**시스템 프로그래밍 과제 #1 보고서**

2019147542 하동현

1. **사전 조사 보고서**
2. **스케줄링 관련 커널 자료구조 분석**
3. struct task\_struct 구조체

task\_struct 구조체에는 프로세스나 스레드가 가지고 있는 수많은 정보들이 담겨 있고, 그만큼 다양한 자료 구조를 사용하고 있다. 사전 조사에서는 과제에서 사용될 멤버 변수 위주로 어떤 변수가 있으며 그 변수가 무슨 자료구조를 사용하고 누가 참조하게 되는지 위주로 사전 조사를 진행해 보았다.

1. char comm[TASK\_COMM\_LEN]



comm 이라는 변수는 task의 command, 즉 task의 이름 정보를 가지고 있다. 이를 참조하여 task의 이름 정보를 받아올 수 있을 것이다.

1. Pidt pid



pid\_t라는 구조체로 선언된 pid 변수에 Process ID 정보가 들어가 있다. pid\_t는 typedef를 통해 int로 정의되어 있으므로 정수형으로 정보를 저장하면 될 것이다.

1. int prio



task의 우선순위, 즉 가중치를 저장하는 변수이다. 일반 프로세스는 100~140 사이의 값을 가질 수 있다.

1. u64 start\_boottime

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

부팅 이후로부터 task가 시작된 시간을 기록하는 변수이다.



kernel/fork.c의 copy\_process() 함수에서 이 변수에 값이 할당된다. 즉, 프로세스를 fork하여 시작하는 순간 ktime\_get\_boottime\_ns() 함수를 통해 프로세스 시작 시간을 기록하는 것이다. start\_time과 start\_boottime의 차이는, start time은 프로세스의 총 실행 시간을 기록한 것이라면, start\_boottime은 부팅 시간으로부터 언제 시각에 시작되었는지를 기록한 것이다. 즉 과제 명세서에 적힌 대로 start\_boottime 변수를 참조해야 할 것이다. 또한 이는 ns 단위로 기록되므로 1000000으로 값을 나눠 주어야 ms로 나타낼 수 있을 것 같다.

1. const struct sched\_class \*sched\_class

sched\_class 변수는 task가 어떤 스케줄 방식을 따를 건지에 대한 정보가 들어있다. sched\_class는 sched\_entity와 함께 스케줄러 동작에 중요한 역할을 하는 것으로 보인다. 그러므로 아래에서 더 자세히 조사하려고 한다.

1. struct sched\_entity 구조체

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

sched\_entity는 task\_struct 구조체 안에서 se라는 변수 이름으로 선언된 구조체이다. 이 구조체는 CFS 스케줄러가 동작할 때 Process 자신이 자신에 대한 weight, runtime, 우선순위, RB-tree에서 자신의 위치 정보 등을 알아야 하기 때문에 선언된 구조체이다. 즉, CFS 스케줄러를 동작시키기 위해 task\_struct 안에 새로운 자료구조를 만든 것이다. 주요한 멤버 변수를 자세히 살펴보려고 한다.

1. u64 vruntime

vruntime은 CFS 스케줄러에서 task를 리스케줄링 하거나 배정할 때 매우 중요한 요소이다. vruntime은 지금까지 task가 cpu에서 실행된 시간을 기록하는데, 이는 절대적인 시간이 아닌 그 프로세스의 우선순위까지 고려한 시간이다. vruntime의 계산 방식은 다음과 같다.

텍스트, 폰트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

즉, 우선순위가 높은 프로세스는 더 큰 weight를 가지고, 이는 우선순위가 낮은 프로세스(=weight가 작은 프로세스)보다 분모가 커져 시간이 더 느리게 늘어나는 것처럼 계산된다. 즉, 우선순위가 높은 프로세스는 vruntime이 느리게 늘어나므로 cpu에서 작업할 수 있는 시간이 더 늘어나는 것이다. CFS 스케줄러는 리스케줄링 하는 순간에 run queue에 있는 task 중 가장 작은 vruntime을 가진 task를 다음에 실행할 task로 선택하게 된다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

공식에 있는 load weight는 kernel/sched.c와 sched/core.c 에 정의되어 있으며, 우선순위에 따른 weight를 계산한다. 이는 prio가 1 감소할 때마다 약 1.25배씩 증가한다. 이 weight는 time slice를 계산 할 때도 사용된다.

