Priority Round Robin Xv6 Scheduler Documentation:

Scheduler Basic Operation:

The new Xv6 scheduler takes the priority P into account. The priority indicates the base-2 logarithm of the maximum number of consecutive timer ticks the process may get during the "Round Robin" scanning of the processes.

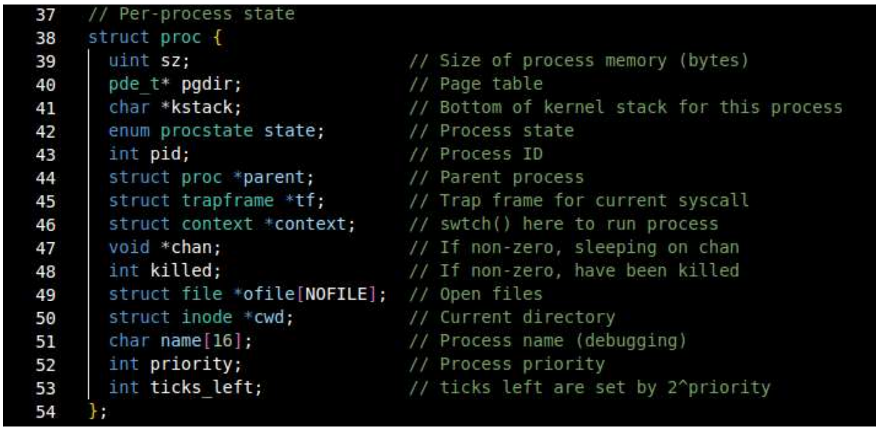
If the process is runnable (not sleeping), it should be scheduled 2^P times in sequence before considering the next process.

If before the process finished its 2^P ticks, it needs to sleep waiting on some event and is not runnable, it should not be scheduled, and the scheduler should continue in the round robin scanning of the next processes.

Kernel Changes:

Creation of 3 new system calls:

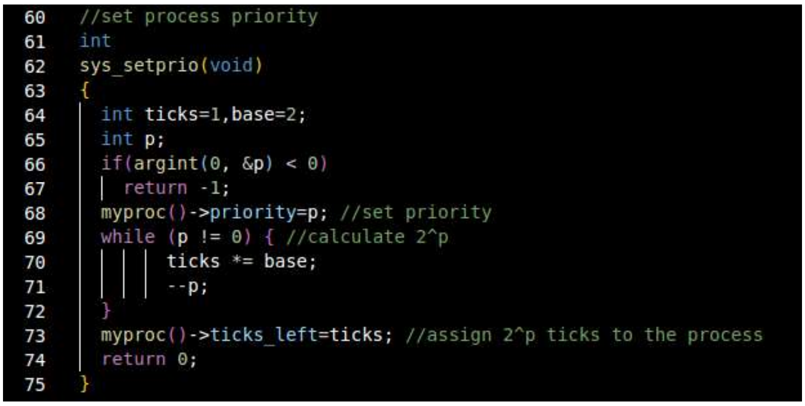
Firstly, we added two members to the process struct – priority and ticks_left:



```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int priority;                // Process priority
  int ticks_left;              // ticks left are set by 2^priority
};
```

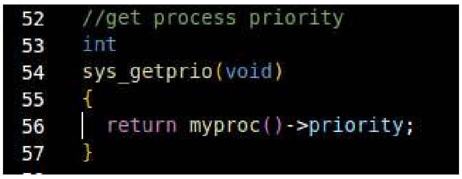
Then we defined three system calls:

1. setprio(P) to set process priority:



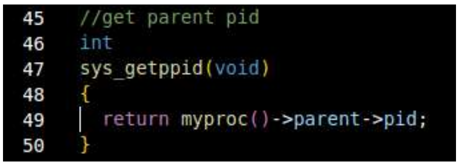
```
//set process priority
int
sys_setprio(void)
{
  int ticks=1,base=2;
  int p;
  if(argint(0, &p) < 0)
    return -1;
  myproc()->priority=p; //set priority
  while (p != 0) { //calculate 2^p
        ticks *= base;
        --p;
  }
  myproc()->ticks_left=ticks; //assign 2^p ticks to the process
  return 0;
}
```

2. getprio() to get process priority:



```c
//get process priority
int
sys_getprio(void)
{
  return myproc()->priority;
}
```

3. getppid() to get process parent PID:



```c
//get parent pid
int
sys_getppid(void)
{
  return myproc()->parent->pid;
}
```

