COMP 4511

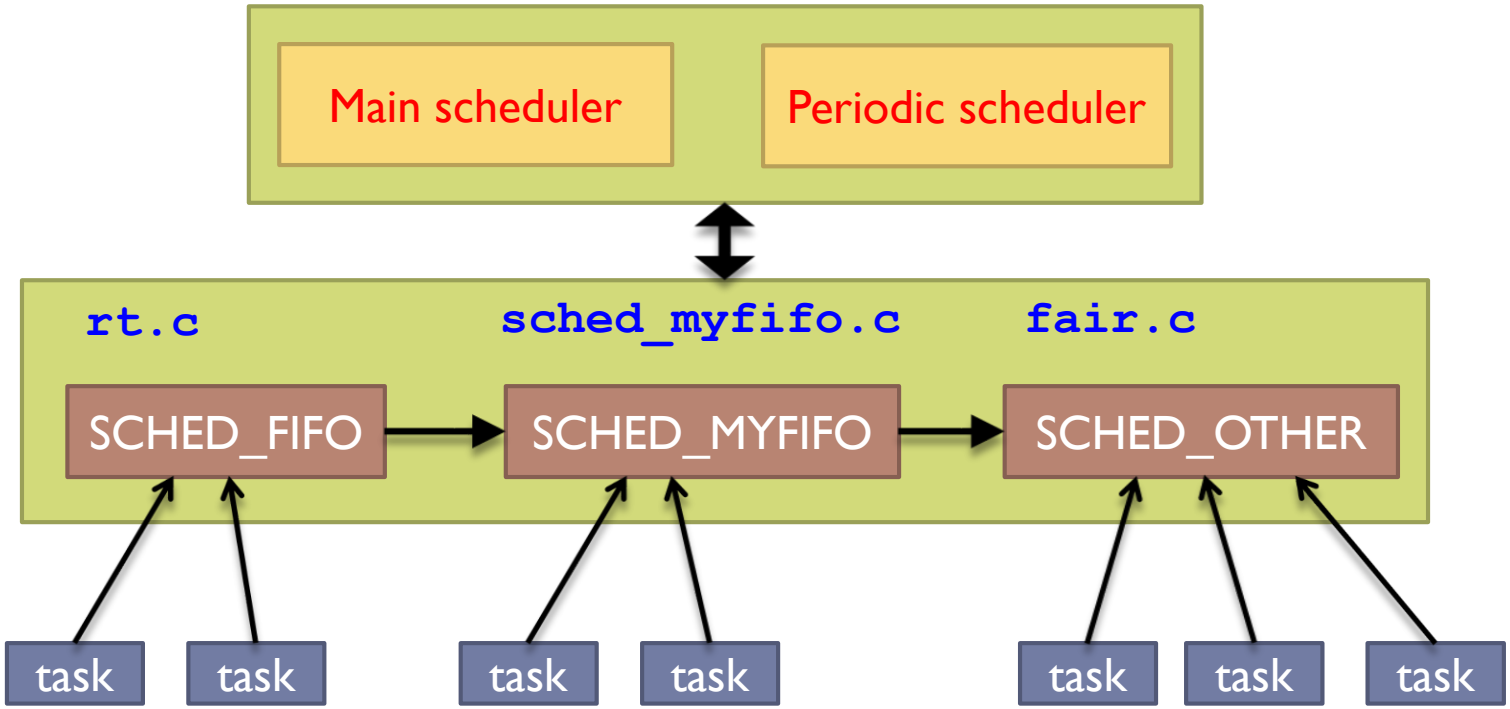
Implementing a First-in-first-out (FIFO) scheduler

Goals

- You will learn:
  - How to implement a new scheduler into the kernel

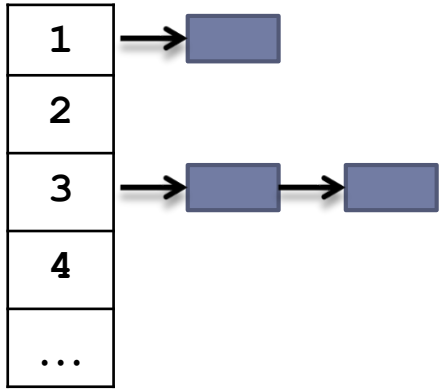
Review of Linux process scheduling

- The introduction of scheduling classes has made the core scheduler quite extensible