1. struct cfs\_rq

텍스트, 스크린샷, 폰트, 평행이(가) 표시된 사진

자동 생성된 설명

cfs\_rq 구조체는 cfs 방식으로 스케줄 되어야하는 task들을 모아놓은 run queue이다. cfs 스케줄 방식을 사용하기 위해, 내부에는 이 run queue에 저장된 task들의 가중치를 node로 가지는 RB Tree 구조를 저장하고 있다. 또한 이 run queue의 task들의 load를 합한 값을 struct load\_weight load에 저장한다. nr\_running에는 run queue에 존재하는 task의 수를 기록한다.

1. const struct sched\_class 구조체

sched\_class 구조체는 스케줄러 동작 방식을 정의하고, sched\_class 자료구조를 가지는 각각의 변수(스케줄러 방식을 알리는)를 여러 파일에 선언한다. 그 위치와 세부 사항은 L02 강의노트를 참고하였다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

각 sched\_class들은 고유의 스케줄 방식을 따르며 policy라는 변수에 따라서도 작동이 달라지게 된다.

1. rt\_sched\_class in linux/sched/rt.c

텍스트, 영수증, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

real-time 스케줄링 방식을 가지는 sched\_class이다.

1. fair\_sched\_class in linux/sched/fair.c

텍스트, 영수증, 폰트이(가) 표시된 사진

자동 생성된 설명

현재 리눅스에서 가장 많이 쓰이는 CFS 방식을 따르는 sched\_class이다.

1. idle\_sched\_class in linux/sched/idle\_task.c

텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명

다른 task에 영향을 주는 것을 피하는 방식을 위한 스케줄링 클래스이다.

아래는 강의노트에 존재하는 sched\_class 구조체가 가지는 함수들의 목록인데, 중요해 보이는 몇몇의 코드만 찾아보았다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. put\_prev\_task(), set\_next\_task

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

1. enqueue\_task(), dequeue\_task()



두 함수를 통해 현(prev) 프로세스를 run queue에 넣고, rb tree의 가장 왼쪽에 있는 node, 즉 vruntime이 가장 작은 node를 dequeue하여 스케줄링 하는 함수들이다.

1. **schedule() 함수의 동작 과정**

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

리눅스 스케줄러의 개략적인 동작 과정[[1]](#footnote-1)

1. **RT 의 schedule**

RT 스케줄러는 0~99의 priority를 가지는 rt task를 스케줄링 할 때 사용된다. 이스케줄러는 어떠한 작업이든 똑같은 시간 내에 끝날 수 있도록 하기 위해 고안된 스케줄러 이다. 이 task들은 SCHED\_FIFO 정책이나 SCHED\_RR 정책으로 rt 스케줄러에서 동작할 수 있게 한다. SCHED\_FIFO 방식은 first in first out 말 그대로 처음 들어온 작업을 먼저 시행 후, 다음 작업을 수행하는 정책이다. preemption이 없는 한, 한번 cpu를 점유한 task는 계속해서 작업한다. 즉, time slice가 존재하지 않는 것이다. 이에 반해 SCHED FIFO 방식은, FIFO 방식과 큰 차이는 없으나 priority에 따라 time slice가 배정이 되고, 배정받은 time slice가 끝나게 되면 교체되게 된다.