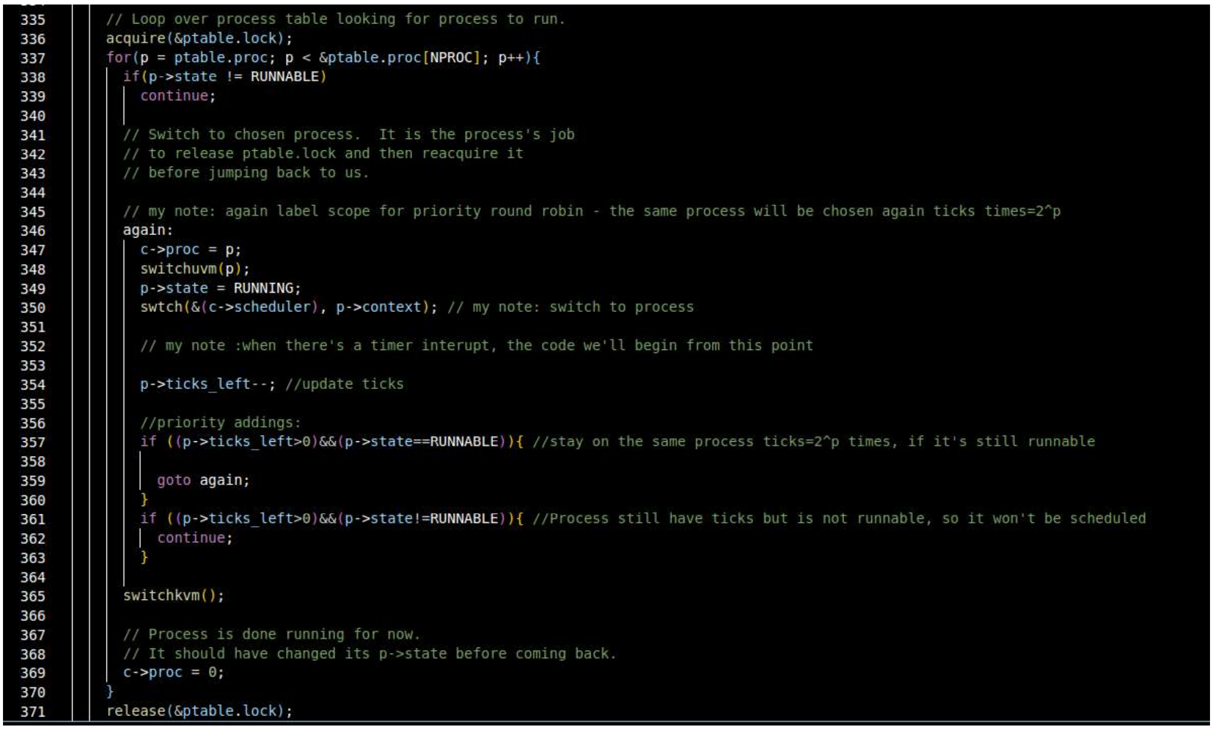
Note: all the suitable adjustments to create system calls were made in the relevant files.

Scheduler Adjustments for Priority:

The regular scheduler iterates over the ptable till runnable process is chosen, and right after the program switches to this chosen process.

We added this routine a priority management, by counting the ticks of the chosen process. As mentioned before: $ticks = 2^{priority}$.

