CDA0017: Operating Systems

Donghyun Kang (donghyun@changwon.ac.kr)

NOSLab (https://noslab.github.io)

Changwon National University

Issues in CPU virtualization

- CPU 시간을 다수의 프로세스가 나누어 씀 (time sharing)
  - 성능 저하
  - 제어 문제
    - CPU에 대한 통제를 유지하면서 프로세스를 수행함

제한적 직접 실행

- 프로그램을 CPU에서 직접 실행

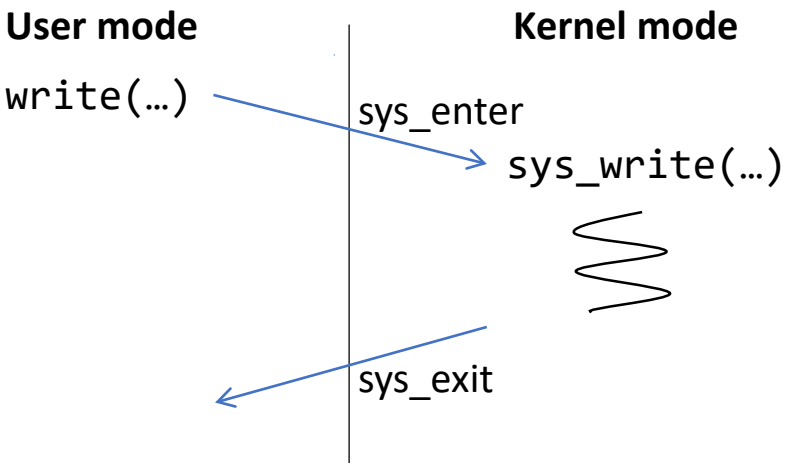
- 문제점
  - 문제 1: 악의적인 H/W 독점 가능
  - 문제 2: 시분할 (time sharing) 구현이 어려움

운영체제	프로그램
프로세스 목록의 항목을 생성	
프로그램 메모리 할당	
메모리에 프로그램 탑재	
argc/argv를 위한 스택 셋업	
레지스터 내용 삭제	
call main() 실행	
	main() 실행
	main에서 **return** 명령어 실행
프로세스 메모리 반환	
프로세스 목록에서 항목 제거	

〈그림 9.1〉 **직접 실행 프로토콜(제한 없음)**

문제 1: 제한된 연산

- H/W 접근 권한 제한으로 악의적인 H/W 독점 방지
  - 사용자 모드 (user mode)
  - 커널 모드 (kernel mode)

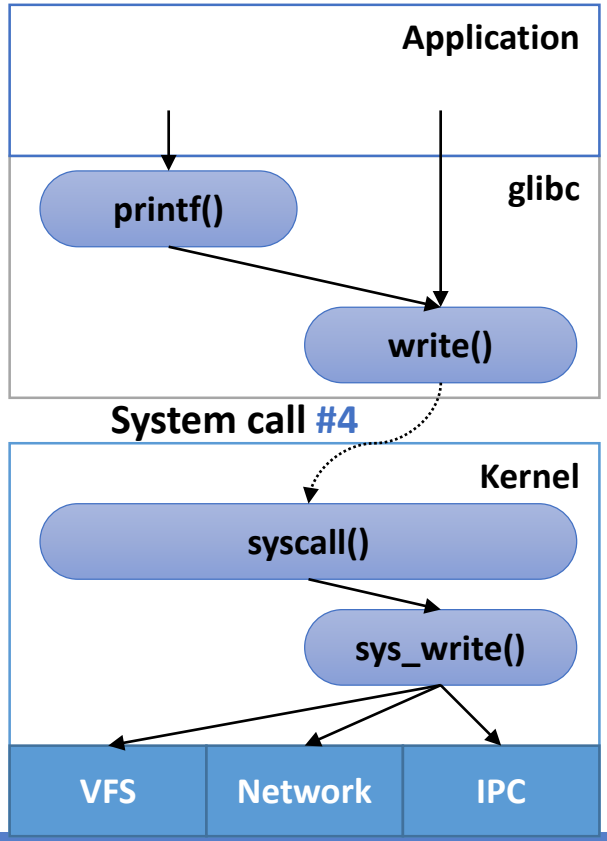


sys_write() 함수의 위치는 어떻게 찾아가나?

