OS-Project4

Task1: 五种 CPU 调度算法（含时间计算bonus）

1、FCFS

（1）add （）函数：正常插入即可

（2）schedule（）函数：由于插入时每次新加入的任务在链表的头部，因此首先进行链表反转，再依次调用 run 函数进行执行，在终端中打印结果。

（3）时间计算：

观察可以发现，FCFS、SJF、Priority 三种调度算法，只需知道各个任务的执行顺序，就可以根据以下公式计算时间：

$$
\begin{aligned}
turn &= \sum (time * (taskCount - i)) \\
wait &= \sum time * (taskCount - i - 1)) \\
response &= \sum (time * (taskCount - i - 1))
\end{aligned}
$$

其中 i 表示该任务的执行顺序（0-base），taskCount 表示总的任务个数。

（4）实验结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ make
fcfs
gcc -Wall -c schedule_fcfs.c
gcc -Wall -o fcfs driver.o schedule_fcfs.o list.o CPU.o
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ ./fcf
s schedule.txt
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T8] [10] [25] for 25 units.
Average Turnaround Time: 94.375000
Average Waiting Time: 73.125000
Average Response Time: 73.125000
```

2、SJF

（1）add（）函数：在插入时维护链表的顺序。每新读入一个任务节点，就根据节点任务的持续时间，在恰当的位置插入。

（2）schedule（）函数：由于插入的时候已经保证了任务执行顺序，因此直接执行即可。

（3）时间计算：同上。

（4）实验结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ ./sjf
 schedule.txt
Running task = [T6] [1] [10] for 10 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T8] [10] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Average turnaround time: 82.500000
Average waiting time : 61.250000
Average response time : 61.250000
```

3、Priority

（1）add（）函数：在插入时维护链表的顺序。每新读入一个任务节点，就根据节点任务的优先级，在恰当的位置插入。

（2）schedule（）函数：由于插入的时候已经保证了任务执行顺序，因此直接执行即可。

（3）时间计算：同上。

（4）实验结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ make
priority
gcc -Wall -c schedule_priority.c
gcc -Wall -o priority driver.o schedule_priority.o list.o CPU.o
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ ./pri
ority schedule.txt
Running task = [T8] [10] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T6] [1] [10] for 10 units.
Average turnaround time: 96.250000
Average waiting time : 75.000000
Average response time : 75.000000
```

4、RR

（1）add（）函数：直接插入即可。

（2）schedule（）函数：由于插入时每次新加入的任务在链表的头部，因此首先进行链表反转，再根据 rr 调度算法，判断剩余时间与时间片大小的关系。当一轮循环后没有新的任务执行时可以判断执行结束。

（3）时间计算：新增全局变量 time 用于标识当前时间，当某个任务首次执行时标记时间，参与 response计算、完成执行时参与 turn计算。wait 与 turn 相差每个任务的 burst 时间。

（4）实验结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ ./rr
schedule.txt
Running task = [T1] [4] [20] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T8] [10] [25] for 10 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T4] [5] [15] for 5 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T8] [10] [25] for 10 units.
Running task = [T2] [3] [25] for 5 units.
Running task = [T3] [3] [25] for 5 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T8] [10] [25] for 5 units.
Average Turnaround Time: 128.750000
Average Waiting Time: 107.500000
Average Response Time: 35.000000
```

5、Priority_rr

（1）add（）函数：按照优先级按顺序插入。

（2）schedule（）函数：直接在上个算法中进行拓展，增加一层外层循环嵌套，用于保证优先级最高的一（批）任务在完成 rr 后才会轮到其余任务。

（3）时间计算：同上。

（4）实验结果：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch5/project/posix$ ./pri
ority_rr schedule.txt
Running task = [T8] [10] [25] for 10 units.
Running task = [T8] [10] [25] for 10 units.
Running task = [T8] [10] [25] for 5 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T4] [5] [15] for 5 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T2] [3] [25] for 5 units.
Running task = [T3] [3] [25] for 5 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T6] [1] [10] for 10 units.
Average Turnaround Time: 105.000000
Average Waiting Time: 83.750000
Average Response Time: 68.750000
```

Task2: 原子运算来处理竞态

_sync_fetch_add（）函数可以用于原子处理变量。

首先添加一个全局变量。

之后只需在 add（）函数中，添加一个原子锁，通过全局变量，对全局变量进行原子操作，即可保证再多线程并发执行过程中，任务会被正确读取并插入到链表的相应位置。