intel Corp., Intel® 64 and IA-32 Architectures, Volume 1: Basic Architecture
- ✓ Intel Corp., Intel® 64 and IA-32 Architectures, Volume 3A: System Programming Guide, Part 1
- ✓ phillip@ussrback.com, Using Assembly Language in Linux

<자료구조 예시>

7 Pintos source code 분석

표 5. [struct] thread

```
struct thread
  {
    tid_t tid;                          /* Thread identifier. */
    enum thread_status status;          /* Thread state. */
    char name[16];                      /* Name (for debugging purposes). */
    uint8_t *stack;                     /* Saved stack pointer. */
    int priority;                       /* Priority. */

    struct list_elem elem;              /* List element. */

#ifdef USERPROG
    uint32_t *pagedir;                  /* Page directory. */
#endif

    unsigned magic;                     /* Detects stack overflow. */
  };
```

.tid

thread 를 구분하는 고유의 id 를 저장하는 영역이다.

.status

thread 의 현재 상태 정보를 저장하는 영역이다. 수행 중인 thread 는 RUNNING, 수행할 준비가 되어있으나 CPU 를 기다리는 thread 는 READY, 이벤트나 장치를 기다리기 위해 대기하는 thread 는 BLOCKED, 수행을 마치고 종료되는 thread 는 DYING 상태를 갖는다.

.name

thread 의 이름을 저장하는 영역이다. 주로 디버깅 메시지를 확인할 때 사용된다.

.stack

thread 가 사용하는 stack 의 sp 를 저장하는 영역이다.

.priority

thread 의 우선순위를 저장하는 영역이다. PRI_MIN 은 0, PRI_DEFALUT 는 31, PRI_MAX 는 63 을 의미하며, 숫자가 높을수록 높은 우선순위를 가진다.

.elem

thread 구조체 사이에 리스트 구성을 위해 이전 thread 구조체와 다음 thread 구조체의 pointer 를 저장하는 영역이다.

.magic

magic number 는 thread 정보와 stack 영역 사이에 위치한다. stack overflow 가 발생할 경우, magic number 가 변경되기 때문에 에러를 검출하는데 사용된다.

thread 구조체의 모습은 다음과 같다.

<그림>

<함수 설명 예시>

표 9. [function] palloc_init()

```
void
palloc_init (void)
{
  /* End of the kernel as recorded by the linker.
     See kernel.lds.S. */
  extern char _end;

  /* Free memory. */
  uint8_t *free_start = pg_round_up (&_end);                              (1)
  uint8_t *free_end = ptov (ram_pages * PGSIZE);                          (2)
  size_t free_pages = (free_end - free_start) / PGSIZE;                   (3)
  size_t user_pages = free_pages / 2;                                     (4)
  size_t kernel_pages;
  if (user_pages > user_page_limit)                                       (5)
    user_pages = user_page_limit;
  kernel_pages = free_pages - user_pages;                                 (6)

  /* Give half of memory to kernel, half to user. */
  init_pool (&kernel_pool, free_start, kernel_pages, "kernel pool");      (7)
  init_pool (&user_pool, free_start + kernel_pages * PGSIZE,              (8)
             user_pages, "user pool");
}
```

(1) **Pintos** 의 linkscript 인 *kernel.lds.S* 에 정의된 끝 주소(_end)를 참조하여 **Pintos** 가 사용할 메모리 영역의 시작 주소를 변수 free_start 에 저장한다.

(2) page 개수와 page 크기(PGSIZE)를 곱해서 전체 영역의 크기를 구하고 그 값을 변수 free_end 에 저장한다.

(3) 사용 가능한 전체 메모리 영역의 페이지 개수를 구하여 변수 free_pages 에 저장한다.

(4) 전체 페이지 개수 중 절반을 user_pages 에 할당한다.

(5) 만약 유저 페이지 수(user_pages)가 유저 페이지의 최대 크기(user_page_limit)보다 크면, 유저 페이지 수를 유저 페이지의 최대 크기와 동일하게 설정한다.

(6) user_page 에 할당하고 남은 나머지 페이지 개수를 kernel_pages 에 할당한다.

(7) “kernel pool”이라는 이름의 base address 를 free_start 로 하고, page 개수가 kernel_pages 개인 memory pool 을 생성한다.

(8) “user pool”이라는 이름의 base address 를 (free_start + kernel pool 의 크기)로 하고, page 개수가 user_pages 개인 memory pool 을 생성한다.

palloc_init 함수가 동작한 후의 memory pool 의 구조는 다음과 같다.

<그림>