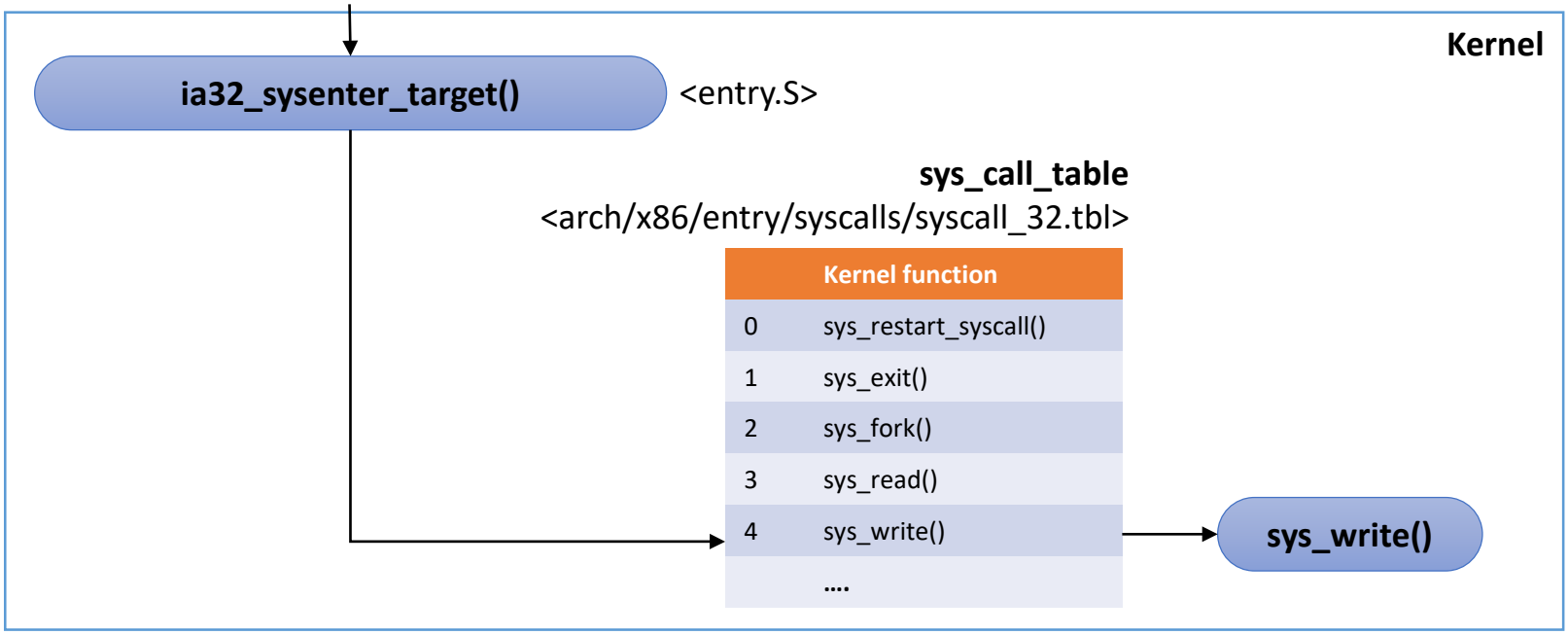
시스템 콜 (system call)

- 시스템 콜은 표준 C 라이브러리 (standard C library) 기반으로 구현됨
  - libc, glibc, uclibc, …
- 시스템 콜은 번호 기반으로 호출 됨
  - System call table

Number	Kernel function
0	sys_restart_syscall()
1	sys_exit()
2	sys_fork()
3	sys_read()
4	sys_write()



