CS 153 Lab2 Report - Implement priority scheduling

Qihua Liang; SID 861195605
Yunqing Xiao; SID X664888

I. Analysis

1. Priority Scheduler

The original scheduler in xv6 is a type of Round Robin scheduler, as it continually scans through all processes, and executes the first runnable process until all runnable processes are finished. It can be considered as Round Robin with a fixed time slice—a infinite period of time. The problem lies in is that once a long process is scanned and executed, all other processes have to wait until it is finished.

Priority scheduler outweighs R.R. in handling processes with high priority. It will choose next job to run based on its priority, which reflects the necessity of a specific process.

2. Priority with aging

Priority scheduler could lead to starvation as process with low priority may never get scheduled if new processes with higher priorities are continually feeded in. To avoid such starvation, priority aging is a general solution.

3. Priority inheritance

In prevention of reverse priority, the high priority process waiting for a lower priority process which is being executed and protected by lock at the same time should give its high priority to the lower one. Thus when a medium priority process comes in, it's priority is not high enough to preempt the original low priority process thus it can not be executed before neither the original high priority nor the low priority processes.

II. Implementation in codes

1. Priority Scheduler

In this project, we are modifying the scheduler to priority scheduler, which enables the execution order based on processes' priority. We added a field named "priority" to the structure of process. We define that the priority is integer numbers range from 0 to 31; and smaller the number is, higher the priority the process has. That is, 0 stands for the highest while 31 stands for the lowest priority. With this priority, we modified the scheduler function to enable it to find the process with highest priority to execute.

```
struct proc {
  uint sz;
  pde_t* pgdir;
  char *kstack;
  enum procstate state;
  int pid;
  struct proc *parent;
  struct trapframe *tf;
  struct context *context;
  void *chan;
  int killed;
  struct file *ofile[NOFILE];
  struct inode *cwd;
  char name[16];
  //CS153 Lab1 Part(0)
  int exitStat;
  //CS153 Lab2
  int priority;
};
```

To implement priority scheduler, when we scan the list of processes, if we find a runnable process, we will scan the list of processes again to check if there exists other process with a higher priority. If there is another process with higher priority than the process currently on hold, it means the current process is not the one with the highest priority. We will resume to check the next runnable process in the list. Once we find the process with highest priority, we will switch to execute this process.

```
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state != RUNNABLE)
      continue;

  struct proc *p2 ;
  bool isHighest = true;

  //CS153 Lab2 check if the RUNNABLE process is the highest priority
  for(p2 = ptable.proc; p2 < &ptable.proc[NPROC]; p2++){
    if(p2->state == RUNNABLE && p2->priority < p->priority) {
      isHighest = false;
      break;
    }
  }

  if(!isHighest) {
    isHighest = true;
    continue;
  }
```

2. Setpriority system call

The priority of a processes determines its order of execution and therefore changes the execution performance. In this lab, we just set the priority of a process when it is created, during which the system call setpriority is utilized.

```
void setpriority(int prior) {
  struct proc *curproc = myproc();
  //check prior range
  if(prior > 31)
    prior = 31;
  if (prior < 0)
    prior = 0;

  acquire(&ptable.lock);
  curproc->priority = prior;
  release(&ptable.lock);
}
```

3. Priority Aging

   To implement priority aging, we take the waiting time into consideration. Every time we select one process to be executed and others to wait, we decrease the priority value of waiting processes by one (that is, increase their priorities), and we increase the priority value of the selected process by one to decrease its priority. We also make sure that with such changes in priority, we do not violate the boundary of priority value.

```
//CS153 Lab2 aging implementation
for(p2 = ptable.proc; p2 < &ptable.proc[NPROC]; p2++){
  if(p2->state != RUNNABLE)
    continue;
  if(p2 != p && p2->priority > 0)
    p2->priority --;
  if(p2 == p && p2->priority <31)
    p2->priority ++;
}
```

III. Results of test cases

1. Priority Scheduler and setpriority system call

   Below is the result of running provided testcase.

```
$ lab2

 This program tests the correctness of your lab#2

  Step 2: testing the priority scheduler and setpriority(int priority)) systema call:

  Step 2: Assuming that the priorities range between range between 0 to 31

  Step 2: 0 is the highest priority. All processes have a default priority of 10

  Step 2: The parent processes will switch to priority 0

 child# 6 with priority 10 has finished!

 child# 5 with priority 20 has finished!

 child# 4 with priority 30 has finished!

 if processes with highest priority finished first then its correct
```

2. Priority aging

We modified provided test case to be with 5 children, each decrease the priority value by 1. All children are as below:
Child #4, priority 30
Child #5, priority 28
Child #6, priority 26
Child #7, priority 24

We modified provided test case to be with 4 children, each decrease the priority value by 2 (higher priority). What will happen in theory is as shown in the below table. (Time needed to execute each process is constraint by the two for loops in the test case, and it is only enough to switch to other process for one time.)

child #	original priority	try to execute #7	try to execute #6; finish #6	try to execute #7; finish #7	try to execute #5	try to execute #4; finish #4
4	30	29	28	27	26	DONE
5	28	27	26	25	26	25
6	26	25	DONE	DONE	DONE	DONE
7	24	25	24	DONE	DONE	DONE

Results in the below screenshot are correct.

```
child# 6 with priority 26 has finished!

child# 7 with priority 24 has finished!
child# 4 with priority 30 has finished!


child# 5 with priority 28 has finished!
```