Review of Linux process scheduling

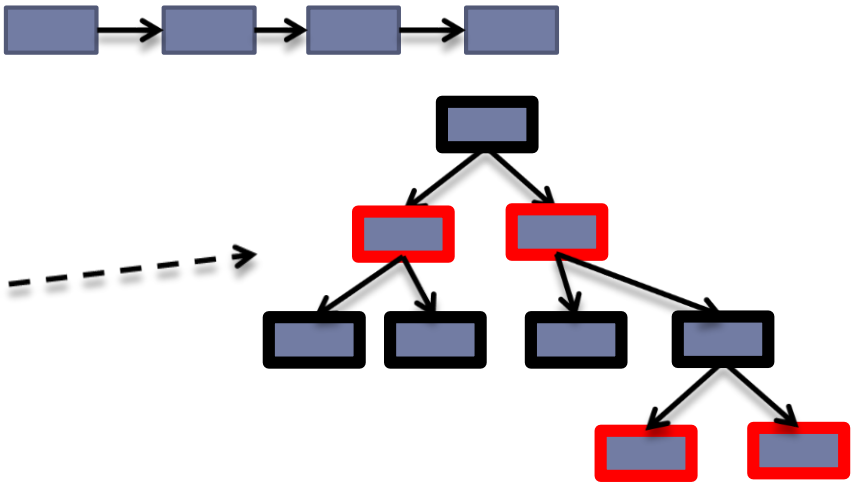
```
struct rq {
        /* Skip … */
        struct cfs_rq cfs;
        struct rt_rq rt;
        struct myfifo_rq myfifo;
};
```



Run queue of SCHED_FIFO

Run queue of SCHED_MYFIFO

Run queue of SCHED_OTHER



Review of Linux process scheduling

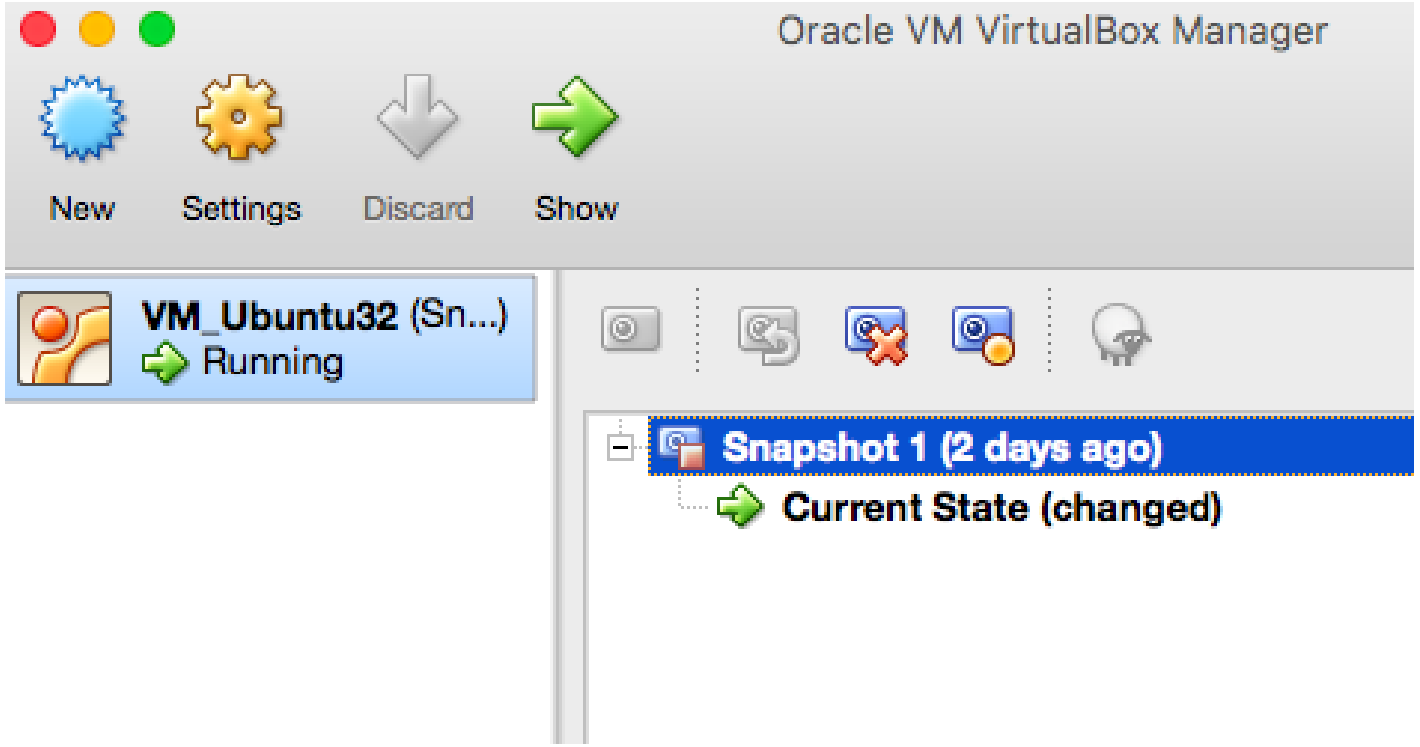
```
struct task_struct {
        /* Skip … */
        struct sched_entity se;
        struct sched_rt_entity rt;
        struct sched_myfifo_entity myfifo;
};
```

