Operating Systems Lab

Week 7 Assignment

Name: SHISHU

Reg No: 2020CA089

Question 2. XV6 Process Priority Scheduling: In the previous question, we have learned how to change the priority of a process. In this question, we will implement a very simple priority scheduling policy. We simply choose a runnable process with the highest priority to run. (In practice, multilevel queues are often used to put processes into groups with similar priorities.) As we have done in the previous question, we assume that a process has a value between 0 and 20, the smaller the value, the higher the priority. The default value is 10. The program nice that we implemented in the previous question is used to change the priority of a process.

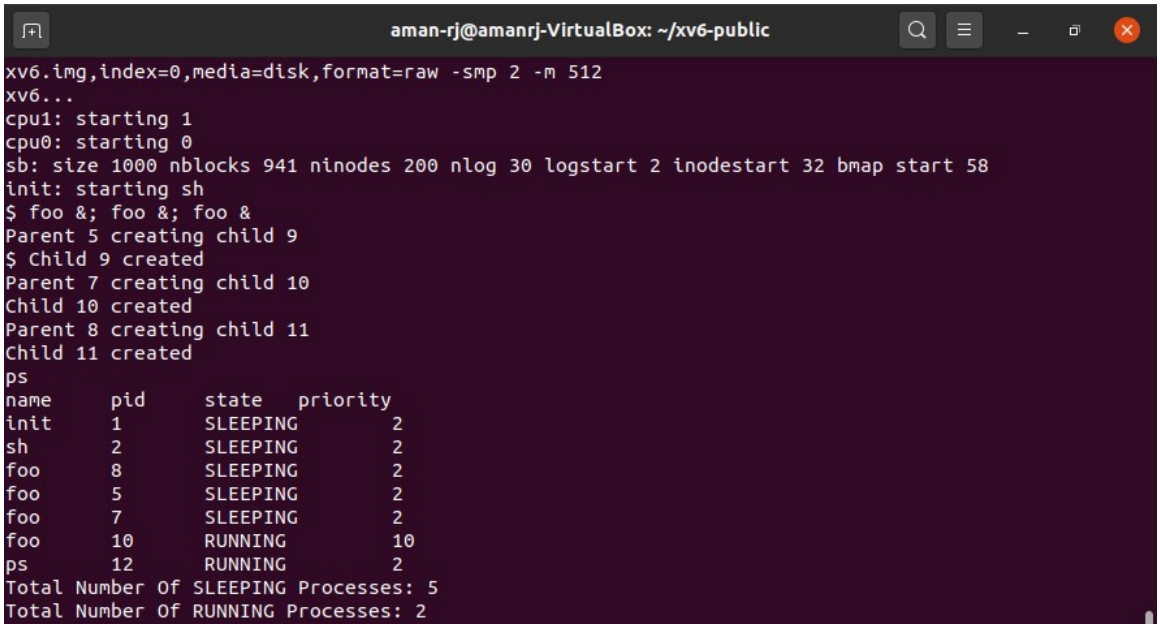
i. **Give high priority to a newly loaded process by adding a priority statement in exec.c**

```
 96  // Commit to the user image.
 97  oldpgdir = curproc->pgdir;
 98  curproc->pgdir = pgdir;
 99  curproc->sz = sz;
100  curproc->tf->eip = elf.entry;  // main
101  curproc->tf->esp = sp;
102  curproc->priority = 2; //A7
103  switchuvm(curproc);
104  freevm(oldpgdir);
105  return 0;
106
```

ii. **Modify foo.c so that the parent waits for the children and adjust the loop for your convenience**

```
17 for ( k = 0; k < n; k++ ) {
18 id = fork ();
19 if ( id < 0 ) {
20 printf(1, "%d failed in fork!\n", getpid() );
21 }
22 else if ( id > 0 ) {
23 //parent
24 printf(1, "Parent %d creating child %d\n", getpid(), id );
25 wait ();
26 }
27 else {
28 // child
29 printf(1, "Child %d created\n",getpid() );
30 for ( z = 0; z < 8000000.0; z += 0.01 )
31 x = x + 3.14 * 89.64; // useless calculations to consume CPU time
32 break;
33 }
34 }
```

