운영체제 Lab 3 보고서

# Goal 1

### 분석

Goal 1의 목표는 uthread\_switch.S를 완성하는 것이다.  
uthread\_swithch.S는 uthread.c에서 current\_thread에서 next\_thread로 쓰레드 스위칭을 하기 위한 함수이다.

우선 thread의 구조를 분석했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

쓰레드는 sp, stack, state를 갖는데 STACK\_SIZE가 8192이기 때문에 총 thread의 크기는 4+8192+4 = 8200byte의 크기를 갖는 것을 알 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

thread\_create 함수를 보면 모든 쓰레드를 초기화해주는 것을 볼 수 있다.  
초기화 과정은 sp의 위치를 t->stack + STACK\_SIZE로 설정해주는데 이는 스택이 비어있을 때 sp의 위치로 초기화 해준 것이다.  
sp의 값을 4를 빼준 뒤 매개변수로 받아온 함수 포인터를 넣어주는 것은 쓰레드의 명령어 시작점을 넣어주는 것이고 이후 32를 더 빼주는 것은 8개의 레지스터 eax, ebx, ecx, edx, ebp, esp, esi, edi를 넣어주기 위한 공간으로 보인다.

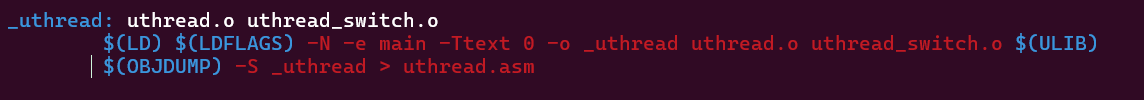
쓰레드 스위치를 하려면 우선 8개의 레지스터를 저장해야 하는 것을 알 수 있다.

이때 esp 레지스터의 경우 복원을 할 경우 마지막 ret 명령어에서 esp의 위치의 값을 eip에 넣은 뒤 해당 주소로 jmp하기 때문에 esp를 제외한 7개의 레지스터만 저장해야 했다.  
그래서 esp의 값을 4만큼 이동한 뒤 7개의 레지스터의 값을 push하고 레지스터를 복원할 때도 7개의 레지스터의 값을 pop한 뒤 4만큼 esp를 이동하면 된다.

push와 pop 사이에서는 현재 스택의 esp를 서로 교환해준 뒤 current\_thread도 바꿔주면 된다.

### 수행

우선 Makefile에 uthread\_switch.S의 정보와 \_uthread 명령어를 등록했다.



텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

uthread\_switch.S  
32바이트를 맞추기 위해 4를 빼 주고 7개의 레지스터를 push했다.  
이후 스택을 교환해주고 current\_thread도 바꿔준 뒤 push의 역순으로 레지스터를 복원해주었다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

thread 출력 시 개수를 확인하기 위해 카운트 값도 같이 출력했다.

두 쓰레드가 번갈아 가며 문제없이 100번씩 출력하고 종료하는 것을 확인할 수 있다.

# Goal 2

### 분석

Goal 2의 목표는 Timer interrupt가 발생했을 때 thread\_schedule 함수를 호출해야 하는 것이다.

주어진 힌트는 uthread\_init 시스템 콜을 만들고 PCB에 스케줄러의 주소를 저장하는 부분을 추가한 뒤 uthread\_init을 통해 thread\_schedule 함수의 주소를 PCB에 저장하는 것이다.

우선 Timer interrupt가 발생했을 때 어떤 일어나는지 분석했다.

Timer interrupt가 발생하면 alltraps에서 현재 레지스터들 값을 push하여 trap frame을 만든다.

x86.h의 trapframe을 보면 push한 순서대로 trapframe 구조체가 구성되는 것을 확인할 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명.텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이후 trap을 call하면 trapno의 값을 통해 시스템 콜인지 일반 인터럽튼지 구분한다.

만약 시스템 콜이라면 현재 프로세스의 트랩 프레임을 해당 트랩 프레임으로 지정한 뒤 시스템 콜을 처리하기 위해 떠난다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

만약 시스템 콜이 아니면 그 밑에 있는 switch 문에서 어떤 종류의 트랩인지 확인하게 된다.

제일 먼저 걸리는 것이 바로 타이머 인터럽트일 경우이다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

위의 처리가 끝나고 switch문을 빠져나가면 타이머 인터럽트가 발생했을 경우에는 해당 if문을 통해 yield를 호출하여 컨텍스트 스위칭을 하러 간다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다시 컨텍스트 스위칭을 하여 해당 프로세스로 돌아오면 trap 함수가 끝이 나고 trap을 호출했던 alltraps 함수로 돌아와 이후 처리를 계속한다.

자연스럽게 trapret 함수로 넘어가 트랩 프레임을 복원하고 iret를 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

iret 명령어는 아래와 같은 명령들을 수행한다 eip, esp 레지스터 값을 복원하고 권한 레벨을 0에서 3으로 바꾸어주는 등의 역할을 수행한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

위에서 보았던 trapframe 구조체의 밑 부분을 보면 iret의 명령 수행과정 순서대로 구성되어 있는 것을 확인할 수 있었다.  
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이를 바탕으로 trapframe의 eip의 값을 바꾸어 놓으면 iret 과정에서 해당 주소로 이동할 것이라는 것을 예상할 수 있었다.

타이머 인터럽트가 발생했을 때 현재 권한 레벨(CPL)에 따라 현재 트랩 프레임에서 레지스터가 담고 있는 값이 다를 수 있다.

현재 권한 레벨은 Ring 0, Ring 3가 있는데 현재 권한 레벨의 값은 cs레지스터 하위 2비트에 저장되어 있다. 00의 경우 Ring 0, 11인 경우 Ring 3라는 의미이기 때문에   
tf->cs & 3 == 0 이면 커널 권한에서 인터럽트가 발생하고 tf->cs & 3 != 0 이면 유저 권한에서 인터럽트가 발생한 것을 확인할 수 있다.

