# Assignment 2: Scheduling Policy Demonstration Program

This assignment aims to implement a program to apply different scheduling policies on created threads and observe their behaviors.

## Linux Scheduling Policy

The scheduling polices can be divided into four categories:

- Fair scheduing policies
  - `SCHED_NORMAL` (CFS, `SCHED_OTHER` in POSIX), `SCHED_BATCH`
- Real-Time scheduing policies
  - `SCHED_FIFO` , `SCHED_RR`
- The other two are idle scheduling policy ( `SCHED_IDLE` ) and deadline scheduling policy ( `SCHED_DEADLINE` )

The default scheduling policy is `SCHED_NORMAL` .

As you are only required to set either of `SCHED_NORMAL` or `SCHED_FIFO` policy to a thread in this assignment, and `SCHED_NORMAL` has been covered in the course, we will introduce the real-time scheduling policy `SCHED_FIFO` here.

### `SCHED_FIFO`

What exactly does `SCHED_FIFO` do?

- `SCHED_FIFO` is a simple real-time scheduling algorithm **without time slicing**.

- When a `SCHED_FIFO` thread becomes runnable, it will **always immediately preempt** any currently running thread with fair scheduling policy.

So, when will a `SCHED_FIFO` thread be scheduled?

1. Blocked by an I/O request (becomes `TASK_INTERRUPTIBLE` or `TASK_UNINTERRUPTIBLE` )
2. Preempted by other `SCHED_FIFO` thread with higher priority.
    - When the thread is in **ready state**.
3. Calls `sched_yield(2)` or `sleep(3)` to give up the CPU resource.

You can run `ps -eo state,uid,pid,ppid,rtprio,time,comm` to list processes with their real-time priorities:

```
$ ps -eo state,uid,pid,ppid,rtprio,time,comm
S   UID     PID    PPID RTPRIO     TIME COMMAND
I     0      13       2      - 00:00:00 rcu_tasks_rude_kthread
I     0      14       2      - 00:00:00 rcu_tasks_trace_kthread
S     0      15       2      - 00:00:00 ksoftirqd/0
I     0      16       2      1 00:00:03 rcu_preempt
S     0      17       2      1 00:00:00 rcub/0
S     0      18       2     99 00:00:00 migration/0
S     0      19       2     50 00:00:00 idle_inject/0
```

The `RTPRIO` represents "real-time policy priority", which ranges from 1 to 99. The process with the smallest value 1 has the lowest priority. On the other hand, the processes with "-" in their `RTPRIO` column means they are not real-time processes (or threads).

As you can see, the processes `rcu_preempt` , `rcub/0` , `migration/0` and `idle_inject/0` are all real-time processes with their priority values.

### Reference

- sched(7) - Linux manual page (https://man7.org/linux/man-pages/man7/sched.7.html)

## Requirements

In this assignment, you are required to implement a program called `sched_demo_<student_id>` , which lets a user run multiple threads with different scheduling policies and show the working status of each thread.

```
$ sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is starting
Thread 3 is starting
Thread 3 is starting
Thread 1 is starting
Thread 1 is starting
Thread 1 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
Thread 2 is starting
Thread 0 is starting
```

The meanings of command-line arguments are:

- `-n <num_threads>` : number of threads to run simultaneously
- `-t <time_wait>` : duration of "busy" period
- `-s <policies>` : scheduling policy for each thread, `SCHED_FIFO` or `SCHED_NORMAL` .
    - The example `NORMAL,FIFO,NROMAL,FIFO` shown above means to apply `SCHED_NORMAL` policy to the 1st and 3rd thread and `SCHED_FIFO` policy to the 2nd and 4nd thread.
- `-p <priorities>` : real-time thread priority for real-time threads
    - The example `-1,10,-1,30` shown above means to set the value `10` to the 2nd thread and value `30` to the 4nd thread.
    - You should specify the value `-1` for threads with `SCHED_NORMAL` policy.

The program can be divided into two sections: the main thread section and worker thread section.

## Main Thread

The main thread first needs to parse the program arguments, sets CPU affinity of all threads to **the same CPU**, and then creates `<num_threads>` worker threads specified by the option `-n` .

For each worker thread, attributes such as scheduling inheritance, scheduling policy and scheduling priority must be set. Next, the main thread will start all threads at once, and finally wait for all threads to complete.

```
int main() {
    /* 1. Parse program arguments */

    /* 2. Create <num_threads> worker threads */

    /* 3. Set CPU affinity */

    for (int i = 0; i < <num_threads>; i++) {
        /* 4. Set the attributes to each thread */
    }
    /* 5. Start all threads at once */

    /* 6. Wait for all threads to finish  */
}
```

> Hint1 💡: `getopt(3)` (https://man7.org/linux/man-pages/man3/getopt.3.html) can be used to parse command-line options.