- The scheduling entity is the place where a **`task_struct`** is linked with a `runqueue`
  - Embed **`struct list_head`** in the scheduling entity

Implementation of a toy FIFO scheduler

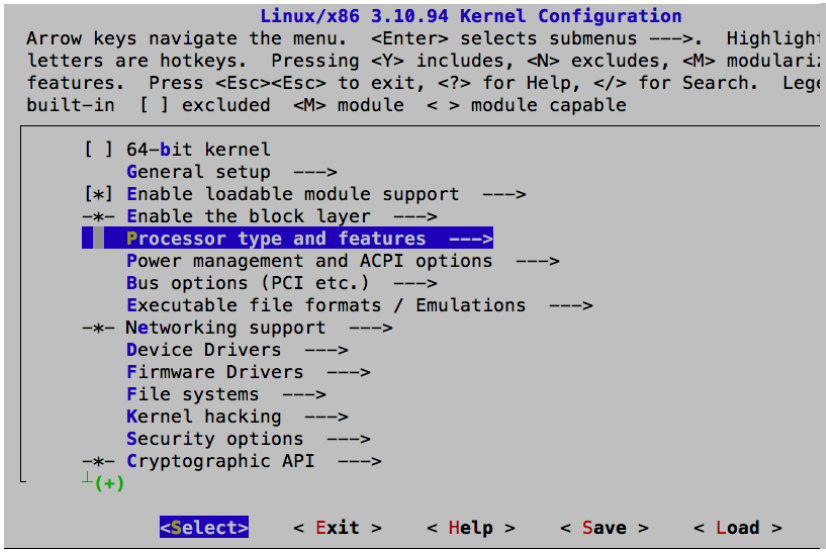
Recommendation: Take a snapshot first!

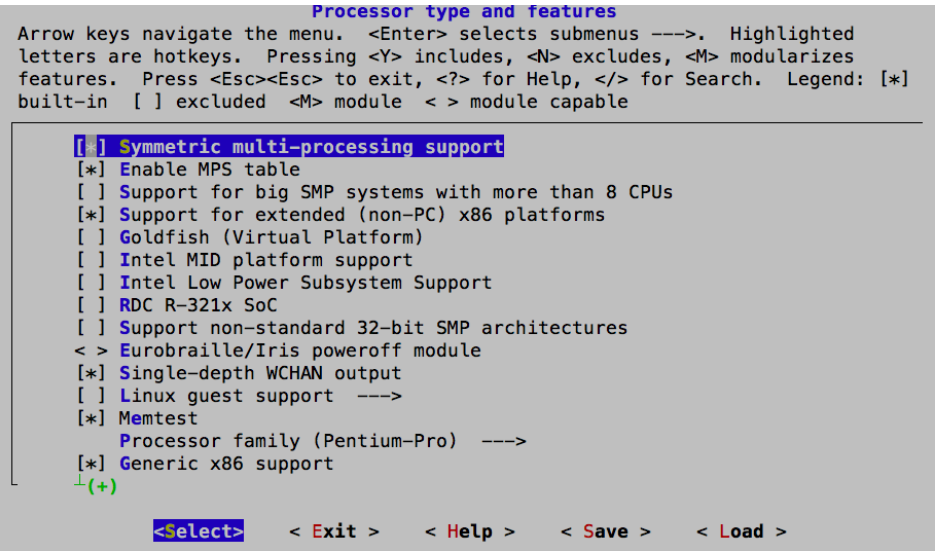
- There are many files to be modified
- Take a snapshot in your VM software
- We can restore from the snapshot to start over