1. **CFS 의 schedule**

CFS 스케줄러는 용어 그대로 모든 task를 공정한 방식으로 할당하겠다는 취지를 가진 스케줄러이다. CFS는 SCHED\_NORMAL, SCHED\_BATCH, SCHED\_IDLE의 세 가지 정책을 가지고 있다. vruntime이라는 변수를 update\_curr()이라는 함수를 통해 계속해서 실시간으로 계산하여, 이를 RB tree에 저장한다. RB Tree는 로그 기반의 시간 복잡도를 가진 자료구조로, 빠른 속도를 가져 tree 자료구조에서 많이 사용되는 방식이다. CFS 스케줄러는 run queue에서 vruntime이 가장 작은 task를 다음 작업으로 할당하게 된다. 가장 작은 vruntime을 가진 노드가 RB tree의 가장 왼쪽 node에 저장되므로, 스케줄러는 새로운 task를 배정하게 될 때 그 node만을 확인하면 된다. 그러면 이제 간단한 순서도를 참조하여 CFS 스케줄러 함수의 동작 과정을 알아보자.

CFS 스케줄러는 크게 두 챕터의 과정으로 나누어져 있다. 먼저, 현재 작업중인 task가 time slice를 전부 소진하였는지를 체크하여 그 task가 preempt 되어야 하는지 확인하는 Checking Preemptability 과정이 있고, 두번째는 preempt 되어야 한다는 게 결정되었을 때, schedule() 함수를 호출하여 task를 교체하는 과정이다.

1. Checking Preemptability

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

먼저 scheduler\_tick() 함수를 통해 rq->curr에 접근하여 현재 실행중인 task의 정보를 가져온다. 만약 현재 task가 CFS 스케줄러를 사용한다면, task\_tick\_fair() 함수를 통해 entity\_tick()을 호출한다. 동시해 update\_curr()을 통해 vruntime을 업데이트한다. entity\_tick()은 cfs\_rq의 nr\_running, 즉 위에서 먼저 살펴본 듯이 run queue의 task가 1개 초과라면 check\_preempt\_tick() 함수를 호출한다. check\_preempt\_tick() 함수를 통해 현재 프로세스가 타임 슬라이스를 전부 소진했다고 판단되면 해당 프로세스가 preempted 될 프로세스, 즉 교체가 되어야 할 프로세스라고 판단한다. 이는 TIF\_NEED\_RESCHED flag를 설정함으로써 schedule() 함수가 작동될 수 있게 한다.

1. Invoking schedule()

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

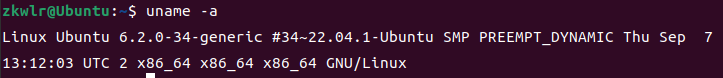
자동 생성된 설명

L02 강의노트에 적힌 순서도를 참고하여 탐구하여 보았다.

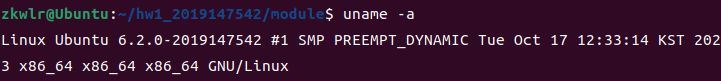
TIF\_NEED\_RESCHED가 설정되면, 먼저 schedule() 함수를 호출한다. 그 후, 현재 task를 RB\_TREE에 넣는다. RB\_tree에는 vruntime의 공식에 따라 실시간으로 계산된 vruntime의 값을 가지는 task가 node로 존재한다. 이 때의 RB\_Tree의 leftmost node를 뽑아내어 다음에 실행될 task로 설정한다. 이후 스케줄링이 더 필요하다고 판단되면, 이 과정을 반복하게 된다.

