Operating Systems–2: CS3523

January 2022

Programming Assignment 7: Priority Scheduling

Report

Readme:

To run the code for part-1:

Extract the file and open terminal in that directory:

Type on the terminal:

$make qemu-nox

This will boot up the xv6. On the xv6 shell to test out the user program type:

$myPrioritySched

Part-1:

In this part we add a system call into the xv6 to change the priority of the current process. For this a user program(myPrioritySched.c) is created which makes the function call(setPriority). The following files were edited in order to include system call:

- syscall.h: This file holds all the preprocessor directives for syscall. The number 22 syscall is used for setPriority.

- defs.h : This file has the definitions of the functions and structures. So, we include the setPriority() function.
- syscalls.c: This file is executed whenever there are system calls in the system. So, we include the setPriority syscall functions here.
- sysproc.c : This file contains the main code for any system call to work. We add the setPriority function in this file.
- usys.S: This file has the assembly code and the names of all the syscalls. So setPriority is added here.
- user.h : This file along with the previous file provides the user interface for the syscalls. The setPriority syscall is added here under the system calls list.
- proc.h : The struct proc is changed to include priority.
- proc.c: Default priority is made 5 here.
- myPrioritySched.c: This is the user program to execute.
- Makefile: This file is used to build and run the entire OS. The myPrioritySched.c is included in it.

When a process runs, it encounters the system call setPriority(int) which takes the priority as argument. The code is from sysproc.c:

```
92  int
93  sys_setPriority(void)
94  {
95    int priority;
96    if(argint(0, &priority) < 0)
97      return -1;
98    if(priority>9 || priority<0)
99    {
100     cprintf("Tried to change incorrect Priority with %d as priority for pid: \n",priority,myproc()->pid);
101     cprintf("Priority Unchanged!\n");
102     return -1;
103   }
104   myproc()->priority=priority;
105   return 0;
106 }
```

This code also checks for the invalid values of priority. And on line 104 it sets the priority. The change added in proc.h and process struct is:

```
108   int
109   sys_getPriority()
110   {
111     return myproc()->priority;
112   }
```

The order of priority is such that **10 is the highest priority and 0 being the lowest.** Default priority is 5 that is allocated when allocproc() function is called:

```
113   p->context->eip = (uint)forkret;
114   p->priority=DEFAULT_PRIORITY;
115
116   return p;
117 }
```

Q. What can be a problem with supporting such a system call in any OS?

Ans. The problem with such a system call is that any process whether it be a user or kernel one they can request to change their priority and make it higher even though it is not required. In this way the order of running processes gets changed.

Q. How could we mitigate this problem?

Ans. To mitigate this problem we can this system call accessible only in the kernel mode, so that only kernel can decide which process needs higher or lower priority.

Part-2

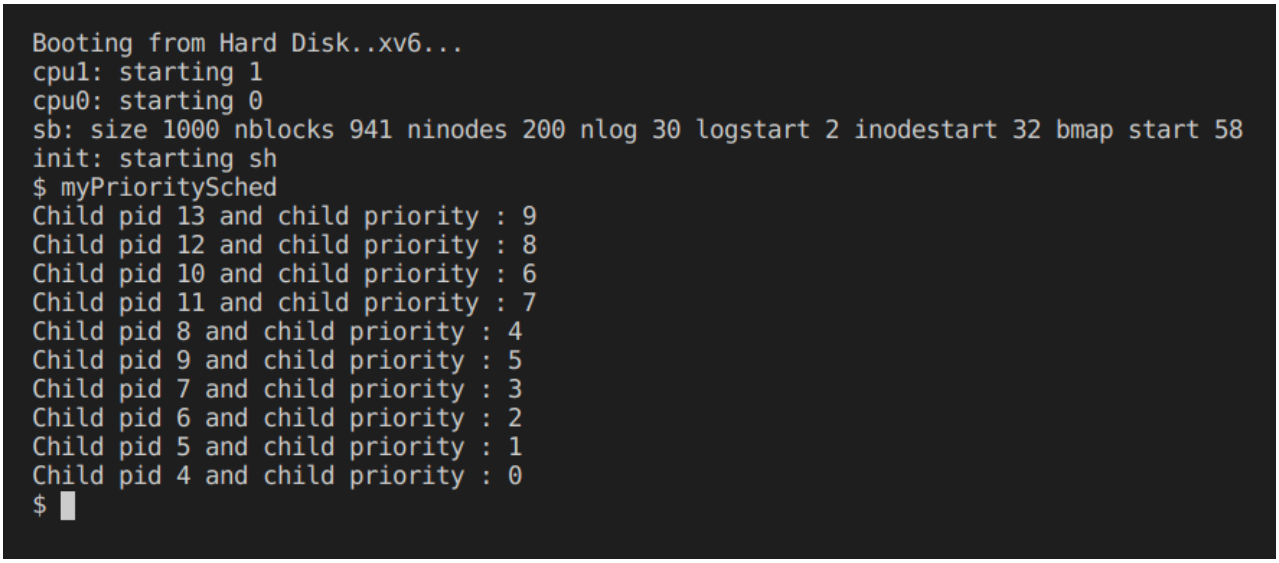
For this part the default process scheduling of xv6 is changed from round robin to higher priority scheduling first. The code for above is:

```
336     int max=-1, process_sched_pid=-1;
337
338     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
339         if(max<p->priority && p->state==RUNNABLE)
340         {
341           max=p->priority;
342           process_sched_pid=p->pid;
343         }
344     }
345
346     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){        // To change
347       if(p->pid!=process_sched_pid)
348         continue;
349
```

The process with max as priority and it should be runnable that process is chosen. If the processes have the same priority, then the default round robin strategy is used for scheduling.

When the user program is run on the xv6 shell using myPrioritysched the following output is printed:

Note: 1. When running on different systems the output might vary based on the hardware.



As we can clearly see, the higher priority processes are scheduled first.

In the user program that used, some extensive calculations are made to consume the cpu time so that meanwhile the scheduler schedules the processes in the order of priority.

Q. There is a problem with such a priority based policy. What is that problem?

Ans. The problem with such scheduling is that the lower priority process may suffer from starvation. If the processes with priority around 9 come repeatedly then lower ones will suffer and not get the chance to execute.

Q. How can it be mitigated?

Ans. To solve the problem we can include aging, i.e. studied in class in which the priority of older process increases with time. This way the priority of lower processes increases and they can be scheduled.