OS HW2

資科工所 312551105 羅羽軒

- 文章位址: https://hackmd.io/@lohsuan/rkdDdjdNp
- 題目敘述: Assignment 2: Scheduling Policy Demonstration Program

Source Part

- Github: Source Code

`sche_test` outcome

```
sudo ./sched_test.sh ./sched_demo ./sched_demo_312551105
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!
```

Report Part

1. Describe how you implemented the program in detail.

Parse program arguments

使用 `getopt` parse 傳入的 argv， `getopt` 會根據 `"n:t:s:p:"` optstring 制定的規則做 parse。

```
while ((opt = getopt(argc, argv, "n:t:s:p:")) != -1) {
    switch (opt) {
    case 'n':
        if (atoi(optarg) != num_threads) {
            printf("Please put -n argument as the first passing value\n");
            printf("Usage: sudo ./sche -n 1 -t 0.5 -s NORMAL -p -1\n");
            exit(EXIT_FAILURE);
        }
        break;
    case 't':
        time_wait = atof(optarg);
        break;
    case 's':
        char *s_string = optarg;
        char *s_token = strtok(s_string, ",");
        for (int i = 0; i < num_threads; i++) {
            policies[i] = s_token;
            s_token = strtok(NULL, ",");
        }
        break;
    case 'p':
        char *p_string = optarg;
        char *p_token = strtok(p_string, ",");
        for (int i = 0; i < num_threads; i++) {
            priorities[i] = atoi(p_token);
            p_token = strtok(NULL, ",");
        }
        break;
    default:
        fprintf(stderr, "Input usage error\n");
        exit(EXIT_FAILURE);
    }
}
```

Create <num_threads> worker thread_info

下面的程式宣告 cread thread 會用到的 attribute，每個變數的作用以註解展示。

```
pthread_attr_t attr[num_threads];       // thread attributes
struct sched_param param[num_threads];  // thread parameters
pthread_t thread[num_threads];          // thread identifiers
thread_info_t thread_info[num_threads]; // thread information
```

Set CPU affinity

- 將所有 thread 設定跑在相同 CPU (ex: cpu_id=0)，細節在註解中。
- (若無加上 `#define _GNU_SOURCE` 在設定affinity 時會失敗)

```
#define _GNU_SOURCE

int cpu_id = 0;
cpu_set_t cpuset;
CPU_ZERO(&cpuset);       // clear cpuset
CPU_SET(cpu_id, &cpuset); // set CPU 0 on cpuset
sched_setaffinity(0, sizeof(cpuset), &cpuset);
```

Set the attributes to each thread

- 根據上面 parse 的 parameter 設定 `thread_info` 中各個 field 的值。
- 並根據 policy 是 FIFO 或 NORMAL 決定 `pthread_create` 需不需要設定並傳入前面宣告之 attr。
- 設定 attr 之步驟細節在註解中說明。
- `pthread_create(&thread[i], &attr[i], thread_func, &thread_info[i])`：
  - 3rd parameter: `thread_func` 是讓此 thread 執行的程式
  - 4th parameter: `&thread_info[i]` 是傳入 thread_func 的參數

```
for (int i = 0; i < num_threads; i++) {
    thread_info[i].thread_id = i;
    thread_info[i].time_wait = time_wait;

    if (strcmp(policies[i], "FIFO") == 0) {
        thread_info[i].sched_policy = SCHE_FIFO;

        // initialize thread attributes
        pthread_attr_init(&attr[i]);
        // set the scheduling inheritance to explicit
        pthread_attr_setinheritsched(&attr[i], PTHREAD_EXPLICIT_SCHED);

        // set the scheduling policy - FIFO
        if (pthread_attr_setschedpolicy(&attr[i], SCHE_FIFO) != 0) {
            printf("Error: pthread_attr_setschedpolicy\n");
            exit(EXIT_FAILURE);
        }

        // set the priority
        param[i].sched_priority = priorities[i]; // set priority

        // set the scheduling paramters
        if (pthread_attr_setschedparam(&attr[i], &param[i]) != 0) {
            printf("Error: pthread_attr_setschedparam\n");
            exit(EXIT_FAILURE);
        }

        // create the thread
        if (pthread_create(&thread[i], &attr[i], thread_func,
                           &thread_info[i]) != 0) {
            printf("Error: FIFO pthread_create\n");
            exit(EXIT_FAILURE);
        }

    } else if (strcmp(policies[i], "NORMAL") == 0) {
        thread_info[i].sched_policy = SCHE_NORMAL;

        pthread_create(&thread[i], NULL, thread_func, &thread_info[i]);

    } else {
        printf("Unexpected input \n");
    }
}
```

Start all thread_info at once

- 利用 `pthread_barrier_init` 設定需要等待多少 thread，數量為num_threads + 1 (num of thread and main thread)
- 將 `pthread_barrier_init` 至於 `thread_func` 最開始，等到所有thread ready 才繼續執行 task

```
pthread_barrier_t barrier;
// initialize barrier threads + main thread = num_threads + 1
pthread_barrier_init(&barrier, NULL, num_threads + 1);
pthread_barrier_wait(&barrier);
pthread_barrier_destroy(&barrier);
```