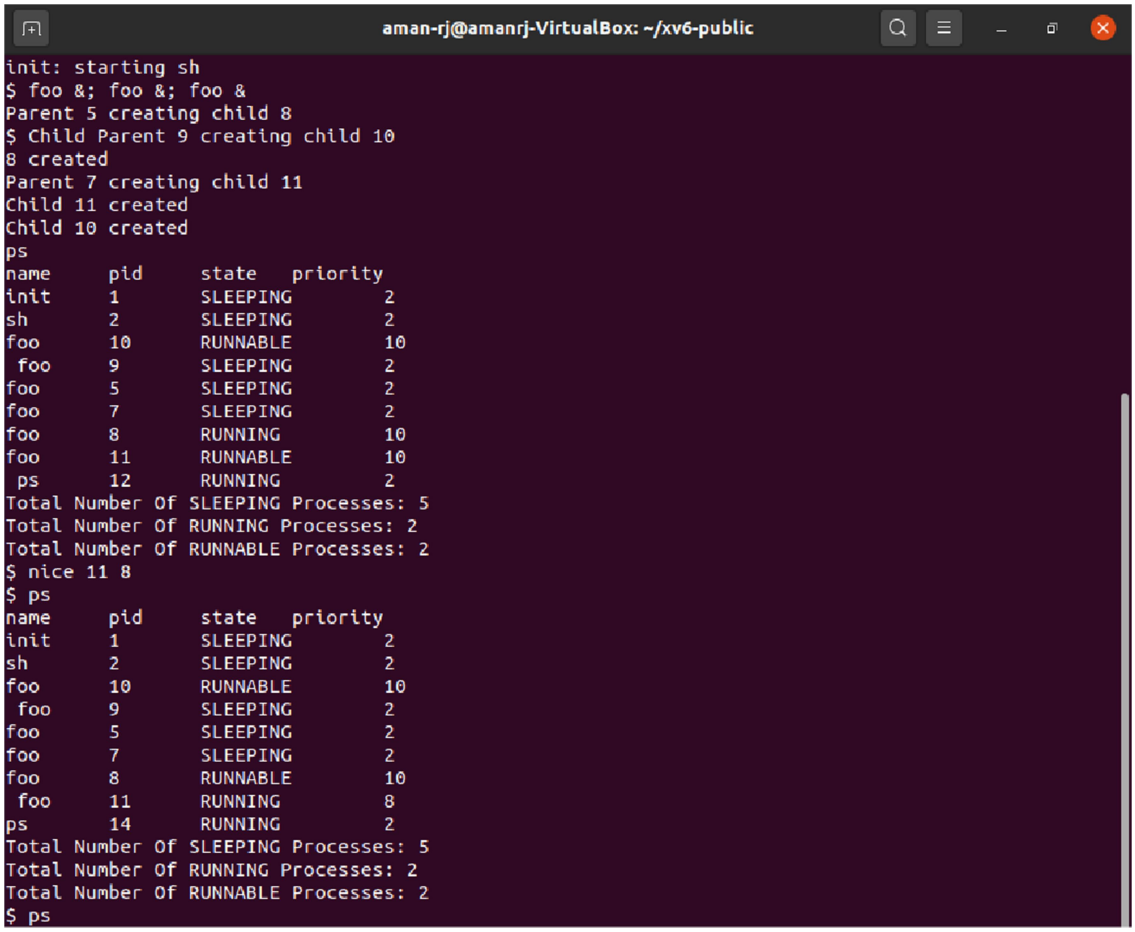
iii. **Observe the default round-robin (RR) scheduling**



iv. Implement Priority Scheduling

```
323 void
324 scheduler(void) //A7
325 {
326   struct proc *p;
327   struct proc *p1;
328   struct cpu *c = mycpu();
329   c->proc = 0;
330
331   for(;;){
332     // Enable interrupts on this processor.
333     sti();
334     struct proc *highP;
335     // Loop over process table looking for process to run.
336     acquire(&ptable.lock);
337     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
338       if(p->state != RUNNABLE)
339         continue;
340       highP=p;
341       //choose one with highest priority
342       for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
343         if(p1->state!= RUNNABLE)
344           continue;
345         if(highP->priority > p1->priority)
346           highP=p1;
347       }
348       p=highP;
349       // Switch to chosen process.  It is the process's job
350       // to release ptable.lock and then reacquire it
351       // before jumping back to us.
352       c->proc = p;
353       switchuvm(p);
```

v. Observe Priority Scheduling


aman-rj@amanrj-VirtualBox: ~/xv6-public
init: starting sh
$ foo &; foo &; foo &
Parent 5 creating child 8
$ Child Parent 9 creating child 10
8 created
Parent 7 creating child 11
Child 11 created
Child 10 created
ps
name pid state priority
init 1 SLEEPING 2
sh 2 SLEEPING 2
foo 10 RUNNABLE 10
foo 9 SLEEPING 2
foo 5 SLEEPING 2
foo 7 SLEEPING 2
foo 8 RUNNING 10
foo 11 RUNNABLE 10
ps 12 RUNNING 2
Total Number Of SLEEPING Processes: 5
Total Number Of RUNNING Processes: 2
Total Number Of RUNNABLE Processes: 2
$ nice 11 8
$ ps
name pid state priority
init 1 SLEEPING 2
sh 2 SLEEPING 2
foo 10 RUNNABLE 10
foo 9 SLEEPING 2
foo 5 SLEEPING 2
foo 7 SLEEPING 2
foo 8 RUNNABLE 10
foo 11 RUNNING 8
ps 14 RUNNING 2
Total Number Of SLEEPING Processes: 5
Total Number Of RUNNING Processes: 2
Total Number Of RUNNABLE Processes: 2
$ ps