1. **실습 과제 보고서**
2. **개발 환경**
   1. **uname -a 결과**

커널 교체 전



커널 교체 후



* 1. **컴파일러 정보**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

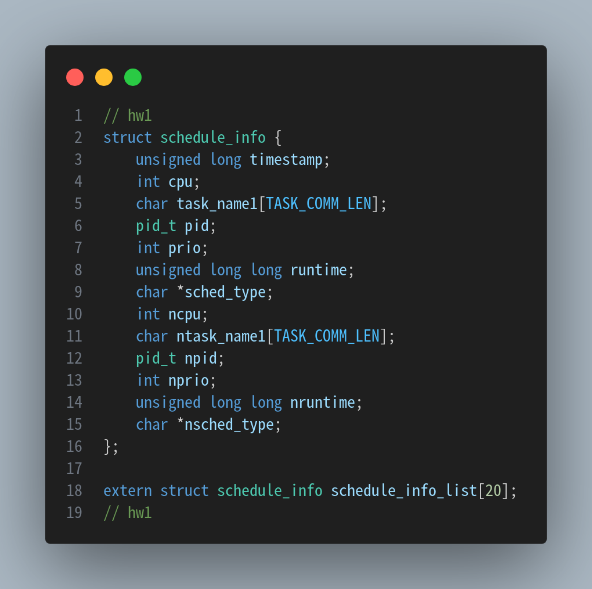
자동 생성된 설명

* 1. **운영체제 및 CPU 정보**

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **커널 모듈 작성 보고서**
2. **커널에 구현한 자료 구조 및 코드**
   * 1. include/linux/sched.h에 구현한 struct schedule\_info



먼저, 모듈이 참조할 sched.h에 스케줄링 된 task의 자료를 담을 구조체 schedule\_info를 만들었다. 명세서에 제시되었던 여러가지 정보들을 구조체가 담고 있다. prev의 정보와 next의 정보를 함께 담기 위해 prev의 정보 변수에 n을 붙혀 next의 정보를 담는 변수도 함께 선언해 주었다. 마지막에는 이 구조체를 담는 20의 길이를 가지는 배열 schedule\_info\_list를 다른 파일에서도 사용할 수 있게 extern으로 선언하였다.

* + 1. kernel/sched/core.c에 구현한 배열- schedule\_info\_list[20]

텍스트, 명함, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

sched.h에 선언한 schedule\_info\_list 배열을 가져와, 정보를 담기 위해 선언하였다. 또한, EXPORT\_SYMBOL()을 통해 다른 모듈에서도 이 배열을 참조할 수 있도록 하였다. 마지막으로, sched\_index를 0으로 설정해 배열에 task를 담을 때 사용할 수 있도록 했다.

* + 1. \_\_schedule() 함수 내부 수정

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

먼저, schedule\_info 구조체인 current\_task\_info 를 만들어, 정보를 담고 sched\_info\_list에 담는 방식으로 코드를 수정하기로 했다. current\_task\_info 의 이름과는 다르게 next task의 정보도 존재하지만, 커널 빌드의 무지막지한 시간 때문에 이 변수명을 그대로 사용하기로 했다. current를 통해 현재 task의 task\_struct 안의 멤버 변수를 가져왔다. 특별히, start\_boottime은 ns 단위로 기록되기에 ms 단위로 출력하기 위해 미리 1000000으로 나눠 주었다. task의 스케줄링 type을 확인하기 위해, 현재 task의 sched\_class가 각각 무엇인지에 따라 sched\_type 변수 안에 문자열로 기록하였다.

이는 next 변수에도 전부 똑 같은 방식으로 저장하였다.

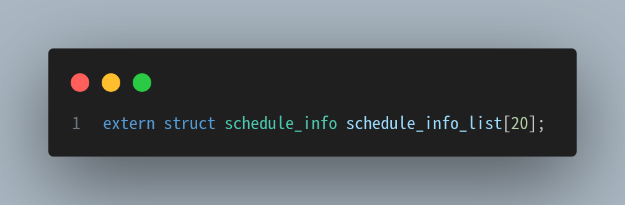
텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

마지막으로, 이렇게 저장된 정보를 가지는 current\_task\_info를 배열에 어떤 식으로 저장할 지 고려하였다. 명세서에서 가장 최근의 20번 \_\_schedule() 동작만을 담으라고 하였으므로, 먼저 20번의 동작, 즉 sched\_index가 20이 되기 전까지는 sched\_index를 index로 갖는 schedule\_info\_list 위치에 task info를 저장하였다. 그 이후에는, 가장 오래된 동작의 task info를 지운 후 한 칸씩 앞으로 당겨 자리를 만들어야 하기에, else 이후의 코드와 같이 구현하였다.

1. **커널 모듈 코드**

예시로 주어진 모듈 코드에서 2가지 부분을 수정하였다.

