xv6-riscv Priority Scheduler — Implementation

Files You Will Edit

- kernel/proc.h
- kernel/proc.c
- kernel/syscall.h
- kernel/syscall.c
- kernel/sysproc.c
- user/usys.pl
- user/user.h
- user/test_priority.c (new file)
- Makefile

Step 1 — Add Fields to struct proc

Edit kernel/proc.h and add these fields inside struct proc:

```
struct proc {
  // ... existing fields ...

  int priority;   // 1 = highest, 10 = lowest
  int rounds;     // how many times scheduler ran this proc
};
```

Step 2 — Initialize the First Process (userinit)

Edit kernel/proc.c inside userinit() after creating the init process:

```
p->priority = 1;
p->rounds = 0;
```

Step 3 — Make Child Inherit Parent Priority (fork)

Edit kernel/proc.c inside kfork() after the child proc (np) is allocated:

```
np->priority = p->priority;
np->rounds = 0;
```

Step 4 — Add a setpriority(int) System Call

4.1 Add syscall number

Edit kernel/syscall.h and add SYS_setpriority using the next free number in your file:

```
#define SYS_setpriority  22
```

4.2 Register in the syscall table

Edit kernel/syscall.c:

```
extern uint64 sys_setpriority(void);

static uint64 (*syscalls[])(void) = {
  // ...
  [SYS_setpriority] sys_setpriority,
};
```

4.3 Implement the syscall handler

Edit kernel/sysproc.c:

```
uint64
sys_setpriority(void)
{
  int pr;
  if(argint(0, &pr) < 0)
    return -1;

  if(pr < 1 || pr > 10)
    return -1;

  struct proc *p = myproc();
  acquire(&p->lock);
  p->priority = pr;
  release(&p->lock);

  return 0;
}
```

Step 5 — Wire the Syscall in User Space

5.1 Add entry in user/usys.pl

xv6-riscv generates syscall stubs via user/usys.pl. Add:

```
entry("setpriority");
```

5.2 Declare in user/user.h

Add the function prototype:

```
int setpriority(int);
```

Step 6 — Add a User Test Program

Create a new file: user/test_priority.c

```
#include "kernel/types.h"
#include "user/user.h"

int main(int argc, char *argv[]) {
  int priority = atoi(argv[1]);
  setpriority(priority);
  while(1) {}
  return 0;
 }
```

Step 7 — Add Program to UPROGS in Makefile

Edit Makefile and add the entry under UPROGS (xv6 expects the underscore form):

```
$U/_test_priority\
```

Step 8 — Replace Round-Robin with Strict Priority Scheduler

kernel/proc.c

Edit kernel/proc.c and replace scheduler() with the code below. Key property: the scheduler picks exactly one best RUNNABLE process per outer loop iteration.

```
void
scheduler(void)
{
  struct cpu *c = mycpu();
  c->proc = 0;
  for(;;){
    intr_on();
    intr_off();

    struct proc *bestproc = 0;
    int bestpriority = 11; // priorities are 0-10

    for(struct proc *p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE && p->priority < bestpriority) {
        if (bestproc != 0) {
          release(&bestproc->lock);
        }
        bestpriority = p->priority;
        bestproc = p;
      } else {
      release(&p->lock);
      }
    }

    if (bestproc == 0) {
      // nothing to run
      asm volatile("wfi");
      continue;
    }
```

```
    bestproc->rounds++;
    c->proc = bestproc;
    bestproc->state = RUNNING;
    swtch(&c->context, &bestproc->context);

    c->proc = 0;
    release(&bestproc->lock);
  }
}
```

Step 9 — Modify procdump to Print Priority and Rounds

Edit kernel/proc.c inside procdump() and print extra fields:

```
printf("%d %s %s %d %d", p->pid, state, p->name, p->priority, p->rounds);
printf("\n");
```

Quick Test

Build and run xv6, then from the shell run several processes at different priorities:

```
test_priority 1 &
test_priority 5 &
test_priority 10 &
```

Press Ctrl+P to trigger procdump. You should see rounds increasing mostly for priority 1, then 5, then 10.