Schedule lab 报告

姓名：卢虹宇
学号：203202269

方案一：Round Robin

基本思路

每个任务被分配一个固定的时间片，当时间片用完时（Ktimer信号出现），任务被放回队列的末尾，等待下一个轮转，确保所有任务都能公平地获得CPU时间。

得分

Test Case	Amplification	Finish Rate Hi Prio	Finish Rate Lo Prio	Finish Rate	Ave TL Rate	Elapsed Time	Cal Needed Time	Score
1	0.137213	1	1	1	1	1000036	1000036	1
2	0.185436	1	1	1	1	994316	994316	1
3	0.211537	1	1	1	1	988009	1010066	1
4	0.19527	1	1	1	1	990066	990066	1
5	0.184725	0.989796	1	0.992901	1.00259	1000821	1000821	0.990336
6	0.330656	0.92	0.947368	0.926724	1.0656	1053829	1094254	0.86967
7	0.229915	1	1	1	1	985488	985488	1
8	0.313372	0.962963	1	0.971888	1.02471	1060435	1060923	0.948451
9	0.348553	0.9375	0.980769	0.95122	1.0435	1001267	1001267	0.911566
10	0.391989	0.901961	0.958333	0.918164	1.09498	1056373	1066419	0.838524
11	0.225193	0.976744	0.982143	0.978678	1.00401	992708	992708	0.974767
12	0.160597	1	1	1	1	1050507	1050507	1
13	1.27853	0.652174	0.660377	0.654886	1.68229	765128	833847	0.389283
14	2.16582	0.375	0.477273	0.400763	2.37499	976249	1149604	0.168743
15	1.93575	0.44186	0.473684	0.446927	2.28712	783836	944724	0.195411
16	2.95558	0.391304	0.4375	0.397297	3.1843	765035	839865	0.124767

分析

RR调度策略虽然能保证任务调度的公平性，但对于CPU密集型任务的处理效果不佳。正是因为它过于注重公平，导致在调度决策中考虑的因素较为单一，无法针对不同类型的任务进行优化处理。

方案二：deadline越近的任务优先

基本思路

优先处理截止时间较近的任务。通过比较任务的截止时间，选择最接近当前时间的任务进行处理，以尽量减少任务超时的可能性。

得分

Test Case	Amplification	Finish Rate Hi Prio	Finish Rate Lo Prio	Finish Rate	Ave TL Rate	Elapsed Time	Cal Needed Time	Score
1	0.138182	1	1	1	1	1000036	1000036	1
2	0.185856	1	1	1	1	994316	994316	1
3	0.214204	1	1	1	1	1009456	1010066	1
4	0.170509	1	1	1	1	990066	990066	1
5	0.167653	1	1	1	1	1000821	1000821	1
6	0.2742	0.96	1	0.969828	1.02238	1093011	1094254	0.948594
7	0.212152	1	1	1	1	985488	985488	1
8	0.262445	0.962963	1	0.971888	1.02471	1060435	1060923	0.948451
9	0.26613	1	0.980769	0.993902	1.00208	1001267	1001267	0.991839
10	0.3438	0.941176	0.958333	0.946108	1.05905	1056123	1066419	0.893356
11	0.197316	0.976744	1	0.985075	1.00245	992708	992708	0.982663
12	0.153002	1	1	1	1	1050507	1050507	1
13	0.33933	1	0.943396	0.981289	1.02743	766871	833847	0.955095
14	0.255325	1	0.977273	0.994275	1.00077	976249	1149604	0.993505
15	0.736776	0.651163	0.789474	0.673184	1.10592	788422	944724	0.60871
16	0.708708	0.804348	0.8125	0.805405	1.24539	792106	839865	0.646712

分析

没有考虑到优先级的问题，可以看到最后几个case高优先级的任务完成率都比较低，并且这种策略也可能导致饥饿问题，如果总是有新的任务到达且其截止时间更近，较长的任务可能永远得不到执行

方案三：多级（3级）优先队列（MLFQ）

基本思路

将任务分配到不同优先级的队列中来管理任务的执行。每个优先级队列有不同的时间片长度，任务在使用完时间片后会被降级到下一个优先级队列。完成I/O操作的任务会被提升到最高优先级队列，以提高响应速度。调度器从最高优先级队列开始选择任务执行，确保短任务快速响应，同时防止长任务饥饿。I/O任务则通过优先级队列进行调度，保证其高效处理。