****

먼저, \_\_schedule() 함수 동작에 따른 task의 정보가 담긴 schedule\_info\_list 배열을 사용하기 위해 위와 같이 선언하였다.



명세서의 제출 형식에 따라, schedule\_info\_list 배열에서 저장된 각 정보를 자료형에 맞게 출력하게 하였다.

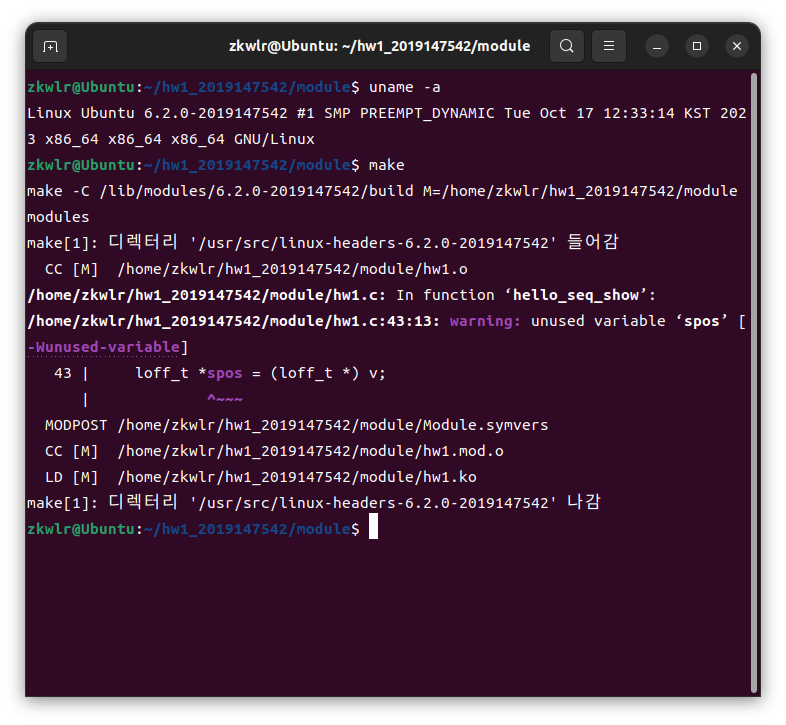
1. **동작 과정**

동작 과정은 위에 코드 분석을 통해 설명하였으므로 순서도를 통해 다시 한번 설명하려고 한다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

1. **실습 과제 결과 검증 및 분석**

****

먼저 make를 통해 hw1.ko가 잘 컴파일 되었다.

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이후 lsmod 명령을 통해 hw1 모듈이 등록된 것을 확인할 수 있었다.

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

이후 cat /proc/hw1 명령을 입력하니, 명세서의 예시대로 trace #19까지 출력되는 것을 확인하였다. CPU의 개수, task가 사용중인 CPU 번호, 스케줄러 종류등이 올바르게 나오는 것을 확인할 수 있었다.

텍스트, 전자제품, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

sudo rmmod hw1을 통해 모듈을 제거하였고, 이후 lsmod를 통해 확인한 결과 hw1 모듈이 제거됨을 확인하였다.



제거 후 cat /proc/hw1 명령을 사용하면 작동하지 않는 것을 확인할 수 있었다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

make clean 명령을 통해 Makefile과 모듈 프로그래밍 구현물을 제외한 나머지 파일이 삭제되었다.