Turn-off SMP support

- Linux kernel supports Symmetric multiprocessing (SMP)
- Enabling SMP will make the scheduler implementation a bit complicated
  - `sudo make menuconfig`
  - Processor type and features -> SMP support -> Press 'N' to disable





Step-1: define scheduling policy no.

- Add the following line

```
#define SCHED_MYFIFO  6
```

after `#define SCHED_IDLE 5`

in `include/uapi/linux/sched.h`

Note: You must put it before the `#endif` directive

```
/*
 * Scheduling policies
 */
#define SCHED_NORMAL            0
#define SCHED_FIFO              1
#define SCHED_RR                2
#define SCHED_BATCH             3
/* SCHED_ISO: reserved but not implemented yet */
#define SCHED_IDLE              5

#define SCHED_MYFIFO            6

/* Can be ORed in to make sure the process is reve
#define SCHED_RESET_ON_FORK     0x40000000

#endif /* _UAPI_LINUX_SCHED_H */
```

Step-2: define the scheduler entity

- Recall that:
  - For NORMAL scheduling class, it has **`struct sched_entity`** for the implementation of CFS
  - For Real-time scheduling class, it has **`struct sched_rt_entity`**
- We define **`struct sched_myfifo_entity`** to encapsulate needed information to help implement the new scheduling policy in **`include/linux/sched.h`**
  - Hint: It is better to add right before **`struct sched_entity`**

```
struct sched_myfifo_entity {
        struct list_head run_list;
    };
```

Add the following to the definition of **`struct`** `task_struct`:

- Hint: It is better to add right after **`struct sched_rt_entity rt;`**

```
struct sched_myfifo_entity myfifo;
```

Step-3: define a new runqueue

- Recall that:
  - For NORMAL scheduling class, it has **`struct cfs_rq cfs`** in **`struct rq`**
  - For Real-time scheduling class, it has **`struct rt_rq rt`** in **`struct rq`**
- We define **`struct myfifo_rq`** in **`kernel/sched/sched.h`**

```
struct myfifo_rq {
    struct list_head queue;
    atomic_t nr_running;
};
```

- And add the following line to the definition of **`struct rq`** right after **`struct rt_rq rt;`**

```
struct cfs_rq cfs;
struct rt_rq rt;

struct myfifo_rq myfifo; /* myfifo rq */
```

Step-4: implement sched_myfifo.c

- Add the definition of myfifo sched_class in **kernel/sched/sched.h**
  - **Add extern const struct** sched_class myfifo_sched_class
  - **after** extern const struct sched_class idle_sched_class

```
extern const struct sched_class stop_sched_class;
extern const struct sched_class rt_sched_class;
extern const struct sched_class fair_sched_class;
extern const struct sched_class idle_sched_class;

extern const struct sched_class myfifo_sched_class; /* myfifo */
```

Step-4: implement sched_myfifo.c

- Create a new file `sched_myfifo.c` under the folder `kernel/sched`

```
#include "sched.h"
const struct sched_class myfifo_sched_class = {
    .next = &fair_sched_class,
    .enqueue_task = enqueue_task_myfifo,
    .dequeue_task = dequeue_task_myfifo,
    .yield_task = yield_task_myfifo,
    .check_preempt_curr = check_preempt_curr_myfifo,
    .pick_next_task = pick_next_task_myfifo,
    .put_prev_task = put_prev_task_myfifo,

    .set_curr_task = set_curr_task_myfifo,
    .task_tick = task_tick_myfifo,
    .get_rr_interval = get_rr_interval_myfifo,
    .prio_changed = prio_changed_myfifo,
    .switched_to = switched_to_myfifo,
    .select_task_rq = select_task_rq_myfifo,
};
```

