Operating System Assignment 2 Report

Student: **Chia-Hsuan Lin** (312551014)

Date: Nov. 5, 2023

I. Describe the implementation of the program in detail.

Main Thread

1. Parse program arguments.

   The main thread first needs to parse the program arguments. I used `NUM_THREADS`, `TIME_WAIT`, `policies`, and `priorities` variables to store the program arguments parsed by the `getopt` function.

```c
int policies[MAX_THREADS] = {0};
int priorities[MAX_THREADS] = {0};

// Parse program arguments
int option, index;
char *token;
while ((option = getopt(argc, argv, "n:t:s:p:")) != -1) {
  switch (option) {
    case 'n':
      NUM_THREADS = strtol(optarg, NULL, 10);
      if (NUM_THREADS > MAX_THREADS) {
        printf("%s: maximum number of threads = 32\n", argv[0]);
        exit(1);
      }
      break;
    case 't':
      TIME_WAIT = strtod(optarg, NULL);
      break;
    case 's':
      index = 0;
      token = strtok(optarg, ",");
      while (token != NULL && index < MAX_THREADS) {
        policies[index] = !strcmp(token, "FIFO") ? SCHED_FIFO :
SCHED_NORMAL;
        token = strtok(NULL, ",");
        index++;
```

```
      }
      break;
    case 'p':
      index = 0;
      token = strtok(optarg, ",");
      while (token != NULL && index <= MAX_THREADS) {
        priorities[index] = strtol(token, NULL, 10);
        token = strtok(NULL, ",");
        index++;
      }
      break;
  }
}
```

2. Set CPU affinity for the main thread.

   I've configured the program to exclusively run on CPU core 0 by utilizing the `sched_setaffinity` function from the Linux scheduler API. This involves setting up a `cpu_set_t` structure and adding CPU core 0 to it.

```
cpu_set_t cpu_set;
CPU_ZERO(&cpu_set);   // Clear the set
CPU_SET(0, &cpu_set); // Add core 0 to the set
sched_setaffinity(getpid(), sizeof(cpu_set), &cpu_set);
```

3. Create worker threads, set their attributes, and run it.

   To define the scheduling requirements for worker threads, the programmer must utilize several functions. Specifically, I've used `pthread_attr_setaffinity_np` to establish CPU affinity, `pthread_attr_setschedpolicy` to set the scheduling policy, and `pthread_attr_setschedparam` to determine the scheduling priority. However, for the scheduling policy to take effect, it is imperative to set the inherit-scheduler attribute to `PTHREAD_EXPLICIT_SCHED` immediately after calling the `pthread_attr_setschedpolicy` function.

```
// Declare worker threads and related resources
pthread_t threads[NUM_THREADS];
pthread_attr_t thread_attr[NUM_THREADS];
thread_info_t thread_info[NUM_THREADS];
struct sched_param thread_sched_param[NUM_THREADS];
pthread_barrier_init(&barrier, NULL, NUM_THREADS);
```

```
// Set attributes and run threads
for (int i = 0; i < NUM_THREADS; i++) {
  thread_info[i].id = i;
  pthread_attr_init(&thread_attr[i]);
  pthread_attr_setaffinity_np(&thread_attr[i], sizeof(cpu_set),
&cpu_set);
  pthread_attr_setschedpolicy(&thread_attr[i], policies[i]);
  pthread_attr_setinheritsched(&thread_attr[i],
PTHREAD_EXPLICIT_SCHED);
  thread_sched_param[i].sched_priority = priorities[i];
  pthread_attr_setschedparam(&thread_attr[i], &thread_sched_param[i]);
  pthread_create(&threads[i], &thread_attr[i], thread_func,
&thread_info[i]);
}
```

4. Wait for all threads to finish.

```
for (int i = 0; i < NUM_THREADS; i++)
  pthread_join(threads[i], NULL);
```

Worker Thread

To ensure the correctness of the program, each newly created worker thread must wait for the other threads before executing its task. I've achieved this by using a pthread barrier which blocks the thread until all threads reach this point. After that, the thread runs the loop three times.

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

  // Do some heavy tasks
  thread_info_t *thread_info = (thread_info_t *)arg;
  for (int i = 0; i < 3; i++) {
    printf("Thread %d is running\n", thread_info->id);
    // Busy for <TIME_WAIT> seconds
    time_t start, current = 0;
    time(&start);
    do {
      time(&current);
    } while (difftime(current, start) < TIME_WAIT);
```

```
    }

    // Exit the function
    pthread_exit(NULL);
}
```

II. Describe and explain the result of the following case.

Command

```
./sched_demo -n 3 -t 1.0 -s NORMAL,FIFO,FIFO -p -1,10,30
```

Result

```
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
```

Thread	Policy	Priority
0	NORMAL	-1
1	FIFO	10
2	FIFO	30

Threads with the FIFO policy are scheduled in the order they arrive, and their execution is based on their priority. Higher priority threads are scheduled before lower priority threads. Threads with the NORMAL policy are typically scheduled in a way that provides fair access to the CPU, and they do not have explicit priorities. So, based on this scenario, Thread 2 will be executed first, followed by Thread 1, and then Thread 0.

III. Describe and explain the result of the following case.

Command

```
./sched_demo -n 4 -t 0.5 -s NORMAL,FIFO,NORMAL,FIFO -p -1,10,-1,30
```

Result

```
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
```

Thread	Policy	Priority
0	NORMAL	-1
1	FIFO	10
2	NORMAL	-1
3	FIFO	30

In this scenario, threads with the FIFO policy will be executed sequentially, one after the other, like the previous case. However, threads with the NORMAL policy will try to evenly distribute the CPU time by switching between the running threads.

IV. Describe the implementation of the N-second-busy-waiting.

To introduce a delay in the program's execution without using the `sleep` function, the programmer can achieve this by first capturing the start time. Then, they can continuously update the current time while also checking if the difference between the current time and the start time is greater than the desired waiting time within a `while` loop.

```c
time_t start, current = 0;
time(&start);
do {
  time(&current);
} while (difftime(current, start) < TIME_WAIT);
```