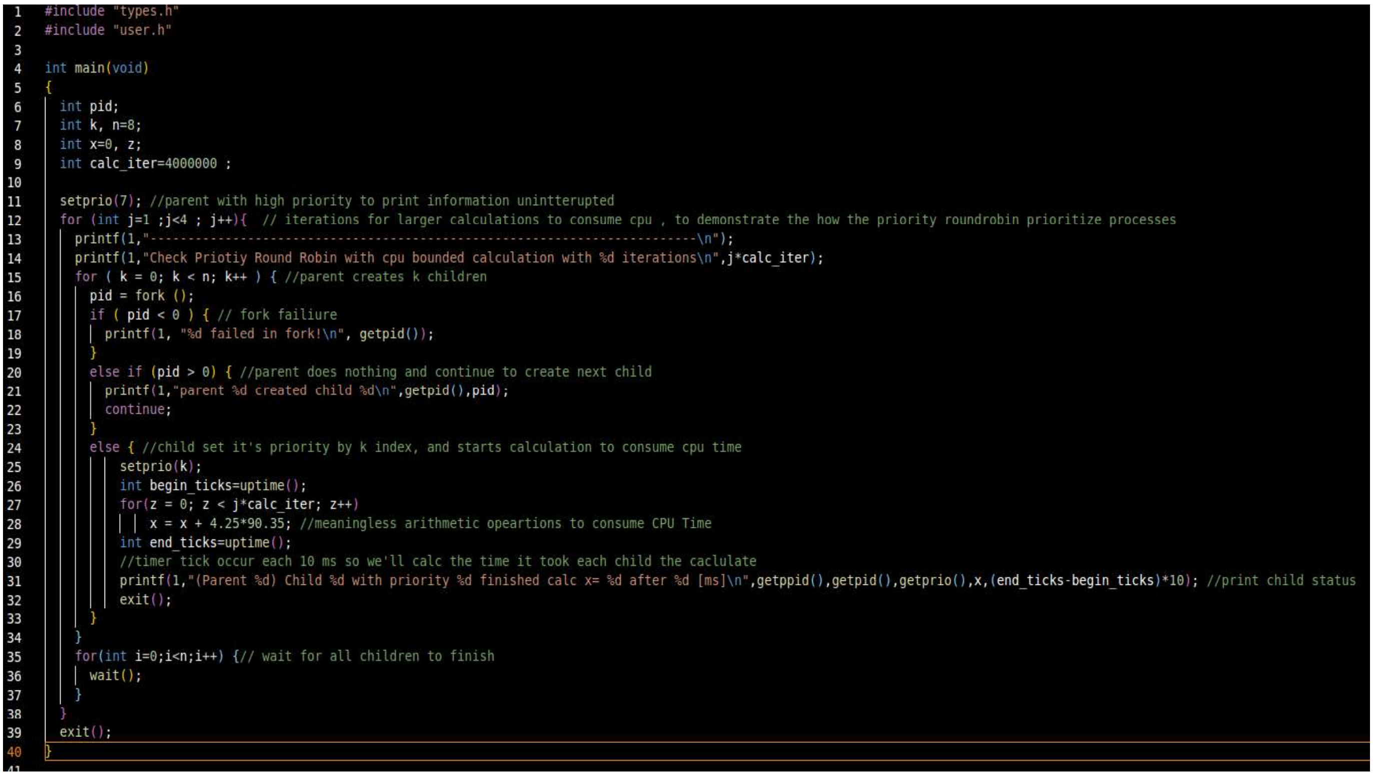
Inside the "again" label scope, if the chosen process still has ticks left and it is still runnable, it will be chosen again, till it has no ticks left. When the process ticks are over, the next process we'll be chosen. If a process still has ticks left but it's not runnable anymore, the next process will be chosen.



```c
// Loop over process table looking for process to run.
acquire(&ptable.lock);
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
  if(p->state != RUNNABLE)
    continue;

  // Switch to chosen process.  It is the process's job
  // to release ptable.lock and then reacquire it
  // before jumping back to us.

  // my note: again label scope for priority round robin - the same process will be chosen again ticks times=2^p
  again:
    c->proc = p;
    switchuvm(p);
    p->state = RUNNING;
    swtch(&(c->scheduler), p->context); // my note: switch to process

    // my note :when there's a timer interupt, the code we'll begin from this point

    p->ticks_left--; //update ticks

    //priority addings:
    if ((p->ticks_left>0)&&(p->state==RUNNABLE)){ //stay on the same process ticks=2^p times, if it's still runnable

      goto again;
    }
    if ((p->ticks_left>0)&&(p->state!=RUNNABLE)){ //Process still have ticks but is not runnable, so it won't be scheduled
      continue;
    }

  switchkvm();

  // Process is done running for now.
  // It should have changed its p->state before coming back.
  c->proc = 0;
}
release(&ptable.lock);
```

User Space Test Program "demosched":

We implemented a basic demonstration for the priority scheduler:



