PA#2 : My Thread Scheduler

SCE213 (Operating Systems)
Spring 2022

TA: Seungsu Baek (백승수)

Goal

- Implement your own user level thread library and scheduler.
  - We’re going to implement
    - User level thread library
    - Scheduler

Background: Thread and Scheduler

- User level thread
  - Implemented by users and the kernel is not aware of existence of these threads.
  - Easier and faster than kernel-level threads.
  - Many to one model.
  - Do not fully utilize hardware resource..
- Scheduler
  - Scheduler finds the process to run depend on scheduling policy.
  - First In First Out, Round Robin, Priority, Shortest Job First, etc...
- Signal
  - Signal is an event generated by the Linux system in response to some condition.
  - When the process receives the signal, predefined handler is running.

Problem specification

- Make thread block and scheduling them.
- Implement some function in ***uthread.c***
- The functions that have to be implemented by you have anotation(**TODO**).

ucontext.h

- The <ucontext.h> header defines the ucontext_t type as a structure that includes at least the following members.

```
struct ucontext_t {
    ucontext_t *uc_link     pointer to the context that will be resumed
                            when this context returns
    sigset_t    uc_sigmask  the set of signals that are blocked when this
                            context is active
    stack_t     uc_stack    the stack used by this context
    mcontext_t  uc_mcontext a machine-specific representation of the saved
                            context
}
```

- Use getcontext(), makecontext(), swapcontext(), etc...
- Handle signal using some function in <signal.h> header.

Thread workflow

- When threads terminated, threads have to inform their state. Because main thread wait until the thread terminated. It is not visible to programmer.
  So you have to create the context that will be resumed after the terminated thread.

- See __exit(), __initialize_exit_context().

Testcases and Outputs

- Testcases
  - Input of main program
  - First line is scheduling policy
  - CREATE makes to call thread_create. 0 4 0 means tid, lifetime, priority, respectively.
  - JOIN makes to call thread_join. 0 means the target thread's tid is 0.
  - When you implement thread creation, use this information.

```
FIFO
CREATE 0 4 0
CREATE 1 3 0
JOIN 0
JOIN 1
```

Testcases and Outputs

- Outputs
  - SWAP -1 -> 0 means that scheduler do context swap from -1 to 0.
  - JOIN 0 means that main thread join successfully thread 0.
  - Do not print when context switching from main to main.

```
SWAP -1 -> 0
SWAP 0 -> 1
SWAP 1 -> -1
JOIN 0
JOIN 1
```

Check your program in local

- Use make
  - `make test` : All testcases are executed and show the results.
  - `make test_*` : Particular testcase are executed and show the results

```
gnup@DESKTOP-GOCUB3C:~/sce213-project2$ make test
Test Fisrt In Fisrt Out SCHEDULER
        Test seems correct

Test Round Robin SCHEDULER
        Test seems correct

Test Priority SCHEDULER
        Test seems correct

Test Shortest Job First SCHEDULER
        Test seems correct

gnup@DESKTOP-GOCUB3C:~/sce213-project2$
```

PA#2: Deliverables

- Submission by May ?, 11:59 PM
  - Submit only ***uthread.c*** for the code
- Pasubmit
  - Please see “Handout” button in the PA description for more information
  - Start project by cloning PA#2 repository

    git clone https://github.com/csl-ajou/sce213-project2.git