Step-4: implement sched_myfifo.c

```
static void enqueue_task_myfifo(struct rq *rq,
    struct task_struct *p, int wakeup, bool head)
{
    struct sched_myfifo_entity *myfifo_se = &p->myfifo;
    list_add_tail(&myfifo_se->run_list, &rq->myfifo.queue);
    atomic_inc(&rq->myfifo.nr_running);
    printk(KERN_INFO"[SCHED_MYFIFO] ENQUEUE: Process-%d\n", p->pid);
}

static void dequeue_task_myfifo(struct rq *rq,
    struct task_struct *p, int sleep)
{
    struct sched_myfifo_entity *myfifo_se = &p->myfifo;
    list_del(&myfifo_se->run_list);
    atomic_dec(&rq->myfifo.nr_running);
    printk(KERN_INFO"[SCHED_MYFIFO] DEQUEUE: Process-%d\n", p->pid);
}
```

Step-4: implement sched_myfifo.c

```
static void yield_task_myfifo(struct rq *rq)
{
    struct sched_myfifo_entity *myfifo_se = &rq->curr->myfifo;
    struct myfifo_rq *myfifo_rq = &rq->myfifo;
    list_move_tail(&myfifo_se->run_list, &myfifo_rq->queue);
}

static void check_preempt_curr_myfifo(struct rq *rq,
    struct task_struct *p, int flags)
{
    if (rq->curr->policy == SCHED_FIFO || rq->curr->policy == SCHED_RR)
        return ;
    if (rq->curr->policy == SCHED_MYFIFO)
        return ;
    /* preempt normal tasks */
    resched_task(rq->curr);
}
```

Step-4: implement sched_myfifo.c

```
static struct task_struct *pick_next_task_myfifo(struct rq *rq)
{
    struct sched_myfifo_entity *myfifo_se = NULL;
    struct task_struct *p = NULL;
    struct myfifo_rq *myfifo_rq = &rq->myfifo;
    if (list_empty(&myfifo_rq->queue))
        return NULL;
    myfifo_se = list_entry(myfifo_rq->queue.next,
                           struct sched_myfifo_entity,
                           run_list);
    p = container_of(myfifo_se, struct task_struct, myfifo);
    return p;
}

static void put_prev_task_myfifo(struct rq *rq, struct task_struct *p)
{
    /* it is the place to update the current task's
     * runtime statistics */
}
```

Step-4: implement sched_myfifo.c

```
static void set_curr_task_myfifo(struct rq *rq) {
}

static void task_tick_myfifo(struct rq *rq,
                             struct task_struct *p, int queued) {
    if (p->policy != SCHED_MYFIFO) return ;
}

unsigned int get_rr_interval_myfifo(struct rq *rq,
                                    struct task_struct *p) {
    return 0;
}

static void prio_changed_myfifo(struct rq *rq, struct task_struct *p,
                                int oldprio, int running) {
}

static void switched_to_myfifo(struct rq *rq, struct task_struct *p,
                               int running) {
}
```

Step-5: merge into the kernel

- **First**
  - modify the **`Makefile`** under folder `kernel/sched`
  - Add `sched_myfifo.o` after `fair.o`
  - For example:

```
  GNU nano 2.2.6                    File: kernel/sched/Makefile

CFLAGS_core.o := $(PROFILING) -fno-omit-frame-pointer
endif

obj-y += core.o clock.o cputime.o idle_task.o fair.o sched_myfifo.o rt.o stop_task.o
obj-$(CONFIG_SMP) += cpupri.o
obj-$(CONFIG_SCHED_AUTOGROUP) += auto_group.o
obj-$(CONFIG_SCHEDSTATS) += stats.o
obj-$(CONFIG_SCHED_DEBUG) += debug.o
obj-$(CONFIG_CGROUP_CPUACCT) += cpuacct.o
```

