schedlab 实验报告

------2020200982 闫世杰

要求合理分配 cpu,io 资源
首先根据任务类型选择所要进行的处理：
kTimer:时间片结束导致中断发生,无需对事件进行处理
kTaskArrival:任务到来,将新任务添加到任务列表 list
kTaskFinish:任务结束,将完成的任务从 list 中删除,并将 cpu 资源设为空闲
kIoRequest:io 请求,将 list 中对应任务的 type 设为 kIoRequest,有 io 待完成,不能使用 cpu
KIoEnd:io 结束,将 list 中对应任务的 type 设为 kTaskArrival,io 结束,可继续使用 cpu 资源

处理完任务之后将 io_id cpu_id 设为任务到来时正在使用资源的进程 id
更新 list 中对应 cpu_id 的任务的工作时间(done),同时判断是否需要降低优先级 level
再进行时间判断,查看 time-period 是否结束,如果结束将所有的任务调至最高级 level9,同时根据任务队列中的任务数量更新 time-period 的大小

完成上述工作之后,对任务调度进行选择:
遍历计算任务列表 list 中各个任务 io/cpu 的优先级得分,选择优先级得分最高的任务进行调度

计算函数:io 及 cpu 得分采用相同的得分计算方式
截止时间 deadline 工作时间 donetime 等级 level 优先级 priority 到达时间 arrivaltime
ddl：为使任务及时完成,deadline 为首要因素,且剩余越短权重越高(使用函数$\frac{k}{lefttime^2}$
level: 只考虑高等级的任务
donetime: 在所有高等级的任务中选择工作时间较少的工作(等同于实施 Round Robin
(以该等级中工作的最长时间为基准,使用 k*(1-$\frac{donetime}{donetimemax}$ 计算
priority:优先级高的任务的得分加倍
arrivaltime:考虑 responsetime,赋予少许的权重
MLFQ 调度的实现难度不大,只需要注意在处理到来事件时维护变量的值以及一些临界情况即可
但是难在调整各个参数使得调度算法有较好的表现,由于在参数调试上难以前进,在不断尝试之后,最终只能达到 79 分
后选择转换思路,取消评分机制,直接依据 ddl 和优先级对最高层的任务进行排序,达到 85 分

最后转换做法:只简单的考虑 deadline 和优先级,不对任务进行分级,只是对所有的任务进行简单的排序,选择靠前的任务,并将其优先级调低(为防止单一任务独占 cpu 资源
前期所有处理都和上述的 MLFQ 相同,只是在调度选择上进行一个简单的排序,并进行选择及更新,最终获得 87 分
提交了两份代码,policy.cc 为简单排序做法,获得 87 分,MLFQ_max.cc 为 MLFQ 做法,获得 85 分