

schedlab实验报告

- 实验目的：理解并实现CPU调度算法，比较不同调度策略的性能差异
- 实验内容：编写一个模拟调度器的策略函数，并撰写实验报告
- 姓名：房向南
- 学号：2022201560

实验内容

在本部分我会重点介绍我的policy函数的优化过程，在最后一版会结合代码说明，前面的仅是描述思路。

FCFS策略及其优化

原始思路1

最早我使用了FCFS策略，只为测试和熟悉实验流程以及测试流程，并想搞清楚最简单的策略之一能达到多少分数。我首先使用了queue作为任务和io的队列的容器。对于每一次到来的事件，遍历事件列表，判断事件类型，根据不同事件类型做出相对应的操作。如对于taskarrival，我就先判断当前有没有任务正在执行，如果没有就直接把任务id赋值给action中的cpuTask；有的话就直接push进入队列。

得分：24，并且有未通过的点，查看错误发现是时间超过了maxtime。

优化方案1.1

我依旧抱着尝试的态度利用双向队列进行简单的优化，这次的主要思想就是区分任务的优先级，如果任务优先极高就pushfront，优先级低就pushback。其他方面不做任何改动。

得分：39，这次全部检查点都通过了，但是得分偏低。检查发现后几个点的低优先级任务甚至只有不到0.1的finish_rate。

ddl排序策略

原始思路2

我想利用stl中map的天然排序的优势，把键设置成任务的ddl，对应的值就是任务，让他们在map中按照ddl升序排列。对事件的处理没有变化。这一版我没有提交，但是从本地测试的角度看，结果不会太好。

之后我尝试改变事件处理策略，判断事件类型之后分别处理不如直接入队列然后以每次调用policy函数（即事件集合到达）为单位执行io和cpu切换。

优化方案2.1

在执行io和cpu切换的时候，挑选距离时间最近但是没超时的任务执行，这样可以进一步强化这个序列的优势。在这里我还遇到了一个问题，在每次处理完事件集合之后，不应该在当前cpu任务为空时才更换cpu任务，那样我设置的ddl升序队列就没有太大意义了。不出所料，有问题的思路只得了45分。

最终策略

我采用了set作为等待队列的容器，它的优点是：

1. 插入和删除简单，不需要像map那样用迭代器遍历整个数据结构。
2. 比较函数可以自己确定，而不是简单的升序排列。

我的比较函数如下所示。

```
struct Compare
{
    bool operator()(const Event::Task &a, const Event::Task &b) const
    {
        float aa = (float)(a.deadline-a.arrivalTime), bb = (float)(b.deadline-b.arrivalTime);
        a.priority == Event::Task::Priority::kHigh ? aa /= 1.012 : aa *=1.015;
        b.priority == Event::Task::Priority::kHigh ? bb /= 1.012 : bb *=1.015;
        return aa<bb;
    }
};
```

可以看到，我用了deadline减去arrivaltime作为排序的策略。我的想法是，加入arrivaltime作为考量，可以提前一些“时间紧任务重”的任务的完成时间，可以更好的应对后面的高压检查点。

但是这在最开始是行不通的，因为第一个检查点会出现超时的情况。我就采取了调整不同优先级的任务的权重的方法，灵活控制队列中任务的排序情况。经过我的测试，对优先级高的任务除以1.03左右的数，对优先级低的任务乘以1.01左右的数就能在保证不影响其他检查点的情况下通过第一个检查点。分数和检查点中高低优先级任务的finish_rate对这两个参数不算太敏感。

事件处理部分，因为使用了set，所以可以直接插入和删除,无需遍历。以任务到来和任务完成为示例：

```
if (events[i].type == Event::Type::kTaskArrival)
{
    taskset.insert(events[i].task);
}
else if (events[i].type == Event::Type::kTaskFinish)
{
    taskset.erase(events[i].task);
}
```

遍历完事件序列之后，如果当前的io为空，那就按照之前所说策略挑选一个事件去做io。cpu不用判断当前是否为空，直接切换为时间最紧张的未超时事件。

得分：89

实验总结

- 在本实验中，我先后使用了两种思路，并分别进行了优化，最后用ddl-arr的思路得到了89分。
- 通过本实验，我更好的理解了相关调度算法和cpu调度的过程。理解了不同策略在cpu调度上的性能差异
- 本实验贴合课堂内容，难度稍高，培养学生创新能力、提高学生专业素养。

