Lab Report: Scheduler

Win Thant Tin Han SID: 862258943

11/19/2023

1 Changes and Actions Taken

Here is a general summary of the steps I took as I worked on this lab:

- 1. 'syscall.h' – added new line to define
- 2. 'syscall.c' – added 'extern int sys_setpriority(void);' and '[SYS_setpriority] sys_setpriority'
- 3. 'usys.S' – added new syscall
- 4. 'defs.h' – added definition of setpriority under '//proc.c' : 'int setpriority(int priority);'
- 5. 'user.h' – added: 'int setpriority(int priority);' under system calls,
- 6. 'sysproc.c' defined function 'int sys_setpriority (void)'
- 7. 'proc.h' added variable under 'struct proc'
- 8. 'proc.c' worked on 'allocproc' and 'scheduler'. added code to 'fork()' so that child has parent's priority Also added new func: 'int setpriority(int priority)'
- 9. 'Makefile' changed cpu tick ('CPUS:=1') , added '_testùnder 'UPROGS'

1.1 Part 1: Adding New Field to Proc Structure

We need to initialize `priority_value` in allocproc() and fork(). The code is modified so that `priority_value` is initially set to 10. Additionally it is made so that the child processes will have the same priority as the parent.

1.1.1 proc.h and proc.c

The key section of this part of the lab is to define a variable in proc.h to store the priority value of the process. We add a priority value to each process (lets say taking a range between 0 to 31). When scheduling from the ready list we will always schedule the highest priority thread/process first. All the processes should have a default initial priority of 10.

1.1.2 Initializing in proc.h

I added another field in proc struct for proc.h: `int priority_value`.
Value range: [0, 31]; 0: highest; 31: lowest.

1.1.3 allocproc() in proc.c

`p->priority _value = 10;` was added to the `allocproc()` in `proc.c`. This made sure to set processes to have a value of 10 initially.

1.1.4 fork() in proc.c

```
    if (np->parent->priority_value < np->priority_value){
        np->priority_value = np->parent->priority_value;
    }
    else {
        np->priority_value = 10;
    }}
```

We needed to check if the priority value of the child is greater than the priority of the parent. If it was, it means it has low priority so we set it to the same value as that of the parent.

1.2 Part 2: Adding System Call to Update Priority Value

Followed same steps as Lab1 to add a new system call named `setpriority`. This modification required updates to several files, including `user.h`, `defs.h`, `sysproc.c`, `proc.c`, `syscall.h`, `syscall.c`, `proc.h`, `usys.S`, and `Makefile`. When this system call occurs, we change the priority value of the current proc. Then transfer control to scheduler immediately because the priority rank has been changed. This is done by calling `yield()`.

1.2.1 syscall.h, usys.S, and syscall.c

The function definition for the new system call is made in `syscall.h` and `syscall.c`.
`int setpriority(int)` has a system call number 22.
Similarly, `usys.S` is modified to have: `SYSCALL(setpriority)`

1.2.2 user.h and defs.h

We also have to add the system call type in `user.h` and `defs.h`, so we put `int setpriority(int priority);` underneath system calls in `user.h`.
`int setpriority(int priority);` was added to `defs.h`, under the comment for `//proc.c`

1.2.3 sysproc.c

The system call handler for `setpriority` function is defined so that it extracts the integer value passed as the first argument (index 0) and stores it in the priority variable. It then calls the `setpriority` function with `priority` as the parameter.

1.2.4 proc.c

The scheduler has been modified to always select the highest priority process from the ready list. This ensures that processes with lower numerical priority values are scheduled first.

For all these processes, we have to keep in mind to acquire and release appropriately, to control access to shared resources, ensuring that only one thread or process can access a critical section of code at a time.

1.3 Part 3: Updating the 'scheduler'

The next part of the lab involves changing the scheduler given to us from round robin to priority.
In Round Robin, we:

- Loop over process table (lock protected)
- Only choose the "RUNNABLE" process, which are ready to execute
- Next RUNNABLE process p acquires CPU resource and start RUNNING.

In priority scheduling, we:

- Loop over process table - find RUNNABLE proc with the highest priority.
- The chosen RUNNABLE process acquires CPU resources and start RUNNING.

1.3.1 proc.c

We update the `void scheduler (void)` function in proc.c. I followed these concepts while implementing the change:

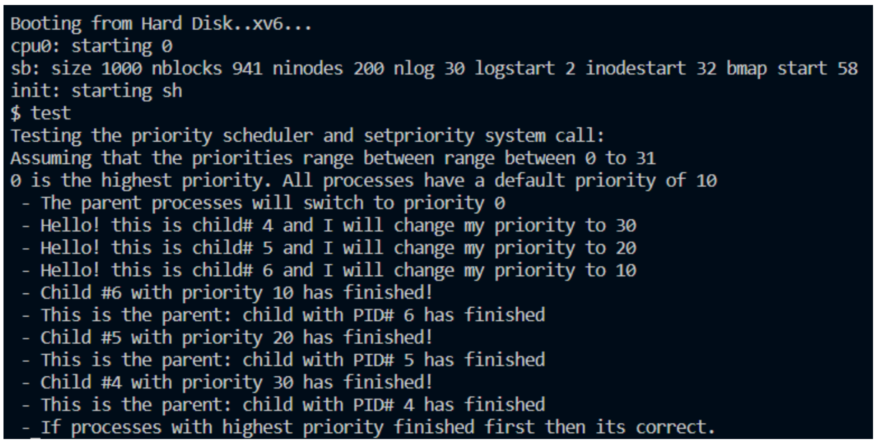
1. Enable interrupts on this processor.
2. Loop over process table looking for process to run.
3. Find the process with the highest priority among the RUNNABLE processes.
4. Check if this process has higher priority than the current chosen_proc.
5. If a RUNNABLE process with the highest priority is found, switch to it.

In scheduler, i declared and set an integer named `highest_priority`to 32 (larger than highest from 0 - 31). As the program goes through the list of processes, it will update the highest priority if it has a priority that is less than this highest priority. This is because we are assuming 0 is the 'highest priority.' In this way, after the iteration of the table, the process with the 'lowest value' (which is the highest priority) will run first.

2 Results and What I learned

From this lab, I learned the basics about how to implement scheduler. After going to lectures and office hours, I think i understand how to do it better, I think it was pretty fun and interesting. The output for test.c is:

Test4:



```
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
$ test
Testing the priority scheduler and setpriority system call:
Assuming that the priorities range between range between 0 to 31
0 is the highest priority. All processes have a default priority of 10
 - The parent processes will switch to priority 0
 - Hello! this is child# 4 and I will change my priority to 30
 - Hello! this is child# 5 and I will change my priority to 20
 - Hello! this is child# 6 and I will change my priority to 10
 - Child #6 with priority 10 has finished!
 - This is the parent: child with PID# 6 has finished
 - Child #5 with priority 20 has finished!
 - This is the parent: child with PID# 5 has finished
 - Child #4 with priority 30 has finished!
 - This is the parent: child with PID# 4 has finished
 - If processes with highest priority finished first then its correct.
```