得分

Test Case	Amplification	Finish Rate Hi Prio	Finish Rate Lo Prio	Finish Rate	Ave TL Rate	Elapsed Time	Cal Needed Time	Score
1	0.13635	1	1	1	1	1000036	1000036	1
2	0.185576	1	1	1	1	994316	994316	1
3	0.211661	1	1	1	1	986937	1010066	1
4	0.180495	1	1	1	1	990066	990066	1
5	0.173129	1	1	1	1	1000821	1000821	1
6	0.238363	1	0.947368	0.987069	1.00434	1053525	1094254	0.982801
7	0.220745	1	1	1	1	985488	985488	1
8	0.28435	0.962963	1	0.971888	1.02471	1060435	1060923	0.948451
9	0.307862	0.958333	1	0.971545	1.0344	1001267	1001267	0.939236
10	0.358942	0.921569	0.958333	0.932136	1.06857	1056373	1066419	0.872324
11	0.227463	0.953488	0.982143	0.963753	1.01389	992708	992708	0.950547
12	0.159046	1	1	1	1	1050507	1050507	1
13	0.342065	0.934783	1	0.956341	1.00741	766642	833847	0.949307
14	0.559602	0.839286	0.818182	0.833969	1.11589	975148	1149604	0.747361
15	1.04063	0.534884	0.631579	0.550279	1.36185	785750	944724	0.404066
16	0.73237	0.782609	0.75	0.778378	1.16451	765831	839865	0.66842

分析

这个策略实现起来比较麻烦，并且最后效果也一般，理论上这个策略应该能适应各种类型的任务，有可能是我的优先队列级数设置的有问题。

方案四：任务优先级与deadline综合考虑的优先队列

基本思路

此方案结合任务的优先级和截止时间来决定任务的调度顺序。实际比较方式实现如下：

```
bool operator>(const Task &other) const
{
  float r1, r2, rate = 1.1（调参得到）;
  r1 = (this->priority == Task::Priority::kHigh) ? 1 : rate;
  r2 = (other.priority == Task::Priority::kHigh) ? 1 : rate;
  return deadline * r1 > other.deadline * r2;
}
```

得分

Test Case	Amplification	Finish Rate Hi Prio	Finish Rate Lo Prio	Finish Rate	Ave TL Rate	Elapsed Time	Cal Needed Time	Score
1	0.139324	1	1	1	1	1000036	1000036	1

Test Case	Amplification	Finish Rate Hi Prio	Finish Rate Lo Prio	Finish Rate	Ave TL Rate	Elapsed Time	Cal Needed Time	Score
2	0.185856	1	1	1	1	994316	994316	1
3	0.214204	1	1	1	1	1009456	1010066	1
4	0.170509	1	1	1	1	990066	990066	1
5	0.16841	1	1	1	1	1000821	1000821	1
6	0.2742	0.96	1	0.969828	1.02238	1093011	1094254	0.948594
7	0.212152	1	1	1	1	985488	985488	1
8	0.262445	0.962963	1	0.971888	1.02471	1060435	1060923	0.948451
9	0.266287	1	0.980769	0.993902	1.00208	1001267	1001267	0.991839
10	0.356491	0.941176	0.958333	0.946108	1.07849	1056123	1066419	0.877254
11	0.197316	0.976744	1	0.985075	1.00245	992708	992708	0.982663
12	0.158269	1	1	1	1	1050507	1050507	1
13	0.455083	1	0.773585	0.925156	1.06176	767205	833847	0.871339
14	0.265571	1	0.931818	0.982824	1.00455	976249	1149604	0.978376
15	1.02577	0.813953	0.526316	0.768156	1.26635	785882	944724	0.60659
16	0.886693	0.934783	0.5	0.878378	1.31831	791480	839865	0.666293

分析

这是我最后采用的方法，是基于方案二的改进，考虑到了任务的优先级，但其实优化效果并不明显，如果参数设置不当甚至还比不上方案二，经过调参后取得了微弱的优化。

运行截图