完整代码和实验截图

RUC ICS 2023

🏠 首页

📖 题库

🏆 比赛

🔑 获取密码

← 返回比赛

佐伯沙弥香保护协会

比赛 - schedlab

您可以看到其他人的提交。

题目: 1

提交者: 佐伯沙弥香保护

分数: 0

~ 100

语言: 不限

状态: 不限

🔍 查询

👤 我的提交

编号	题目	状态	分数	总时间	内存	代码 / 答案文件	提交者	提交时间
#44700	#A. schedlab	— Partially Correct	89	1420 ms	20.59 M	C++ 17 (schedlab) / 2.5 K	佐伯沙弥香保护协会	2024-04-07 12:31:39
#44683	#A. schedlab	— Partially Correct	88	1434 ms	20.55 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-07 12:18:03
#44681	#A. schedlab	— Partially Correct	88	1435 ms	20.63 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-07 12:16:49
#44680	#A. schedlab	🚩 Judgement Failed	83	1438 ms	20.57 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-07 12:15:02
#44679	#A. schedlab	— Partially Correct	89	1450 ms	20.53 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-07 12:14:38
#44678	#A. schedlab	— Partially Correct	89	1421 ms	20.57 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-07 12:14:15
#44677	#A. schedlab	— Partially Correct	89	1416 ms	20.62 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-07 12:13:46
#44343	#A. schedlab	— Partially Correct	89	1430 ms	20.55 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-06 20:43:54
#44341	#A. schedlab	— Partially Correct	89	1448 ms	20.58 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-06 20:40:03
#44340	#A. schedlab	🚩 Judgement Failed	83	1428 ms	20.54 M	C++ 17 (schedlab) / 3.3 K	佐伯沙弥香保护协会	2024-04-06 20:37:50

