OS\_Project#2 MLFQ Scheduler

2013012148 Lee Jaeil

**1. Design new scheduler with MLFQ**

**Existing xv6 scheduler**

In initial xv6 scheduler, scheduler find RUNNABLE process from ptable sequentially.  
[In code] for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){ if(p->state != RUNNABLE) continue; proc = p; ...}

**1) 3-Level feedback queue**

In actually, I implement mlfq-scheduler not with queue, but with for-loops which runs in each level.  
First, distinguish processes with levels, level0(most priority), level1(second priority) and level2(lowest-priority). Scheduler picks processes in level0 first, and if there is no process in level0, move level1 and pick processes in level1(and next pick processes in level2). I implement this with if-else statement, like if(!isLev0Empty){ if there is no level0 proc, continue. else allocate isLev0Empty to zero(this make lev0 process start next loop)}else if(!isLev1Empty){ same }else{ same}. When process is created, put it in most priority level, level0(in allocproc function, initialize p->ticks and p->priority\_lev to zero. When checking time, if the process used all assigned time(level0, level1 and level2 are assigned 5, 10, 20 ticks), scheduler make process move to lower-level(p->priority\_lev++).

**1-2) get used cpu time**

First, in proc.h, i addition member variables, int ticks, int priority\_lev, in struct proc. And, at first, I record process's used time by using time interrupt handler, which yield() is called in(like proc->ticks++, which means that when time interrupt occurs increase current process's used time). But it works in random(Sometimes level2's result is zero;;). So I try to count used cpu\_time in scheduler(I think this is more correct). I make getTicks function(this function access ticks, during acquire(&tickslock)~release(&tickslock)), and I make some variables which difference of startTicks and finishTicks is saved in. Using these variables, scheduler makes a decision, whether reduce it's priority level or not.

**2) Round Robin policy**

In same level queue, Round Robin scheduler is applied. level0's time quantum is 5 ticks, level1's time quantum is 10 ticks, and level2's time quantum is 20 ticks. It means that level1's response time is 5ticks, and level1's response time is 10 ticks, and level2's response time is 20 ticks. In code, Once level0 process runs, it runs during 5 ticks. When it used 5 ticks, it's priority level is reduced and next level0 process will run(once process's priority level is reduced, it cannot increase until priority boosting). level1 and level2 is the same as level0.

**3) Priority boosting**

To prevent starvation, each process's priority is initialized by priority boost periodically. When It's time to do priority boost(100 ticks is time to do priority boosting), scheduler puts all processes will in level0, and make all processes's CPU time be initialized to zero. When priority boost is done, start MLFQ scheduling again. In code, boosting time is checked by calculating difference of current ticks and last boosting ticks. Scheduler boost priority when difference become 100 ticks.

**2. Stride (I only implement mlfq, so this is only my view.. sorry)**

**1) Request the portion of CPU and guarantee the calling process to be allocated that CPU time**

First, make cpu\_share system call which requests the portion of CPU. if process A call cpu\_share system call and request 30% CPU time, process will receive 30 tickets(if there are 100 tickets in total). Then, sum of the other processes's tickets are 70. When we think a large number is 2100, stride of A is 70 and stride of B is 30. stride of each process is recorded in each proc struct(if there is no process which records stride value, that is to say all processes's strides are 0, run MLFQ scheduling). So when process A runs, process A's pass value becomes 70(initial value is 0). As scheduler picks the process which has the lowest pass value, when we regard the other processes as process B, process B is picked three times(we assume that scheduler pick process every 1t). The other processes, process B, run in MLFQ scheduling during 3t. As process B's pass value becomes 90, scheduler picks process A. This mechanism is repeated until all job is finished.

**2) Process get CPU time up to 80%**

When process call cpu\_share system call, if total processes's CPU time is over 80% , cpu\_share system call is not accepted. Exception handler makes process not to request CPU time more than 80%. So there are more than 20% CPU time for processes which run in MLFQ scheduling.