Introduction to Operating Systems Project #1: Getting Acquainted with Background Knowledge

04/07/2022

Prof. Seongsoo Hong
sshong@redwood.snu.ac.kr
SNU RTOSLab
Dept. of Electrical and Computer Engineering
Seoul National University

Agenda

I. **Goal of Project #1**
II. Project Description
III. Background
IV. Project Submission

과제 목적과 내용 (1)

- 과제 목적
  - 하드웨어 종속성이 많은 “시스템 초기화”와 “인터럽트 처리”를 등 운영체제가 어떻게 처리하는지 이해한다
  - “i386 아키텍처”, “C 함수의 호출 규약(calling convention)”에 대해서 이해한다
    - Project #2 수행을 위해 필요

과제 목적과 내용 (2)

- 과제 내용
  - eOS 소스 코드 분석
    - 초기화 및 인터럽트 처리 루틴
    - 인터럽트 관리 모듈
  - C 서브루틴 코드 분석

- 제출물
  - 명시된 내용에 대한 보고서

Agenda

I. Goal of Project #1
II. **Project Description**
III. Background
IV. Project Submission

1. eOS 소스 코드 분석 (1)

- 시스템 초기화와 인터럽트 처리 루틴
  - 시스템 초기화, 인터럽트 요청 등의 하드웨어 신호가 발생하면 CPU는 RAM에 저장된 “인터럽트 벡터 테이블”을 참조하여 적절한 루틴으로 jump한다
    - eOS의 인터럽트 벡터 테이블은 에뮬레이션 모듈에 정의됨
  - 분석할 내용
    - 시스템 초기화 루틴
      - `main` 함수에서 인터럽트 벡터 테이블의 reset entry인 `_vector[0]`로 jump하여 시스템 초기화를 완료할 때까지의 코드
    - 인터럽트 처리 루틴
      - `_gen_irq` 함수에서 인터럽트 벡터 테이블의 irq entry인 `_vector[3]`로 jump하여 인터럽트 처리를 완료할 때까지의 코드

1. eOS 소스 코드 분석 (2)

- 인터럽트 관리 모듈
  - 인터럽트를 enable/disable하거나
    특정 irq를 mask/unmask 하는 등의 기능을 제공
  - 분석할 내용
    - 인터럽트 관리 모듈(hal/linux/interrupt.c,
      hal/linux/interrupt_asm.S)에 구현된 API들을 분석
      - hal/linux/interrupt.c
        - » `eos_ack_irq(), eos_get_irq(), eos_disable_irq_line(), eos_enable_irq_line()`
      - hal/linux/interrupt_asm.S
        - » `eos_disable_interrupt(), eos_enable_interrupt(), eos_restore_interrupt()`

2. C 서브루틴 링키지 코드 분석 (1)

- C 서브루틴 링키지 코드 분석
  - C 컴파일러는 함수가 호출될 때마다 정해진 규약에 따라 caller와 callee의 코드 사이에 특정 목적을 가진 코드를 생성
  - 분석할 내용
    - 예제 코드의 어셈블리 코드를 분석하여 함수가 호출되는 때부터 다시 리턴할 때 까지 일어나는 일련의 과정과 해당 시점의 스택의 모습들을 line-by-line으로 정리

2. C 서브루틴 링키지 코드 분석 (2)

- 예제 코드

```
int add(int a, int b) {
    int x;
    x = a + b;
    return x;
}

int mul(int a, int b) {
    return a * b;
}

int main(int argc, char **argv) {
    int a, b, c;
    int * ret;
    a = 10;
    b = 10;
    c = 5;
    *ret = mul(a, add(b, c));
}
```

2. C 서브루틴 링키지 코드 분석 (3)

- 어셈블리 코드 확인 방법
  - gcc
    - 다음 명령을 수행하여 C 소스 파일을 컴파일하면 어셈블리 코드 파일을 얻어 확인할 수 있다
    - `gcc -S <filename.c>`