1. **과제를 수행하면서 있었던 문제 및 해결방법**

가장 먼저 부딪혔던 벽은, 커스텀 커널을 빌드하는 과정이었다. 저번 OS의 1차 과제에서 리눅스 커널을 설치하는 과제를 하긴 했지만, 커스텀 커널을 설치하는 것은 또 다른 문제였다. 리눅스 커널의 크기는 내 생각보다도 훨씬 크고 빌드도 오래 걸렸다. 일단 컴퓨터의 용량이 부족하여 SSD를 구매하는 것부터 시작하였다. 이후 소스코드를 수정하고 빌드하는 과정에서 많은 어려움을 겪었다. 일단 한번 한번의 빌드 시간이 길고, 빌드를 중간에 끊었을 때 어떤 문제가 발생할 지도 몰라 중지할 수도 없었다. 이는 첫 컴파일 이후에는 시간이 확실히 줄어서 괜찮았던 것 같다. 또한 빌드를 완료하고 커널을 설치하는 과정도 복잡하여 자료를 찾아보는데 많은 시간을 할애했던 것 같다. 나는 항상 Debian package를 만들지 않고 make-make install modules-make install 명령을 통해 바로바로 리눅스 환경에서 커널을 업데이트해 코드를 수정하였고, 과제 완료 후 .deb 패키지를 생성하니 한결 수월했던 것 같다. 한번은 kernel panic이 발생하여, 가상 환경을 아예 쓰지 못하게 되었다. 다행이도 이전에 snapshot image를 저장해 놓아서, 시간을 매우 아낄 수 있었다. 또한 git 환경을 사용하여 커널 프로그래밍 도중 발생한 여러 문제에 대처할 수 있었다.

코드 내적으로는, 커널의 정보를 모듈 코드를 통해 외부로 가져오는 것에 계속 실패하였다. 이후 symbol을 통해서만 외부에서 정보를 참조할 수 있다는 것을 알게 되었고, 코드를 추가하니 참조를 수월하게 할 수 있었다. 이외에 코드 구현상으로는 별로 어려운 점이 없었던 것 같다.

1. **참고한 문헌/사이트**

시스템 프로그래밍 L02-processscheduling 강의노트

<https://velog.io/@jinh2352/Linux-9-%EC%9D%BC%EB%B0%98-%ED%83%9C%EC%8A%A4%ED%81%AC-%EC%8A%A4%EC%BC%80%EC%A4%84%EB%A7%81-CFS> 일반 테스크 스케줄링

<https://information.koreainfoguide.com/entry/%EC%8A%A4%EC%BC%80%EC%A5%B4%EB%9F%AC-%EA%B8%B0%EC%B4%88> 리눅스 스케줄러 기초

<https://gist.github.com/hipiphock/11fa1c1e5dad03766881a1219d0aa692> hipiphock/scheduler\_overview.md

<https://jamcorp6733.tistory.com/201> 리눅스 CFS 스케줄러

<https://elixir.bootlin.com/linux/v6.2/source> 리눅스 소스코드 참조 사이트

<https://ndb796.tistory.com/534> 우분투 커널 소스코드 수정 및 빌드하는 방법

<https://blackinkgj.github.io/FedoraKernelInstall/> 리눅스에 커스텀 커널 설치 방법

<https://extrememanual.net/9326> 우분투 오래된 이전 커널 삭제 방법

<https://ryotta-205.tistory.com/12> 커널 컴파일(우분투)

<https://velog.io/@mythos/Linux-Tutorial-9-%EC%BB%A4%EB%84%90-%EB%AA%A8%EB%93%88-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D> 커널 모듈 프로그래밍

<https://jhnyang.tistory.com/299> C 전처리기 지시어

<https://codingcoding.tistory.com/193> 리눅스 proc 파일 시스템 이해하기

<https://waterfogsw.tistory.com/16> Linux kernel Structure

<https://blog.naver.com/jeix2/80007580165> task\_struct 구조체

<https://darkengineer.tistory.com/119> 테스크 디스크립터 자료구조

<https://poplinux.tistory.com/105> extern : 개발한 함수를 외부에 공개하는법

<https://blog.naver.com/cre8tor/90193630398> EXPORT\_SYMBOL

<https://velog.io/@mythos/Linux-Tutorial-6-%EB%8D%B0%EC%9D%B4%ED%84%B0-%ED%83%80%EC%9E%85> 리눅스 커널의 다양한 데이터 타입

1. <https://information.koreainfoguide.com/entry/%EC%8A%A4%EC%BC%80%EC%A5%B4%EB%9F%AC-%EA%B8%B0%EC%B4%88> 스케줄러 기초, 2019.9.14 [↑](#footnote-ref-1)