Step-5: merge into the kernel

- **Second**
  - locate `__sched_fork(struct task_struct *p)` in `kernel/sched/core.c`
  - This function performs scheduler related setup for a newly forked process `p`
  - In this function, we need to initialize the necessary fields of `struct myfifo_entity myfifo` in `p`

```
static void __sched_fork(struct task_struct *p)
{
        p->on_rq                        = 0;

        p->se.on_rq                     = 0;
        p->se.exec_start                = 0;
        p->se.sum_exec_runtime          = 0;
        p->se.prev_sum_exec_runtime     = 0;
        p->se.nr_migrations             = 0;
        p->se.vruntime                  = 0;
        INIT_LIST_HEAD(&p->se.group_node);
        INIT_LIST_HEAD(&p->myfifo.run_list); /* init myfifo */
```

Step-5: merge into the kernel

- **Third**
  - locate `__sched_setscheduler(struct task_struct *p, int policy, struct sched_param *param, bool user)` in `kernel/sched/core.c`
  - In the beginning of the code, it is going to check the supported types of scheduling policies
  - Since `SCHED_MYFIFO` is newly created, we should avoid our type being reported as `-EINVAL`

```
if (policy != SCHED_FIFO && policy != SCHED_RR &&
        policy != SCHED_MYFIFO &&
        policy != SCHED_NORMAL && policy != SCHED_BATCH &&
        policy != SCHED_IDLE)
    return -EINVL;
```

Step-5: merge into the kernel

- **Fourth**,
  - locate `__setscheduler(struct rq *rq, struct task_struct *p, int policy, int prio)` in `kernel/sched/core.c`
  - Change the statement

`&fair_sched_class` to `&myfifo_sched_class;`

```
/* we are holding p->pi_lock already */
p->prio = rt_mutex_getprio(p);
if (rt_prio(p->prio))
        p->sched_class = &rt_sched_class;
else
        /*p->sched_class = &fair_sched_class;*/
        p->sched_class = &myfifo_sched_class;

set_load_weight(p);
```

Additional changes in core.c

- Inside kernel/sched/core.c, under the `void sched_fork(struct task_struct *p)` function
  - make the following additional changes:

```
if (!rt_prio(p->prio)  &&  p->policy != SCHED_MYFIFO )
        p->sched_class = &fair_sched_class;
```

Step-5: merge into the kernel

- **Fifth**
  - initialize the newly defined runqueue of **SCHED_MYFIFO**
  - Add the prototype in **kernel/sched/sched.h**
    - **extern void init_myfifo_rq(struct myfifo_rq *fifo_rq);**
  - Right after
    - **extern void init_rt_rq(struct rt_rq *rt_rq, struct rq *rq);**
  - Add the following function definition in **kernel/sched/sched_myfifo.c**

```
void init_myfifo_rq(struct myfifo_rq *fifo_rq)  {
        INIT_LIST_HEAD(&fifo_rq->queue);
        atomic_set(&fifo_rq->nr_running, 0);
}
```

Step-5: merge into the kernel

- **Sixth**,
  - locate `__init sched_init(void)` in `kernel/sched/core.c`
  - After the following line:
    - `init_rt_rq(&rq->rt, rq);`
  - Add
    - `init_myfifo_rq(&rq->myfifo);`

Step-5: merge into the kernel

- **Seventh**,
- change `.next` field of `rt_sched_class` in `kernel/sched/rt.c`

From

```
.next = &fair_sched_class,
```

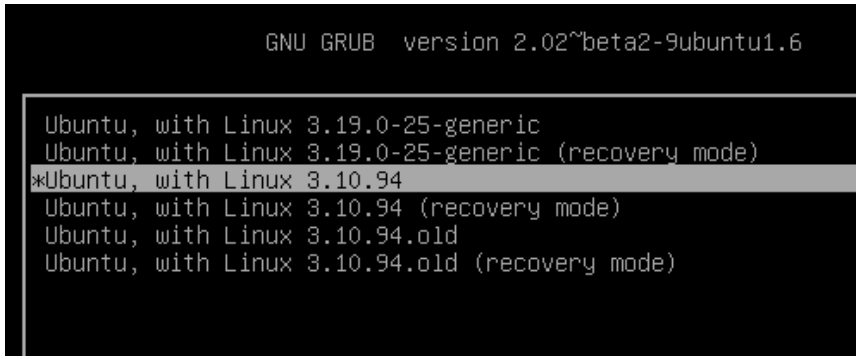
To

```
.next = &myfifo_sched_class,
```

```
const struct sched_class rt_sched_class = {
        /*.next                 = &fair_sched_class,*/
        .next                   = &myfifo_sched_class,
        .enqueue_task           = enqueue_task_rt,
        .dequeue_task           = dequeue_task_rt,
        .yield_task             = yield_task_rt,
```

