

CSIS0230A Principles of Operating Systems (Class A)

Tutorial 8

Scheduling and nice level

The computer in front of you is configured so that the kernel is modified and has the counter we made during tutorial 3. A test program `testcounter.c` is also placed in the home directory, which demonstrates the increment and decrement of the counter, as well as getting its value and resetting the value to zero.

Write a program which create 4 new processes using `fork()`, adjusted to run at nice levels 0, 5, 10 and 15 respectively, by using `nice()` to add 5 to the nice level after each `fork()`. For convenience, we will call these processes 0, 1, 2 and 3. All of them should then call `inccount` a 1000000 times. Everytimes an unexpected number is returned (i.e., some other processes have called `inccount`), the process prints out the previous block of counter values it gets from the system call, in the following format:

```
1: curr = 60275, last range 23946-38280 (14335)
```

This means the process with name 1 makes the `inccount()` system calls 14335 times, getting values 23946 to 38280, but after that the next value obtained is 60275 rather than 38281. Output similar lines for the last group of values returned by the system call. This can be done using the following code:

```
/* define NUM_ITERATIONS here */
```

```
void do_work(int id) {
    int left = -1, last, i, curr = -1;
    for (i = 0; i < NUM_ITERATIONS; ++i) {
        if (left == -1)
            last = left = inccount();
        else {
            curr = inccount();
            if (curr != last + 1) {
                printf("%d: curr = %d, last range %d--%d (%d)\n",
                    id, curr, left, last, last-left+1);
                left = curr;
            }
            last = curr;
        }
    }
    if (left != last)
        printf("%d: curr = %d, last range %d--%d (%d)\n",
            id, curr, left, last, last-left+1);
}
```

After testing the program, **answer the following questions:**

1. The compiler compiles the increment operation within the kernel as a non-atomic operation, which loads the counter, add one to it and store the result back. Is it possible that two processes update the counters and execute the same instruction simultaneously, causing the kernel counter to be incremented once rather than twice? Why?

2. Is it still true if our machine has two or more processors?
3. When a process prints a message, did it wait? Evidence?
4. Reconstruct the complete scheduling sequence.
5. Use a larger number of iterations (e.g., 3000000) and run your program again. What is the most direct effect of the nice-level?

Note: Redhat modified the kernel in Redhat 7.3 to use a brand-new scheduler, which would become the standard scheduler in kernel 2.6 (or 3.0, whichever it is called). The new scheduler still works in epochs, and the nice level still translates to a time quantum given to the process for it to execute within a epoch. However, goodness is no longer used, and a process which wait will not have its time quantum carried over to the next epoch. Instead, the nice level is interpreted as a priority (as well as a time quantum). The kernel will store the ratio between the amount of time it sleeps and the amount of time it runs on CPU. If a process sleeps more, it will receive a temporary boost in priority, and if it sleeps less, it will receive a penalty. The boost or penalty will never be larger than 5, and will be in effect for the whole epoch. In this way, the OS doesn't need to give interactive processes extra time quantum to give it higher priority.

If a process sleeps really a lot, it is identified as “interactive”, and will be allowed to extend its time quantum. This allows an interactive task to be serviced a little bit longer than normal.

However, the most interesting effect of the new scheduler is that it can be, and is, implemented to run in constant time. The old scheduler requires scanning of the whole process table, so the more processes, the slower. The new scheduler needs the same amount of time to execute no matter how many process is running. See this in `kernel/sched.c` of the kernel.

All these information should be made into the lecture notes and notes of tutorial. Unluckily, the instructor knows that Redhat switched to use this scheduler only two days before the tutorial when he tests it. Please accept his apology.