> - `pthread_barrier_^()` 把先後到達的 threads 擋在同一欄秤前，直到所有 thread 到齊才撤下欄秤同時放行。
> - Ref: https://www.cnblogs.com/liyulong1982/p/5480678.html

Worker thread: Do the task

- `clock_gettime` 可以取得 wall-clock time 或程式的 CPU time，其所傳回的時間是 `timespec` struct
- 透過每次更新 `end_time` 並計算與 `start_time` 的差值取得 thread 現在已使用多長時間
- 當 thread 執行的時間超過了 `start_time - end_time` 時就會 `break`，也就是等待了 **busy time** 的時間

```
for (int i = 0; i < 3; i++) {
    printf("Thread %d is running\n", (int)thread_info->thread_id);

    /* Busy for <time_wait> seconds */
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_THREAD_CPUTIME_ID, &start_time);
    int start_time_msec = start_time.tv_sec * MILLI_3 + start_time.tv_nsec / MICRO_6;

    while (1) {
        clock_gettime(CLOCK_THREAD_CPUTIME_ID, &end_time);
        int end_time_msec = end_time.tv_sec * MILLI_3 + end_time.tv_nsec / MICRO_6;
        if ((end_time_msec - start_time_msec) >= busy_time_msec) {
            break;
        }
    }

    sched_yield(); // yield the CPU
}
```

Wait for all thread_info to finish

使用 `pthread_join` 等待所有 thread ter,inate

```
for (int i = 0; i < num_threads; i++) {
    pthread_join(thread[i], NULL);
}
```

Ensure real-time threads can fully utilize the CPU

- set `sysctl -w kernel.sched_rt_runtime_us=1000000` in terminal
- this ensure the **real-time threads can fully utilize the CPU resource without being preempted by fair-time threads**

Compile, Run, and Test

```
# compile
$ gcc sched_demo_312551105.c -lpthread  -o sched_demo_312551105

# run (run with "sudo" to have previlege to create_thread)
$ sudo ./sched_demo_312551105 -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
$ sudo ./sched_demo_312551105 -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30

# test with TA code
$ sudo ./sched_test.sh ./sched_demo ./sched_demo_312551105
```

2. Describe the results of `./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30` and what causes that

- FIFO (thread 1, 2) 因為是 real time policy 比 NORMAL (thread 0) 有更高的優先權，且 thread 2 比 thread 1 的 priority 更大
- 因此先執行 thread 2 接著 thread 1 最後是 NORMAL 的 thread 0

```
▶ Scheduling Policy: git:(main) make run
 sudo ./sched_demo_312551105 -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
 Thread 2 is running
 Thread 2 is running
 Thread 2 is running
 Thread 1 is running
 Thread 1 is running
 Thread 1 is running
 Thread 0 is running
 Thread 0 is running
 Thread 0 is running
```

3. Describe the results of `./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30` , and what causes that

- FIFO (thread 1, 3) 因為是 real time policy 比 NORMAL (thread 0, 2) 有更高的優先權，且 thread 3 比 thread 1 的 priority 更大
- 因此先執行 thread 3 接著換 thread 1，最後 thread 0, 2 因沒有 priority 優先所以交替進行。

```
sudo ./sched_demo_312551105 -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30
Thread 3 is running
Thread 3 is running
Thread 3 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 1 is running
Thread 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
Thread 0 is running
```

4. Describe how did you implement n-second-busy-waiting?

- `clock_gettime` 可以取得 wall-clock time 或程式的 CPU time，其所傳回的時間是 `timespec` struct
- 透過每次更新 `end_time` 並計算與 `start_time` 的差值取得 thread 現在已使用多長時間
- 當 thread 執行的時間超過了 `start_time - end_time` 時就會 `break`，也就是等待了 **busy time** 的時間

```
for (int i = 0; i < 3; i++) {
    printf("Thread %d is running\n", (int)thread_info->thread_id);

    /* Busy for <time_wait> seconds */
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_THREAD_CPUTIME_ID, &start_time);
    int start_time_msec = start_time.tv_sec * MILLI_3 + start_time.tv_nsec / MICRO_6;

    while (1) {
        clock_gettime(CLOCK_THREAD_CPUTIME_ID, &end_time);
        int end_time_msec = end_time.tv_sec * MILLI_3 + end_time.tv_nsec / MICRO_6;
        if ((end_time_msec - start_time_msec) >= busy_time_msec) {
            break;
        }
    }

    sched_yield(); // yield the CPU
}
```

> `CLOCK_THREAD_CPUTIME_ID` : 程式單一 thread 所耗費的時間。
>
> ```
> struct timespec {
>   time_t tv_sec;  /* seconds */
>   long   tv_nsec; /* nanoseconds */
> };
> ```