리눅스에서의 시스템 콜 처리



cf. http://articles.manugarg.com/systemcallinlinux2_6.html

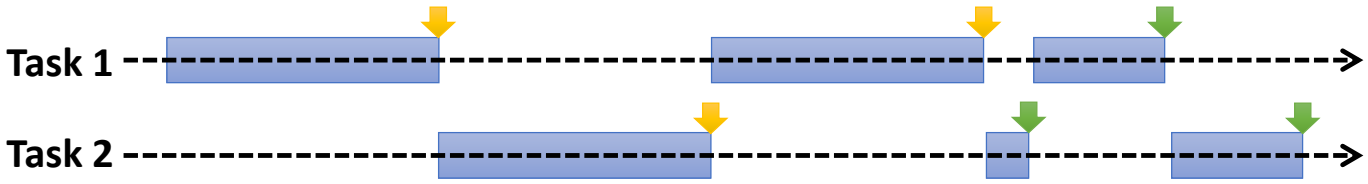
제한된 연산의 프로토콜

운영체제 @부트 (커널 모드)	하드웨어	
트랩 테이블을 초기화한다		
	syscall 핸들러의 주소를 기억한다	
운영체제 @실행 (커널 모드)	**하드웨어**	**프로그램 (사용자 모드)**
프로세스 목록에 항목을 추가한다 프로그램을 위한 메모리를 할당한다 프로그램을 메모리에 탑재한다 argv를 사용자 스택에 저장한다 레지스터와 PC를 커널 스택에 저장한다 **return-from-trap**		
	커널 스택으로부터 레지스터를 복원한다 사용자 모드로 이동한다 main으로 분기한다	
		main()을 실행한다 ... 시스템 콜을 호출한다 운영체제로 **트랩**한다
	레지스터를 커널 스택에 저장한다 커널 모드로 이동한다 트랩 핸들러로 분기한다	
트랩을 처리한다 syscall의 임무를 수행한다 **return-from-trap**		
	커널 스택으로부터 레지스터를 복원한다 사용자 모드로 이동한다 트랩 이후의 PC로 분기한다	
		main에서 리턴한다 **trap(exit())**를 통하여
프로세스의 메모리를 반환한다 프로세스 목록에서 제거한다		

〈그림 9.2〉 **제한적 직접 실행 프로토콜**

문제 2: 프로세스 간 전환

- 시분할 방식 (time sharing)



- 언제?

협조 방식: 시스템 콜 대기

- 프로세스가 yield 시스템 콜을 운영체제가 호출하면 제어권 획득
  - 파일 I/O, 메시지 전송, 새로운 프로세스 생성
- 비정상적인 행위가 발생하면 운영체제가 제어권 획득
  - 다른 프로세스의 메모리 접근
  - 0으로 나누기 연산 수행

프로세스가 무한 루프를 수행한다면?

비협조 방식: 운영체제가 전권을 가짐

- 타이머 인터럽트 (timer interrupt) 이용
  - 부팅 (booting) 시점에 타이머 인터럽트 핸들러 (interrupt handler) 등록
  - 수 밀리 초마다 타이머 인터럽트 발생
  - 현재 수행 중인 프로세스 중단 후 인터럽트 핸들러 수행
    - 커널 스택 (kernel stack)에 현재 수행 중인 프로세스 상태 저장
    - return-from-trap 명령어가 커널 스택의 내용 복구

문맥 교환 (context switch)

운영체제 @부트 (커널 모드)	하드웨어	
트랩 테이블을 초기화 한다		
	syscall 핸들러, 타이머 핸들러의 주소를 기억한다	
인터럽트 타이머를 시작시킨다		
	타이머를 시작시킨다 X msec 지난 후 CPU를 인터럽트한다	

운영체제 @실행 (커널 모드)	하드웨어	프로그램 (사용자 모드)
		프로세스 A
		…
	타이머 인터럽트 A의 레지스터를 A의 커널 스택에 저장 커널 모드로 이동 트랩 핸들러로 분기	
트랩을 처리한다 **switch()** 루틴 호출 A의 레지스터를 A의 proc 구조에 저장 B의 proc 구조로부터 B의 레지스터를 복원 B의 커널 스택으로 전환 **return-from-trap (B 프로세스로)**		
	B의 커널 스택을 B의 레지스터로 저장 사용자 모드로 이동 B의 PC로 분기	
		프로세스 B
		…

〈그림 9.3〉 제한적 직접 실행 프로토콜(타이머 인터럽트)

Q&A