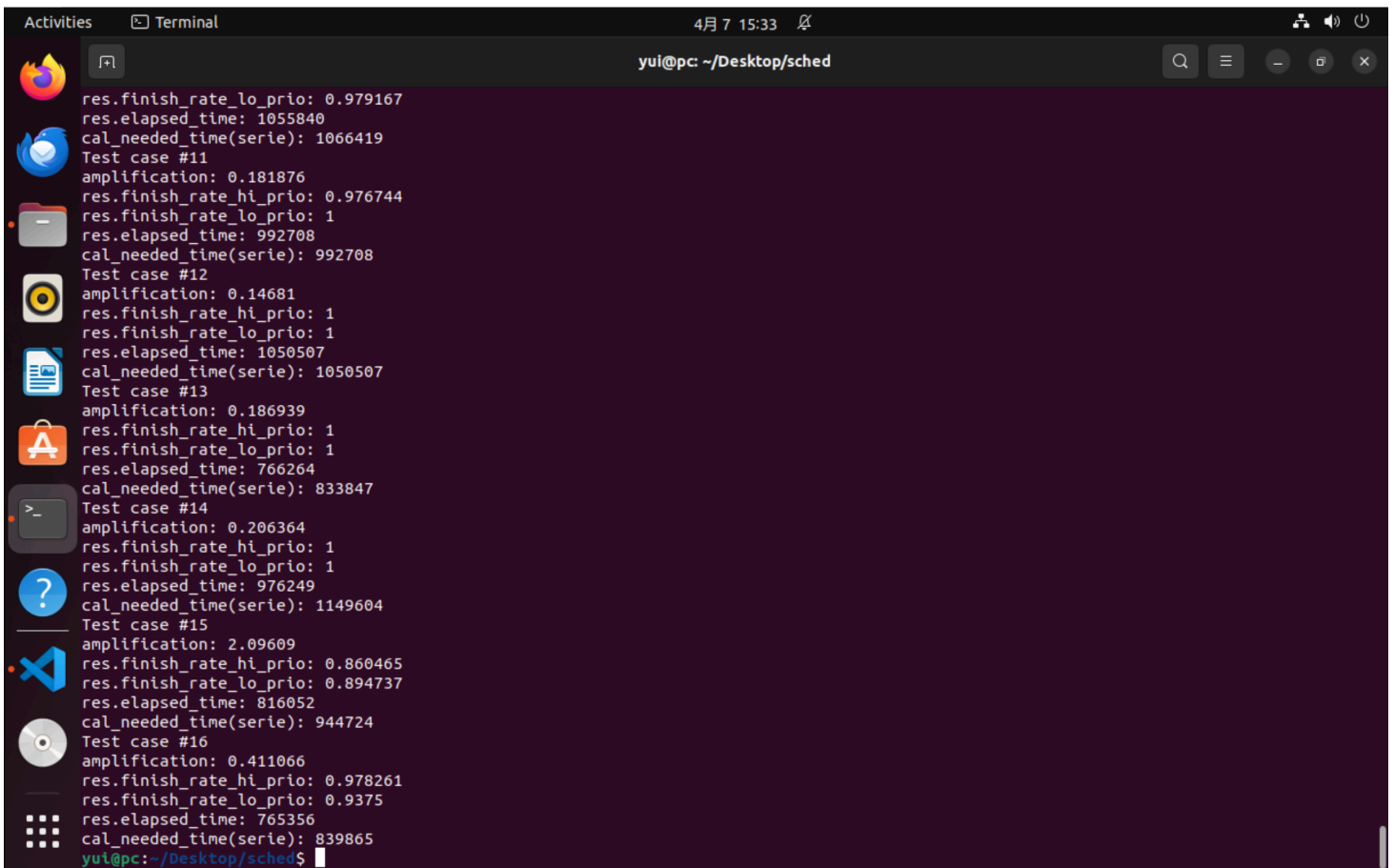
<

1

2

3

>



A terminal window titled "Terminal" with a dark background and light-colored text. The window shows the output of a program, likely a scheduler, displaying results for 16 test cases. The output includes metrics such as finish rate, elapsed time, amplification, and needed time. The window's title bar shows the date and time as "4月 7 15:33". The terminal prompt is "yui@pc: ~/Desktop/sched\$".

```
res.finish_rate_lo_prio: 0.979167
res.elapsed_time: 1055840
cal_needed_time(serie): 1066419
Test case #11
amplification: 0.181876
res.finish_rate_hi_prio: 0.976744
res.finish_rate_lo_prio: 1
res.elapsed_time: 992708
cal_needed_time(serie): 992708
Test case #12
amplification: 0.14681
res.finish_rate_hi_prio: 1
res.finish_rate_lo_prio: 1
res.elapsed_time: 1050507
cal_needed_time(serie): 1050507
Test case #13
amplification: 0.186939
res.finish_rate_hi_prio: 1
res.finish_rate_lo_prio: 1
res.elapsed_time: 766264
cal_needed_time(serie): 833847
Test case #14
amplification: 0.206364
res.finish_rate_hi_prio: 1
res.finish_rate_lo_prio: 1
res.elapsed_time: 976249
cal_needed_time(serie): 1149604
Test case #15
amplification: 2.09609
res.finish_rate_hi_prio: 0.860465
res.finish_rate_lo_prio: 0.894737
res.elapsed_time: 816052
cal_needed_time(serie): 944724
Test case #16
amplification: 0.411066
res.finish_rate_hi_prio: 0.978261
res.finish_rate_lo_prio: 0.9375
res.elapsed_time: 765356
cal_needed_time(serie): 839865
yui@pc:~/Desktop/sched$
```

```
#include "policy.h"
#include <stdio.h>
#include <set>

struct Compare
{
    bool operator()(const Event::Task &a, const Event::Task &b) const
    {
        float aa = (float)(a.deadline-a.arrivalTime), bb = (float)(b.deadline-b.arrivalTime);
        a.priority == Event::Task::Priority::kHigh ? aa /= 1.012 : aa *=1.015;
        b.priority == Event::Task::Priority::kHigh ? bb /= 1.012 : bb *=1.015;
        return aa<bb;
    }
};

std::set<Event::Task, Compare> taskset;
std::set<Event::Task, Compare> ioset;

int currentime=0;
```

```

Action policy(const std::vector<Event> &events, int current_cpu, int current_io)
{
    struct Action act;
    act.cpuTask = current_cpu;
    act.ioTask = current_io;

    for (int i = 0; (long unsigned int)i < events.size(); i++)
    {
        if (events[i].type == Event::Type::kTaskArrival)
        {
            taskset.insert(events[i].task);
        }
        else if (events[i].type == Event::Type::kTaskFinish)
        {
            taskset.erase(events[i].task);
        }
        else if (events[i].type == Event::Type::kIoRequest)
        {
            ioset.insert(events[i].task);
            taskset.erase(events[i].task);
        }
        else if (events[i].type == Event::Type::kIoEnd)
        {
            taskset.insert(events[i].task);
            ioset.erase(events[i].task);
        }
        }else if (events[i].type == Event::Type::kTimer)
        {
            currenttime = events[i].time;
        }
    }
    if (current_io == 0)
    {
        if (!ioset.empty())
        {
            auto it = ioset.begin();
            for (; it != ioset.end(); it++)
            {
                if (it->deadline > currenttime)
                {
                    act.ioTask = it->taskId;
                    break;
                }
            }
        }
    }
}

```

```

    }
    if (it == ioset.end())
        it = ioset.begin();
    act.ioTask = it->taskId;
}
}
if (!taskset.empty())
{
    auto it = taskset.begin();
    for (; it != taskset.end(); it++)
    {
        if (it->deadline > currenttime)
        {
            act.cpuTask = it->taskId;
            break;
        }
    }
    if (it == taskset.end())
        it = taskset.begin();
    act.cpuTask = it->taskId;
}
return act;
}

```