> Hint2 💡: Some useful functions:
>
> - `sched_setaffinity(2)` (https://man7.org/linux/man-pages/man2/sched_setaffinity.2.html)/ `pthread_setaffinity_np(3)` (https://man7.org/linux/man-pages/man3/pthread_setaffinity_np.3.html): Set CPU affinity
> - `sched_setparam(2)` (https://man7.org/linux/man-pages/man2/sched_setparam.2.html)/ `pthread_attr_setschedparam(3)` (https://man7.org/linux/man-pages/man3/pthread_attr_setschedparam.3.html): Set scheduling parameters
> - `sched_setscheduler(2)` ()/ `pthread_attr_setschedpolicy(3)` (https://man7.org/linux/man-pages/man3/pthread_attr_getschedpolicy.3.html): Set scheduling policy

> Hint3 💡: Start all threads one by one, like, in step 4, will have high probability producing incorrect outputs. You may want to see `pthread_barrier_wait(3p)` (https://man7.org/linux/man-pages/man3/pthread_barrier_wait.3p.html) to synchronize threads.

### Worker Thread

Each newly-created worker thread must wait for other threads before executing its task. Then the thread runs the loop for 3 times.

In each loop, it shows a message indicating it's starting and performs the busy work for `<time_wait>` seconds specified by the `-t` option. Finally, it exits the function.

```
void *thread_func(void *arg)
{
    /* 1. Wait until all threads are ready */

    /* 2. Do the task */
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is starting\n", <id-of-the-current-thread>);
        /* Busy for <time_wait> seconds */
    }
    /* 3. Exit the function  */
}
```

Warning ⚠️: You can't use `sleep(3)` or `nanosleep(3)` functions to achieve busy waiting, which just makes the thread to enter sleeping state and put it into the ready queue to be scheduled after the specified time.

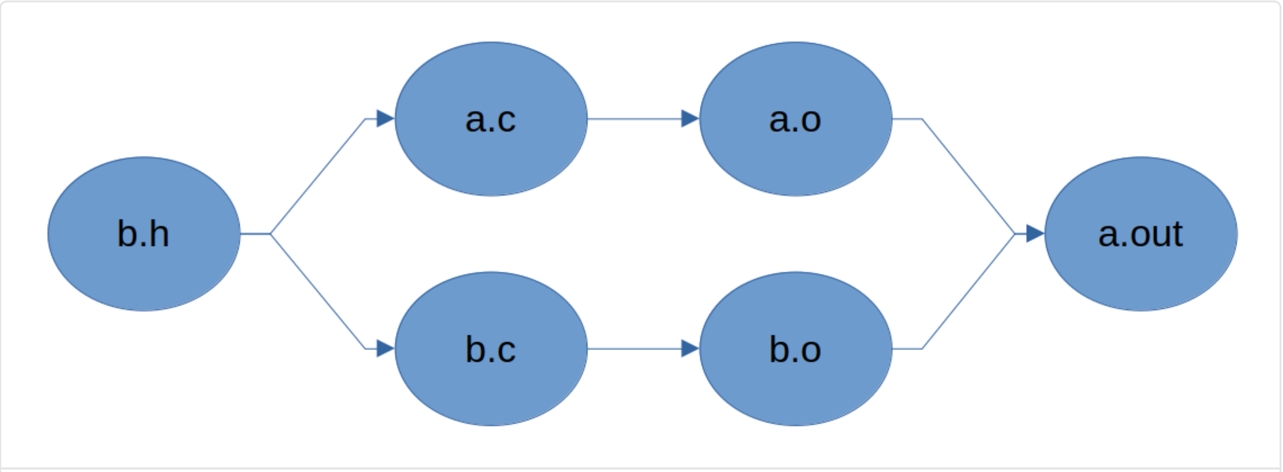
Hint 💡: The time it spends in the busy wait should exclude the time that the thread is being preempted. You could use the `time` command to time your process. It should be around $\text{Time given} \times \text{Process count} \times 3(\text{seconds})$

