

# 운영체제( OS ) 2021 보고서

## 보고서 제출서약서

나는 송실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.

1.1. 나는 동료의 보고서를 베끼지 않았습니다.

1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.

2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)

3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.

4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	운영체제 2021
과제명	리눅스 커널의 스케줄링 정책을 (기본 CFS, 조정된 NICE 값을 적용한 CFS) 확인 할 수 있는 프로그램 작성
담당교수	김 철 홍 교 수
제출인	컴퓨터학부 20162449 김상현
제출일	2021년 11월 7일

# 차 례

1장 프로젝트 동기/목적

2장 설계/구현

3장 수행결과(구현 화면 포함)

4장 결론 및 보충할 것

## 1장 프로젝트 동기/목적

리눅스 커널의 스케줄링 정책을 이해 하고 NICE 값을 조정한 CFS를 확인 할 수 있는 프로그램을 작성한다. 이를 통해 리눅스 기본 스케줄링 CFS 에 대해 이해한다.

- 기본 CFS 와 nice 값을 조정한 CFS 구현 완료
- 커널 소스 수정을 통한 커널 수정을 통한 RealTime FIFO 스케줄러 구현

첨부한 source 파일은 CFS 2가지 구현을 증명하기 위한 cfs.c 와 make 사용 미숙으로 커널 소스에서 수정했던 /usr/src/linux-5.11.22/kernel/sched/core.c 와 /usr/src/linux-5.11.22/.config 파일을 첨부합니다.

## 2장 설계/구현

리눅스 커널의 기본 스케줄링 정책인 CFS 에 대해서 알아보자.

기본 개념 :

‘CFS’란 n개의 프로세스에게 각각의 일정한 시간을 제공하는 스케줄링 방법이다.

이 때 우선 순위를 정하기위해 커널은 nice 값을 각각 부여하는데 이 nice 값을 각 process의 비율로 설정한 다. 이 비율을 기준으로 timeslice를 할당 받는 것이다.

NICE 값은 (-19) ~ 20 사이의 값을 가진다.

NICE 값을 유저가 변경할 수 있게 하는 시스템 콜이 존재하는데 nice()가 이에 해당한다.

```
NICE(1)                                     User Commands                               NICE(1)

NAME
    nice - run a program with modified scheduling priority

SYNOPSIS
    nice [OPTION] [COMMAND [ARG]...]

DESCRIPTION
    Run COMMAND with an adjusted niceness, which affects process scheduling. With no COM-
    MAND, print the current niceness. Niceness values range from -20 (most favorable to
    the process) to 19 (least favorable to the process).

    Mandatory arguments to long options are mandatory for short options too.

    -n, --adjustment=N
        add integer N to the niceness (default 10)

    --help display this help and exit

    --version
        output version information and exit

NOTE: your shell may have its own version of nice, which usually supersedes the version
described here. Please refer to your shell's documentation for details about the
options it supports.
```

사실 nice 값 만들 가지고 운영체제가 수행 순서를 정하지는 않는다. PI 값이 라는 새로운 값이 존재하는데 이는 운영체제가 직접 결정하는 것이므로 바꿀 수 없다. 그래서 운영체제는 변경되는 nice 값을 참고하여 수 행 순서를 정하게 된다.

기본 적인 CFS로 이루어진 스케줄링 방식은 단순히 21개의 자식 프로세스를 생성하여 작업을 부여하면  
간 단히 보여진다.

NICE 값을 변경한 값 은 앞서 설명한 nice() 함수를 이용하여 가장 많은 작업을 하는 프로세스들에는  
가장 앞선 우선순위를 가장 적은 작업량의 프로세스들에게는 가장 느린 우선순위를 부여할 것이다.  
기존의 CFS 의 결과와 비교해보자.

### 3장 수행결과

실행전 예측한 결과 :

기존의 CFS 의 경우 CPU사용률이 제일 큰 프로세스 순서의 역순으로 작업이 이루어 질 것이다.

Nice 값을 조정한 CFS 의 경우 새로운 우선 순위 값에 따라 작업량이 제일 크지만 우선순위가  
제일 높은 프로세스부터 cpu를 할당 받을 것이다.

