Assignment 2: Scheduling Policy Demonstration Program

311552067 劉承熙

1. Describe how you implemented the program in detail

   - Using library to parse the argument

```
/* 1. Parse program arguments */

    int opt;
    static struct option long_options[] = {
        {"num_threads", 1, 0, 'n'},
        {"time_wait", 1, 0, 't'},
        {"policies", 1, 0, 's'},
        {"priorities", 1, 0, 'p'},
        {"help", 0, 0, '?'},
        {0 ,0, 0, 0}
    };
    int num_threads = 0;
    string policies = "";
    string priorities = "";
    while ((opt = getopt_long(argc, argv, "n:t:s:p:h", long_options,NULL)) != EOF) {

        switch (opt) {
        case 'n':
            num_threads = atoi(optarg);
            break;
        case 't':
            time_wait = atof(optarg);
            break;
        case 's':
            policies = optarg;
            break;
        case 'p':
            priorities = optarg;
            break;
        case 'h':
            usage(argv[0]);
        default:
            usage(argv[0]);
            return 1;
        }
    }

void usage(const char* progname) {
  cout << "Usage: "<< progname <<" [options]\n";
  cout << "Program Options:\n";
  cout << "  -n <num_threads>: number of threads to run simultaneously\n";
  cout << "  -t <time_wait>: duration of \"busy\" period\n";
  cout << "  -s <policies>: scheduling policy for each thread, SCHED_FIFO or
SCHED_NORMAL\n";
  cout << "  -p <priorities>: real-time thread priority for real-time threads\n";
}
```

- Define thread and thread attribute, then set affinity, schedule_policy and schedule_priority for any thread.
  pthread_attr_init — for initial thread attribute
  pthread_attr_setaffinity_np — set cpuset (all thread using same core) to thread attribute
  pthread_attr_setschedpolicy — set schedule_policy (SCHED_FIFO or SCHED_OTHER) to thread attribute
  pthread_attr_setschedparam — set sched_priority to thread attribute
  pthread_attr_setinheritsched — set PTHREAD_EXPLICIT_SCHED to thread attribute, ensure thread will create by our setting

```
    pthread_t threads[num_threads];
    int thread_id[num_threads];
    pthread_attr_t thread_attr[num_threads];

    /* 3. Set CPU affinity */
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(3, &cpuset);  // Set affinity to CPU 0

    for (int i = 0; i < num_threads; i++) {
        /* 4. Set the attributes to each thread */
        pthread_attr_init(&thread_attr[i]);

        thread_id[i] = i;

        size_t pos = policies.find(",");
        string policy_str = policies.substr(0, pos);
        int sched_policy = (policy_str == "FIFO") ? SCHED_FIFO : SCHED_OTHER;

        policies.erase(0, pos + 1);

        pos = priorities.find(",");
        string priority_str = priorities.substr(0, pos);
        int sched_priority = stoi(priority_str);

        priorities.erase(0, pos + 1);

        if (pthread_attr_setaffinity_np(&thread_attr[i], sizeof(cpu_set_t),
&cpuset) != 0) {
            perror("Error setting CPU affinity");
            pthread_exit(NULL);
        }

        if (pthread_attr_setschedpolicy(&thread_attr[i], sched_policy) != 0) {
            perror("Error setting scheduling policy");
            pthread_exit(NULL);
        }

        if(sched_priority != -1){
            struct sched_param param;
            param.sched_priority = sched_priority;
            if (pthread_attr_setschedparam(&thread_attr[i], &param) != 0) {
                perror("Error setting scheduling parameters");
                pthread_exit(NULL);
            }
        }
        pthread_attr_setinheritsched(&thread_attr[i],PTHREAD_EXPLICIT_SCHED);

    }
```

- pthread_barrier_init — set barrier for waitting threas, then start all threads at once

  Create thread with the previously configured thread attributes

```
    /* 2. Create <num_threads> worker threads */
    /* 5. Start all threads at once */

    pthread_barrier_init(&barrier, NULL, num_threads);
    for (int i = 0; i < num_threads; i++) {
        if (pthread_create(&threads[i], &thread_attr[i], thread_func,
&thread_id[i]) != 0) {
            perror("Error creating thread");
            exit(EXIT_FAILURE);
        }
    }
```

- pthread_barrier_wait — set barrier for making start all threads at once

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

    /* 2. Do the task */
    for (int i = 0; i < 3; i++) {
        printf("Thread %d is running\n", *(int*)(arg));
        /* Busy for <time_wait> seconds */
        busy_wait(time_wait);
    }

    /* 3. Exit the function  */
    pthread_exit(NULL);
}
```

- Waitting all thread finishing execution

  pthread_attr_destroy — for delete all thread attribute pthread_barrier_destroy — for delete barrier

```
    /* 6. Wait for all threads to finish  */
    for (int i = 0; i < num_threads; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("Error joining thread");
            exit(EXIT_FAILURE);
        }
    }
    for (int i = 0; i < num_threads; i++) {
        pthread_attr_destroy(&thread_attr[i]);
    }
    pthread_barrier_destroy(&barrier);
```

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

Threat 2 and Thread 1 are scheduled using the SCHED_FIFO (Real-Time scheduling policies). Therefore, they are both prioritized and processed based on their assigned priorities. Since Thread 2 has a higher priority number, it takes precedence in utilizing the core, and only after Thread 2 completes its execution, Thread 1 is scheduled

Thread 0 is scheduled using SCHED_NORMAL (Fair scheduling policies). Since there are no other SCHED_NORMAL threads, it completes its execution directly

```
HW2  sudo ./sched_demo -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

Threat 3 and Thread 1 are scheduled using the SCHED_FIFO (Real-Time scheduling policies). Therefore, they are both prioritized and processed based on their assigned priorities. Since Thread 3 has a higher priority number, it takes precedence in utilizing the core, and only after Thread 3 completes its execution, Thread 1 is scheduled

Thread 0 and Thread 2 use the SCHED_NORMAL (Fair scheduling policies), so they will use the CPU after SCHED_FIFO threads. They will take turns utilizing the CPU

```
HW2  sudo ./sched_demo -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 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
Thread 2 is running
Thread 0 is running
```

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

Using the <ctime> library's clock() function, calculate whether the required seconds have elapsed within a while loop.

```
void busy_wait(double seconds) {
    double start_time = (double)clock() / CLOCKS_PER_SEC;
    double current_time;

    do {
        current_time = (double)clock() / CLOCKS_PER_SEC;
    } while ((current_time - start_time) < seconds);
}
```