## Makefile Mini Tutorial

GNU Make is a really handy tool for compiling large programs with just a short command `make`. The part before the ':' is usually called the target, the part after the ':' is called the dependencies of the target, and the part indented after them is called the recipe. Below is an example Makefile:

```
# indicating that target "all" and "clean" are not files
.PHONY: all clean

# set some variables
CC= gcc
CFLAGS= -Wall -Wextra -Werror -O3 -Wpedantic
OUTPUT_OPTION= -MMD -MP -o $@

SOURCE= a.c b.c
OBJS= $(SOURCE:.c=.o)
DEPS= $(SOURCE:.c=.d)
TARGET= a.out

# first command of make
all: $(TARGET)

# import the dependencies of .h .c files from the compiler
-include $(DEPS)

# implicit targets
# %.o: %.c
#        $(CC) $^ -o $@ -c $(CFLAGS)

$(TARGET): $(OBJS)
    $(CC) $^ -o $@

clean:
    @rm -f $(TARGET) $(OBJS) $(DEPS)
```

```
    |- test/
        |- a.c
        |- b.c
        |- b.h
        |- Makefile
----after running command make----
    |- test/
        |- a.c
        |- b.c
        |- b.h
        |- a.d
        |- b.d
        |- a.o
        |- b.o
        |- a.out
        |- Makefile
----after running command make clean----
    |- test/
        |- a.c
        |- b.c
        |- b.h
        |- Makefile
```



Dependency graph of the Makefile

If the dependencies of the target was not changed and the target is older than those of the dependencies, the recipe will not be run again.

reference:

- Youtube reference ~1hr (https://youtu.be/WFLvcMiG38w?si=-A-LnayK4jwDstV8)
- GNU Make Manual (https://www.gnu.org/software/make/manual/make.html)

## Test

You can download the example program `sched_demo` and the test script `sched_test.sh` from E3 to test your program.

There are 3 public testcases by default. If your program passes all testcases, it will show the message "Success!" for each testcase.

```
$ sudo ./sched_test.sh ./sched_demo ./sched_demo_<student_id>
Running testcase 1: ./sched_demo -n 1 -t 0.5 -s NORMAL -p -1......
Result: Success!
Running testcase 2: ./sched_demo -n 2 -t 0.5 -s FIFO,FIFO -p 10,20......
Result: Success!
Running testcase 3: ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30......
Result: Success!
```

> Note : Remember to give the executable and script permission to execute with the command
>
> `chmod +x sched_demo`
>
> `chmod +x sched_test.sh`

However, if your program fails any testcases, the test script will exit immediately and print the message "Failed..." with the `diff` results between two programs.

```
$ sudo ./sched_test.sh ./sched_demo ./sched_demo_<student_id>
Running testcase 1: ./sched_demo -n 1 -t 0.5 -s NORMAL -p -1 ......
Result: Success!
Running testcase 2: ./sched_demo -n 2 -t 0.5 -s FIFO,FIFO -p 10,20 ......
0a1,3
> Thread 1 is running
> Thread 1 is running
> Thread 1 is running
Result: Failed...
```

By the way, you can add your own testcases in the test script:

```
# You can add your own testcases here
testcases=("-n 1 -t 0.5 -s NORMAL -p -1"
           "-n 2 -t 0.5 -s FIFO,FIFO -p 10,20"
           "-n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30")
```

## Submission

Please submit a **zip** file to E3, which contains the program source and the report.

For the program source part (50%):

- The program must be implemented using C or C++.
- Must provide a Makefile that includes at least
  - An `all:` target for compilation of the source program to `sched_demo_<student_id>`
  - A `clean:` target to clean up the executable and the intermediate files
- Make sure your program passes all 3 public testcases.
- Make sure your code can be compiled on **Ubuntu 24.04 x86_64**.
- There will be some hidden test cases but are just to confirm the correctness of your program

> Note : If you don't have an x86 machine, please ask us for help.

> You do not need to worry about error handling in your program.

> It should be clear that **PLAGIARISM** will not be tolerated.

> If your Makefile fails to compile or clean up the directory properly, you will not receive the points for this part.

For the report part, you must answer the following questions (50%):

1. Describe how you implemented the program in detail. (10%)
2. Describe the results of `sudo ./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30` and what causes that. (10%)
3. Describe the results of `sudo ./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30`, and what causes that. (10%)
4. Describe how did you implement n-second-busy-waiting? (10%)
5. What does the `kernel.sched_rt_runtime_us` effect? If this setting is changed, what will happen?(10%)

> **Warning ⚠️: Setting `kernel.sched_rt_runtime_us` to aggresively may affect system processes and system stability!**

The name of the zip file should be `<student_id>.zip`, and the structure of the file should be as the following:

```
<stduent_id>.zip
    |- <student_id>/
        |- report_<student_id>.pdf
        |- Makefile
        |- sched_demo_<student_id>.c (or sched_demo_<stduent_id>.cpp)
        |- other source files ...
```

> The deadline is on **11/25 23:59**.