Step-6: compile and install the kernel

- The steps should be pretty fast as we already re-compile most files in the previous exercise
- Go back to the top Linux directory
  - `sudo make -j4`
  - `sudo make modules`
  - `sudo make modules_install`
  - `sudo make install`
- Reboot and select the new kernel in GRUB
  - `sudo shutdown -r now`



Step 7 – Compile the MyFiFO test program

- Download the `myfifo_test.c` from the course web and place it to any folder
- It is a program to create two processes and assign them to the `MYFIFO` scheduler
- Compile the program by the following command:
  - `gcc -o myfifo_test myfifo_test.c`
- Make sure that you boot up with the re-compiled kernel

```
cspeter@ubuntu:~/comp4511_labs$ uname -r
3.10.94
cspeter@ubuntu:~/comp4511_labs$
```

Step 8 – Test the MyFIFO scheduler

- Expected result:

```
cspeter@ubuntu:~/comp4511_labs$ sudo ./myfifo_test 1000
Parent-1067 started
Child-1068 created
Child-1069 created
Child-1069 finished: Thu Jan 28 16:24:42 2016
Child-1068 finished: Thu Jan 28 16:24:42 2016
Parent-1067 exited
```

Step 9 – Check the Kernel log file

- Command:
  - `tail -15 /var/log/syslog`
- The expected result:

```
[  755.411348] [SCHED_MYFIFO] ENQUEUE: Process-1067
[  755.411542] [SCHED_MYFIFO] DEQUEUE: Process-1067
[  756.970162] [SCHED_MYFIFO] ENQUEUE: Process-1067
[  756.975684] [SCHED_MYFIFO] DEQUEUE: Process-1067
```

Appendix: FIFO with SMP support

Add SMP support to sched_myfifo.c (1/2)

```
static const struct sched_class myfifo_sched_class = {
  .next = &fair_sched_class,
  .enqueue_task = enqueue_task_myfifo,
  .dequeue_task = dequeue_task_myfifo,
  .yield_task = yield_task_myfifo,
  .check_preempt_curr = check_preempt_curr_myfifo,
  .pick_next_task = pick_next_task_myfifo,
  .put_prev_task = put_prev_task_myfifo,

#ifdef CONFIG_SMP
  .select_task_rq = select_task_rq_myfifo,
  .load_balance = load_balance_myfifo,
  .move_one_task = move_one_task_myfifo,
#endif

  .set_curr_task = set_curr_task_myfifo,
  .task_tick = task_tick_myfifo,
  .get_rr_interval = get_rr_interval_myfifo,
  .prio_changed = prio_changed_myfifo,
  .switched_to = switched_to_myfifo,
};
```

Add SMP support to sched_myfifo.c (2/2)

```
/* Implementation for the 3 extra functions for My FIFO scheduler */
#ifdef CONFIG_SMP
static int select_task_rq_myfifo(struct rq *rq,
                  struct task_struct *p,
                  int sd_flag, int flags) {
  return task_cpu(p);
}
static unsigned long load_balance_myfifo(struct rq *this_rq,
                      int this_cpu, struct rq *busiest,
                      unsigned long max_load_move,
                      struct sched_domain *sd,
                      enum cpu_idle_type idle,
                      int *all_pinned, int *this_best_prio){

                /* don't touch in our MYFIFO tasks */
    return 0;
}
static int move_one_task_myfifo(struct rq *this_rq, int this_cpu,
     struct rq *busiest, struct sched_domain *sd,
    enum cpu_idle_type idle){
       return 0;
}
#endif
```