이를 이용해 uthread 명령어를 실행하며 인터럽트 발생 빈도를 확인했을 때 거의 대부분의 경우에서 Ring 0일 때 타이머 인터럽트가 발생하는 것을 알 수 있었다.

printf의 경우 I/O 이기 때문에 타이머 인터럽트가 발생했을 때 유저 권한에서 발생할 확률보다 I/O작업을 하고 있는 커널 레벨 권한에서 발생했을 확률이 높다고 결론을 내렸다.

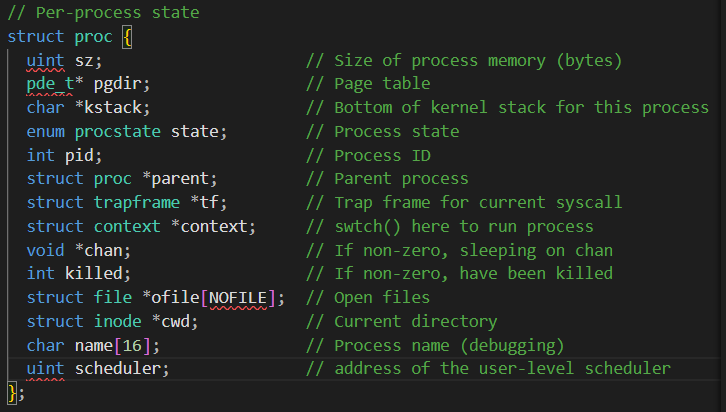
유저 권한에서 타이머 인터럽트가 발생했으면 매개변수로 들어온 tf 구조체에 uthread에서 작업하던 내용이 저장 되어있지만 커널 권한에서 타이머 인터럽트가 발생했다면 tf 구조체에는 커널 레벨에서 작업한 내용이 저장되어 있다. uthread에서의 작업은 시스템 콜로 넘어 갈 때 아래 코드와 같이 트랩 구조체를 현재 프로세스의 트랩에 저장했기 때문에 myproc()->tf을 참조해야 한다.



정리하면 tf->cs & 3 == 0이면 myproc()->tf->eip를 참고하고  
tf->cs&3 != 0이면 tf->eip를 참고하면 된다.

### 수행

제일 먼저 proc.h PCB 구조체 마지막에 uint scheduler를 추가했다.



이후 uthread\_init 함수를 시스템 콜로 등록하였다.

syscall.h

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

syscall.c

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

폰트, 텍스트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

usys.S

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

defs.h

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

user.h

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

sysproc.c

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

proc.c

매개변수로 받은 함수의 주소를 현재 프로세스의 scheduler로 전달해주었다.

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

trap.c

우선 현재 프로세스를 가져와서 현재 프로세스에 등록된 스케줄러가 있을 때 실행했다.

유저 권한에서 트랩이 발생했을 때 tf->eip의 값을 바꾸면 계속해서 메모리 접근 오류로 인터럽트가 발생해 프로세스가 죽었다.

그래서 유저 권한의 트랩의 경우에는 발생 빈도가 매우 적기 때문에 커널 권한에서 타이머 인터럽트가 발생했을 때에만 eip의 값을 바꿨다.

단순히 eip의 값만 바꾸면 기존 실행하던 eip의 값이 사라지기 때문에 함수 call을 할 때처럼 현재 eip의 값을 스택에 저장하고 eip를 바꾸는 방법을 시도했었다.  
하지만 계속해서 인터럽트가 발생해서 프로세스가 죽었다.

결국 커널 권한에서 타이머 인터럽트가 발생했을 때 curproc->tf->eip의 값만 바꾸는 것으로 코드를 짰다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

uthread.c

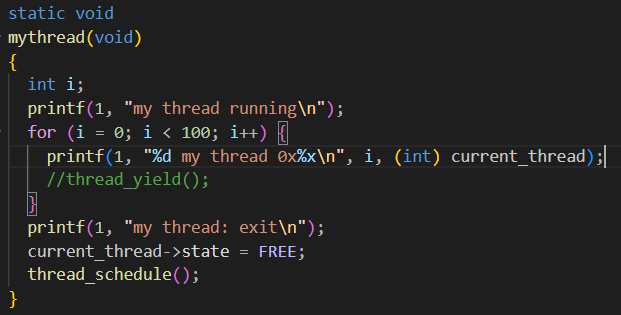
thread\_init에 uthread\_init 시스템 콜을 호출했다. thread\_schedule 함수가 더 밑에 있어 위에 thread\_schedule 함수의 원형도 선언했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



mythread 함수에서는 thread\_yield를 주석 처리하고 디버깅을 위해 printf에 호출 횟수도 추가했다.



thread\_yield에서 기존 RUNNING이던 쓰레드를 RUNNABLE로 바꿔주는 부분이 있는데 그 부분을 thread\_schedule 함수 맨 앞에 만들어주었다.

현재 쓰레드가 0번째 쓰레드가 아니면서 RUNNING 중이면 쓰레드 스위치를 위해 RUNNABLE로 바꾼다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

더 이상 RUNNABLE인 쓰레드가 없어 프로세스를 종료하는 조건문에서 uthread\_init(0)를 통해 프로세스의 스케줄러를 초기화 하여 더 이상 타이머 인터럽트가 발생했을 때 스케줄러를 호출하지 못하게 했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

타이머 인터럽트가 발생할 때 마다 기존 출력을 중단하고 쓰레드 스위칭을 하는 것을 확인할 수 있다. 또한 100번씩 출력하고 무사히 종료하는 것을 확인할 수 있다.