Project 1

B07902053 許浩鳴

reference

Kernel Version

By `uname -r`, we can obtain the kernel version.

In my case, its `3.10.0-693.21.1.el7.x86_64`.

Design

There are three `.c` files: `main.c`, `process.c`, and `scheduler.c`.

`main.c` is the entry point, where the program obtains the process information and starts scheduling according to the given policy by calling the function `scheduling` from `scheduler.h`.

In `process.c`, I have implemented process resuming and idling for context switch, using the system call `sched_setscheduler`. Since `SCHED_OTHER` has a higher priority than `SCHED_IDLE`, we can block a process by setting the policy `SCHED_IDLE` and wake it up by setting `SCHED_OTHER`. In addition, I assign the actual process execution to another CPU by `sched_setaffinity`. This way, the processes will not be interrupted by the scheduler. Finally, I write logs of process executions to kernel buffer with two custom systemcalls mentioned in `kernel_files`.

`scheduler.c` provides algorithms for process scheduling. In order to reduce time complexity, one small trick I have done is to sort the process in advance by their ready time. Once a process is ready, it will be forked and become idled immediatly until it is waken up by the scheduler.

Comparison with Theoretical Results

As our expectation, there is no process executed while it is idling. This is because we assign a CPU to the scheduler exlusively, preventing it from interrupted by child processes. However, not every process is pushed to the ready queue immediately due to CPU context switch. Thus, the short period of delay would cause the overall execution time to be longer.

On the other hand, if the prediction of the scheduler ends ealier than the actual process, the result will still be the same, since the scheduler will wait for the child process to end and not affect the execution order.