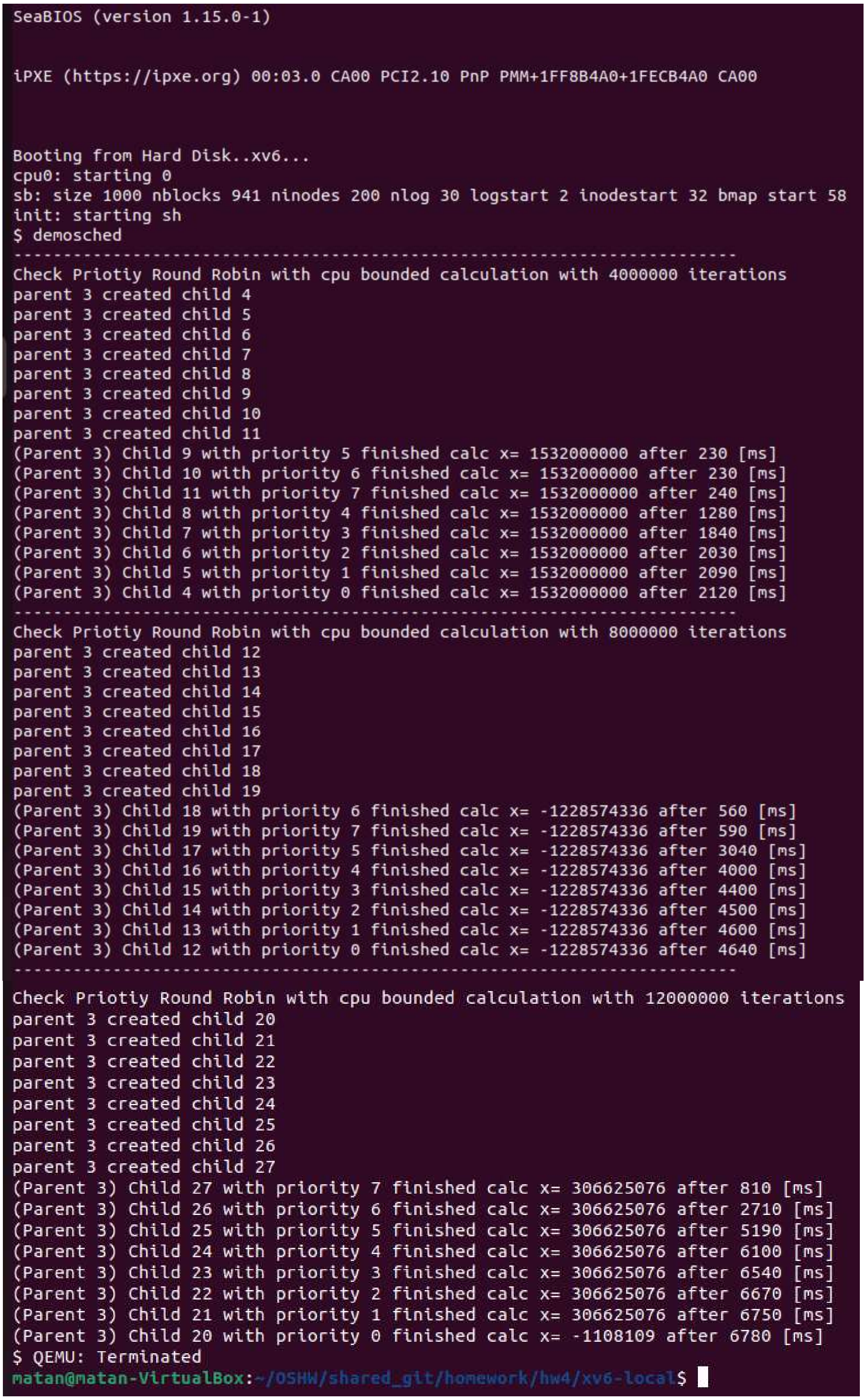
```c
#include "types.h"
#include "user.h"

int main(void)
{
  int pid;
  int k, n=8;
  int x=0, z;
  int calc_iter=4000000 ;

  setprio(7); //parent with high priority to print information unintterupted
  for (int j=1 ;j<4 ; j++){  // iterations for larger calculations to consume cpu , to demonstrate the how the priority roundrobin prioritize processes
    printf(1,"------------------------------------------------------------------------\n");
    printf(1,"Check Priotiy Round Robin with cpu bounded calculation with %d iterations\n",j*calc_iter);
    for ( k = 0; k < n; k++ ) { //parent creates k children
      pid = fork ();
      if ( pid < 0 ) { // fork failiure
        printf(1, "%d failed in fork!\n", getpid());
      }
      else if (pid > 0) { //parent does nothing and continue to create next child
        printf(1,"parent %d created child %d\n",getpid(),pid);
        continue;
      }
      else { //child set it's priority by k index, and starts calculation to consume cpu time
          setprio(k);
          int begin_ticks=uptime();
          for(z = 0; z < j*calc_iter; z++)
              x = x + 4.25*90.35; //meaningless arithmetic opeartions to consume CPU Time
          int end_ticks=uptime();
          //timer tick occur each 10 ms so we'll calc the time it took each child the caclulate
          printf(1,"(Parent %d) Child %d with priority %d finished calc x= %d after %d [ms]\n",getppid(),getpid(),getprio(),x,(end_ticks-begin_ticks)*10); //print child status
          exit();
      }
    }
    for(int i=0;i<n;i++) {// wait for all children to finish
      wait();
    }
  }
  exit();
}
```