결과 :

```
sanghyun@ubuntu:~$ sudo ./a.out
(1) basic CFS , (2) new CFS :1
parent process start!
[hard work] start 12549 ( 0 )
[hard work] start 12550 ( 0 )
[hard work] start 12551 ( 0 )
[hard work] start 12552 ( 0 )
[hard work] start 12553 ( 0 )
[hard work] start 12554 ( 0 )
[hard work] start 12555 ( 0 )
[normal work] start 12558 ( 0 )
[small work] start 12564 ( 0 )
[normal work] start 12560 ( 0 )
[small work] start 12563 ( 0 )
[small work] start 12565 ( 0 )
[normal work] start 12559 ( 0 )
[normal work] start 12557 ( 0 )
[normal work] start 12561 ( 0 )
[normal work] start 12562 ( 0 )
[small work] start 12568 ( 0 )
[small work] start 12566 ( 0 )
[small work] start 12569 ( 0 )
[normal work] start 12556 ( 0 )
[small work] start 12567 ( 0 )
[small work] end 12564 ( 0 )
[small work] end 12565 ( 0 )
[small work] end 12566 ( 0 )
[small work] end 12569 ( 0 )
[small work] end 12568 ( 0 )
[small work] end 12567 ( 0 )
[small work] end 12563 ( 0 )
[normal work] end 12557 ( 0 )
[normal work] end 12562 ( 0 )
[normal work] end 12558 ( 0 )
[normal work] end 12561 ( 0 )
[normal work] end 12559 ( 0 )
[normal work] end 12560 ( 0 )
[normal work] end 12556 ( 0 )
[hard work] end 12553 ( 0 )
[hard work] end 12551 ( 0 )
[hard work] end 12554 ( 0 )
[hard work] end 12555 ( 0 )
[hard work] end 12549 ( 0 )
[hard work] end 12550 ( 0 )
[hard work] end 12552 ( 0 )
parent process ends
sanghyun@ubuntu:~$
```

기본 CFS를 통해 실행해 보았을 때 예상결과와 맞게 결과가 도출 되었다.

```

sanghyun@ubuntu:~$ sudo ./a.out
(1) basic CFS , (2) new CFS :2
parent process start!
[hard work] start 12525 ( -19 )
[hard work] start 12526 ( -19 )
[hard work] start 12527 ( -19 )
[hard work] start 12528 ( -19 )
[hard work] start 12529 ( -19 )
[hard work] start 12531 ( -19 )
[hard work] start 12530 ( -19 )
[normal work] start 12533 ( 0 )
[normal work] start 12534 ( 0 )
[normal work] start 12532 ( 0 )
[normal work] start 12538 ( 0 )
[small work] start 12545 ( 19 )
[small work] start 12540 ( 19 )
[normal work] start 12536 ( 0 )
[normal work] start 12535 ( 0 )
[normal work] start 12537 ( 0 )
[small work] start 12544 ( 19 )
[small work] start 12542 ( 19 )
[small work] start 12541 ( 19 )
[small work] start 12539 ( 19 )
[small work] start 12543 ( 19 )
[normal work] end 12532 ( 0 )
[normal work] end 12534 ( 0 )
[normal work] end 12538 ( 0 )
[normal work] end 12533 ( 0 )
[normal work] end 12537 ( 0 )
[normal work] end 12536 ( 0 )
[normal work] end 12535 ( 0 )
[small work] end 12540 ( 19 )
[small work] end 12545 ( 19 )
[small work] end 12543 ( 19 )
[small work] end 12539 ( 19 )
[small work] end 12542 ( 19 )
[small work] end 12541 ( 19 )
[small work] end 12544 ( 19 )
[hard work] end 12531 ( -19 )
[hard work] end 12530 ( -19 )
[hard work] end 12528 ( -19 )
[hard work] end 12525 ( -19 )
[hard work] end 12527 ( -19 )
[hard work] end 12526 ( -19 )
[hard work] end 12529 ( -19 )
parent process ends

```

NICE 값을 조정 한 CFS 의 경우 예상과는 조금 다르다.

[normal work / normal NICE] > [small work / big NICE] > [big work / small NICE]  
 순으로 작업이 완료되었다.

예상 결과 랑은 약간 차이가 있지만 우선순위에 변경에 따른 유의미한 차이를 찾을 수 있었다.  
 운영체 제는 유저가 주어진 프로세스 간의 우선순위 값은 NICE를 참고하여 CPU를 할당하였고 기본  
 결과보다 다른 결과가 도출 되었다.

**\*\* NICE 값을 기존의 0에서 음수로 줄이기 위해서는 무조건 root 권한으로 실행하여야한다!**



## \* 리눅스 커널 수정을 통한 FIFO 스케줄링

kernel/sched/core.c 소스를 확인해보면 sched\_init() 함수를 확인 할 수 있다. 이 함수의 역할은 task\_group 구조체인 root\_task\_group에 CFS 스케줄러 구조체와 RT 스케줄러 구조체 포인터를 할당하는 역할을 한다.

또 CFS 스케줄러와 RT 스케줄러 소스는 컴파일 시점에 선택적 조건으로 컴파일 되는데 커널 소스 컴파일 정의( .config 에 해당한다 )에 CONFIG\_FAIR\_GROUP\_SCHED 가 정의 되어있으면 CFS 스케줄러 소스가 컴파일 되고 CONFIG\_RT\_GROUP\_SCHED가 정의 되어 있으면 RT 스케줄러 소스가 컴파일 되게 된다.