Agenda

I. Goal of Project #1
II. Project Description
III. **Background**
IV. Project Submission

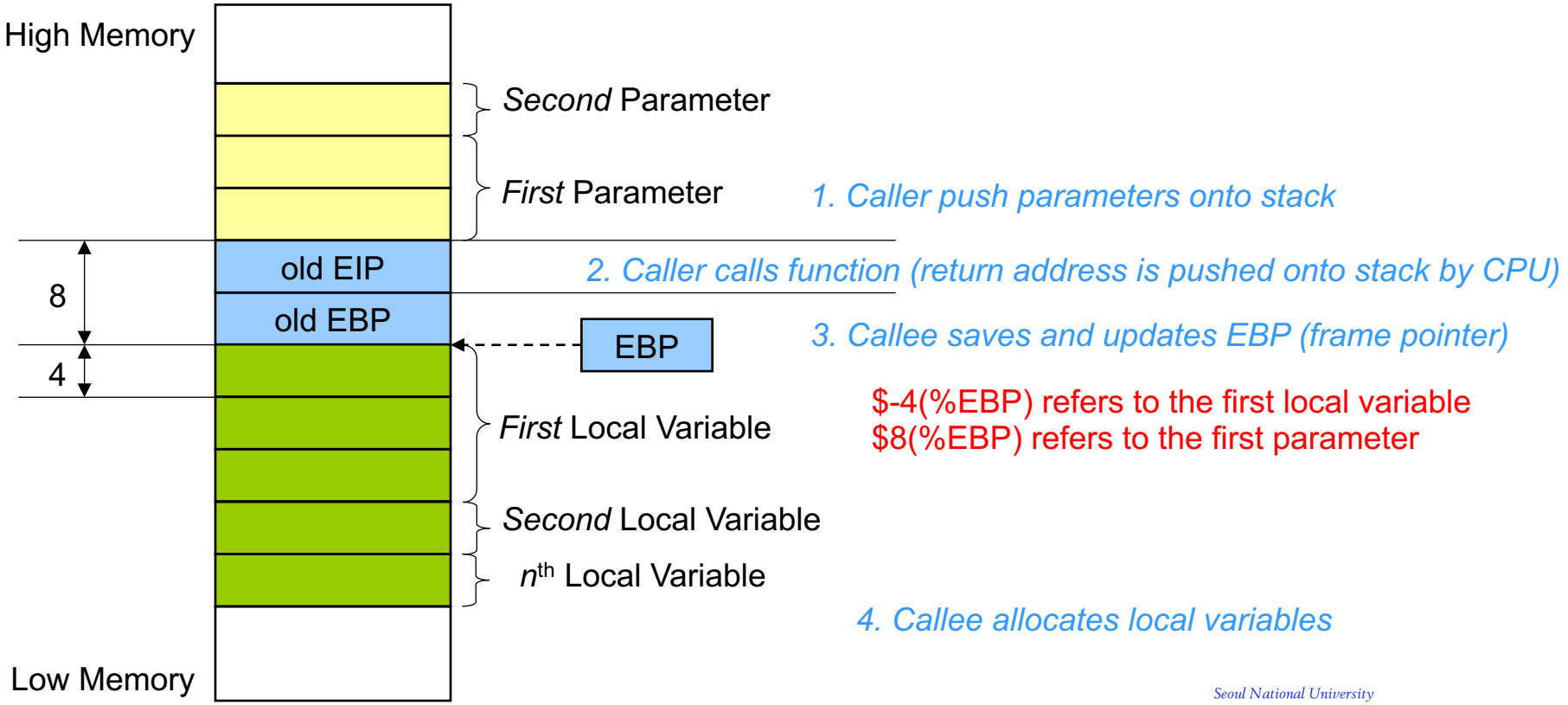
1. C 서브루틴 링키지 (1)

- Prologue code: 함수 호출 시 삽입되는 코드
  1. 전달될 인자들을 저장할 공간을 스택에 마련
  2. 할당된 스택 공간에 인자들의 값을 저장
  3. 리턴 주소를 스택에 저장
  4. 함수를 호출
  5. 호출된 함수의 지역 변수를 위한 공간을 스택에 할당

- Epilogue code: 함수 리턴 시 삽입되는 코드
  1. 자신이 사용한 지역 변수를 위한 공간을 반환
  2. 리턴 주소를 스택에서 얻어 옴

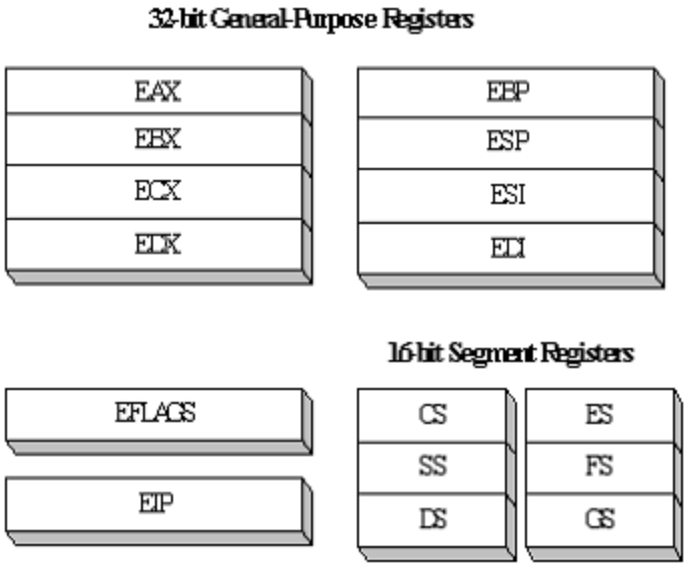
1. C 서브루틴 링키지 (2)

- Stack configuration under i386 calling convention



2. i386 ISA (1)

- Registers
  - 범용 레지스터 (32-bit)
    - EAX, EDX, ECX, EBX, EBP, ESI, EDI, ESP
  - 세그먼트 레지스터 (16-bit)
    - CS, SS, DS, ES, FS, GS
  - 플래그 레지스터
    - EFLAGS
  - 인스트럭션 포인터
    - EIP



2. i386 ISA (2)

- Instructions

mov %eax, %ebx	eax 레지스터의 값을 ebx 레지스터로 복사
mov $100, %eax	100을 eax 레지스터로 복사
mov 100, %eax	메모리 100번지의 값을 eax 레지스터로 복사
mov (%eax), %ebx	eax가 가리키는 메모리 주소의 값을 ebx 레지스터로 복사
mov 10(%eax), %ebx	eax 레지스터가 가리키는 메모리 주소 + 10의 값을 ebx 레지스터로 복사
push %eax	eax 레지스터의 값을 스택에 push. esp 레지스터의 값이 4만큼 감소
pop %eax	스택으로부터 한 개의 데이터를 eax 레지스터로 pop. esp 레지스터 값이 4만큼 증가
call FUNC	다음 인스트럭션의 주소를 스택에 push하고 FUNC로 점프
ret	스택에 저장되어 있는 복귀 주소를 eip로 pop

Agenda

I. Goal of Project #1
II. Project Description
III. Background
IV. **Project Submission**

과제 제출

- 제출물
  - 명시된 내용에 대한 보고서
- 제출 기한
  - 4/28(목) PM 11:59 까지
- 제출 방법
  - ETL로 제출

Question or Comment?