- The parent process creates multiple child process by index n=8.
- Each child sets for itself a priority k, so the later the child was created the higher its priority (child 0 priority 0…. child 7 priority 7).
- Each child preforms a CPU bounded time calculation and prints a status message when it's finished with parent PID, self PID, self-priority and result.
- This routine is being executed 3 times for larger calculation each iteration of j.
- The larger the calculation the more the priority meaning can be observed (see example below)

The purpose of this demo is to show how a later created child finishes the calculation first due to its higher priority.

In the below example it can be observed that the last created child eventually finished the calculation first or second, when the calculation consumes more time.

Example:



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ demosched
--------------------------------------------------------------------------
Check Priotiy Round Robin with cpu bounded calculation with 4000000 iterations
parent 3 created child 4
parent 3 created child 5
parent 3 created child 6
parent 3 created child 7
parent 3 created child 8
parent 3 created child 9
parent 3 created child 10
parent 3 created child 11
(Parent 3) Child 9 with priority 5 finished calc x= 1532000000 after 230 [ms]
(Parent 3) Child 10 with priority 6 finished calc x= 1532000000 after 230 [ms]
(Parent 3) Child 11 with priority 7 finished calc x= 1532000000 after 240 [ms]
(Parent 3) Child 8 with priority 4 finished calc x= 1532000000 after 1280 [ms]
(Parent 3) Child 7 with priority 3 finished calc x= 1532000000 after 1840 [ms]
(Parent 3) Child 6 with priority 2 finished calc x= 1532000000 after 2030 [ms]
(Parent 3) Child 5 with priority 1 finished calc x= 1532000000 after 2090 [ms]
(Parent 3) Child 4 with priority 0 finished calc x= 1532000000 after 2120 [ms]
--------------------------------------------------------------------------
Check Priotiy Round Robin with cpu bounded calculation with 8000000 iterations
parent 3 created child 12
parent 3 created child 13
parent 3 created child 14
parent 3 created child 15
parent 3 created child 16
parent 3 created child 17
parent 3 created child 18
parent 3 created child 19
(Parent 3) Child 18 with priority 6 finished calc x= -1228574336 after 560 [ms]
(Parent 3) Child 19 with priority 7 finished calc x= -1228574336 after 590 [ms]
(Parent 3) Child 17 with priority 5 finished calc x= -1228574336 after 3040 [ms]
(Parent 3) Child 16 with priority 4 finished calc x= -1228574336 after 4000 [ms]
(Parent 3) Child 15 with priority 3 finished calc x= -1228574336 after 4400 [ms]
(Parent 3) Child 14 with priority 2 finished calc x= -1228574336 after 4500 [ms]
(Parent 3) Child 13 with priority 1 finished calc x= -1228574336 after 4600 [ms]
(Parent 3) Child 12 with priority 0 finished calc x= -1228574336 after 4640 [ms]
--------------------------------------------------------------------------
Check Priotiy Round Robin with cpu bounded calculation with 12000000 iterations
parent 3 created child 20
parent 3 created child 21
parent 3 created child 22
parent 3 created child 23
parent 3 created child 24
parent 3 created child 25
parent 3 created child 26
parent 3 created child 27
(Parent 3) Child 27 with priority 7 finished calc x= 306625076 after 810 [ms]
(Parent 3) Child 26 with priority 6 finished calc x= 306625076 after 2710 [ms]
(Parent 3) Child 25 with priority 5 finished calc x= 306625076 after 5190 [ms]
(Parent 3) Child 24 with priority 4 finished calc x= 306625076 after 6100 [ms]
(Parent 3) Child 23 with priority 3 finished calc x= 306625076 after 6540 [ms]
(Parent 3) Child 22 with priority 2 finished calc x= 306625076 after 6670 [ms]
(Parent 3) Child 21 with priority 1 finished calc x= 306625076 after 6750 [ms]
(Parent 3) Child 20 with priority 0 finished calc x= -1108109 after 6780 [ms]
$ QEMU: Terminated
matan@matan-VirtualBox:~/OSHW/shared_git/homework/hw4/xv6-local$
```