지금 과제에서 묻는 부분은 현재 default 로 저장되어있는 CFS 스케줄링을 FIFO 스케줄링으로 커널 소스 수정을 통해 바꿔야하고 이를 위해서 아래의 그림으로 저장되어 있는 .config 파일을 수정하였다.

```
164 CONFIG_CGROUP_SCHED=y
165 CONFIG_FAIR_GROUP_SCHED=y
166 CONFIG_CFS_BANDWIDTH=y
167 CONFIG_RT_GROUP_SCHED is not set
168 CONFIG_UCLAMP_TASK_GROUP=y
169 CONFIG_CGROUP_PIDS=y
170 CONFIG_CGROUP_RDMA=y
171 CONFIG_CGROUP_FREEZER=y
```

```
165 CONFIG_FAIR_GROUP_SCHED= n
167 CONFIG_TRT_GROUP_SCHED = y
```

이후 kernel/sched/core.c 파일에서 프로세스 생성을 담당하는 sched\_fork 함수를 아래와 같이 수정하였다.

```
3709 int sched_fork(unsigned long clone_flags, struct task_struct *p)
3710 {
3711     unsigned long flags;
3712
3713     __sched_fork(clone_flags, p);
3714     /*
3715      * We mark the process as NEW here. This guarantees that
3716      * nobody will actually run it, and a signal or other external
3717      * event cannot wake it up and insert it on the runqueue either.
3718      */
3719     p->state = TASK_NEW;
3720
3721     /*
3722      * Make sure we do not leak PI boosting priority to the child.
3723      */
3724     p->prio = current->normal_prio;
3725
3726     uclamp_fork(p);
3727     if(p->policy == SCHED_NORMAL || SCHED_RR || SCHED_BATCH || SCHED_DEADLINE || SCHED_IDLE){
3728         p->normal_prio = 50;
3729         p->rt_priority = 50;
3730         p->policy = SCHED_FIFO;
3731         p->static_prio = 50;
3732     }
3733     /*
3734      * Revert to default priority/policy on fork if requested.
3735      */
3736     if (unlikely(p->sched_reset_on_fork)) {
3737         if (task_has_dl_policy(p) || task_has_rt_policy(p)) {
3738             p->policy = SCHED_FIFO;
3739             p->static_prio = 50;
3740             p->rt_priority = 50;
3741             //p->policy = SCHED_NORMAL;
3742             //p->static_prio = NICE_TO_PRIO(0);
3743             //p->rt_priority = 0;
3744         } else if (PRIO_TO_NICE(p->static_prio) < 0)
3745             p->static_prio = NICE_TO_PRIO(0);
3746
3747         p->prio = p->normal_prio = __normal_prio(p);
3748         set_load_weight(p, false);
3749     }
```

```

6037
6038 static int _sched_setscheduler(struct task_struct *p, int policy,
6039                               const struct sched_param *param, bool check)
6040 {
6041     struct sched_attr attr = {
6042         .sched_policy    = policy,
6043         .sched_priority  = param->sched_priority,
6044         .sched_nice      = PRIO_TO_NICE(p->static_prio),
6045     };
6046     if(attr.sched_policy == SCHED_NORMAL || SCHED_RR || SCHED_BATCH || SCHED_DEADLINE || SCHED_IDLE){
6047         attr.sched_priority = 50;
6048         attr.sched_policy = SCHED_FIFO;
6049     }
6050     /* Fixup the legacy SCHED_RESET_ON_FORK hack. */
6051     if ((policy != SETPARAM_POLICY) && (policy & SCHED_RESET_ON_FORK)) {
6052         attr.sched_flags |= SCHED_FLAG_RESET_ON_FORK;
6053         policy &= ~SCHED_RESET_ON_FORK;
6054         attr.sched_policy = policy;
6055     }
6056
6057     return __sched_setscheduler(p, &attr, check, true);
6058 }

```

같은 core.c에서 setscheduler 함수에서 6046 줄부터 3개를 추가하였다.

```

File Edit View Search Terminal Help
CC [M] drivers/md/bcache/ extents.o
CC [M] drivers/net/ethernet/stmicro/stmmac/stmmac_pci.o
CC [M] drivers/net/wireless/st/cw1200/debug.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/hw.o
CC [M] drivers/md/bcache/ io.o
CC [M] drivers/net/ethernet/stmicro/stmmac/dwmac-intel.o
CC [M] drivers/net/wireless/st/cw1200/pm.o
CC [M] drivers/md/bcache/ journal.o
LD [M] drivers/net/ethernet/stmicro/stmmac/stmmac.o
LD [M] drivers/net/ethernet/stmicro/stmmac/stmmac-platform.o
LD [M] drivers/net/ethernet/stmicro/stmmac/stmmac-pci.o
CC [M] drivers/net/ethernet/sun/sunhme.o
CC [M] drivers/net/wireless/st/cw1200/cw1200_sdio.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/led.o
CC [M] drivers/md/bcache/ movinggc.o
CC [M] drivers/net/wireless/st/cw1200/cw1200_spi.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/phy.o
CC [M] drivers/md/bcache/ request.o
CC [M] drivers/net/ethernet/sun/sungem.o
LD [M] drivers/net/wireless/st/cw1200/cw1200_core.o
LD [M] drivers/net/wireless/st/cw1200/cw1200_wlan_sdio.o
LD [M] drivers/net/wireless/st/cw1200/cw1200_wlan_spi.o
CC [M] drivers/net/wireless/ti/wlcore/main.o
CC [M] drivers/md/bcache/ stats.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/rf.o
CC [M] drivers/net/ethernet/sun/cassini.o
CC [M] drivers/md/bcache/ super.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/sw.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/trx.o
CC [M] drivers/md/bcache/ sysfs.o
CC [M] drivers/net/ethernet/sun/niu.o
CC [M] drivers/net/wireless/ti/wlcore/cmd.o
LD [M] drivers/net/wireless/realtek/rtlwifi/rtl8192se/rtl8192se.o
CC [M] drivers/md/bcache/ trace.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192de/dm.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192de/fw.o
CC [M] drivers/net/wireless/ti/wlcore/io.o
CC [M] drivers/md/bcache/ util.o
CC [M] drivers/md/bcache/ writeback.o
CC [M] drivers/net/wireless/ti/wlcore/event.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192de/hw.o
CC [M] drivers/md/bcache/ features.o
CC [M] drivers/net/wireless/ti/wlcore/tx.o
CC [M] drivers/net/ethernet/tehuti/tehuti.o
LD [M] drivers/md/bcache/ bcache.o
CC [M] drivers/md/persistent-data/dm-array.o
CC [M] drivers/net/wireless/realtek/rtlwifi/rtl8192de/led.o

```

이후 커널 컴파일 이후에 `chrt -p pid`를 통해 사용중인 스케줄링 방식을 확인해보면 SCHED\_FIFO가 정상적으로 출력됨을 알 수 있다.

```

sanghyun@ubuntu:~$ sudo ./a.out
(1) basic CFS , (2) new CFS :2
parent process start!
[hard work] start 2138 ( -20 )
[hard work] start 2139 ( -20 )
[hard work] start 2140 ( -20 )
[hard work] start 2141 ( -20 )
[hard work] start 2142 ( -20 )
[hard work] start 2145 ( -20 )
[normal work] start 2150 ( -20 )
[normal work] start 2151 ( -20 )
[hard work] start 2148 ( -20 )
[normal work] start 2153 ( -20 )
pid 2145's current scheduling policy: SCHED_FIFO
pid 2141's current scheduling policy: SCHED_FIFO
pid 2145's current scheduling priority: 50
pid 2140's current scheduling policy: SCHED_FIFO
pid 2139's current scheduling policy: SCHED_FIFO
pid 2141's current scheduling priority: 50
pid 2138's current scheduling policy: SCHED_FIFO
pid 2140's current scheduling priority: 50
pid 2139's current scheduling priority: 50
pid 2138's current scheduling priority: 50
pid 2142's current scheduling policy: SCHED_FIFO
pid 2142's current scheduling priority: 50
[normal work] end 2151 ( -20 )
[normal work] end 2153 ( -20 )
[normal work] start 2154 ( -20 )
[normal work] end 2150 ( -20 )
[normal work] start 2155 ( -20 )
[normal work] end 2154 ( -20 )
[normal work] start 2156 ( -20 )
[normal work] end 2155 ( -20 )
[normal work] start 2157 ( -20 )
[normal work] end 2157 ( -20 )
[small work] start 2158 ( -20 )
[small work] end 2158 ( -20 )

```

## 4장 결론 및 보충 할 점

처음에 예상결과와 실제 결과가 많이 달라서 구현에 실패 했다고 생각하여 nice 값을 바꾸는 다양한 방법을 찾게 되었다.

nice 값을 변경하는 다양한 방법 : nice() , setpriority() , sched\_attr() 등 다양한 방법이 있었다. nice 와 setpriority() 는 시스템 함수로써 존재하고 sched\_attr() 함수는 프로세스들이 가지는 task\_struct를 분석하여 struct sched\_attr 에 담고 이때 nice 값 뿐만 아니라 고정 우선순위, 스케줄링 기법 등 또한 설정할 수 있는 함수이고, <sched.h>에 정의 되어 있다.

리눅스 운영체제에서 default로 사용하는 스케줄링에 대해 이해하였다.