Assignment4 Report: Process Scheduling

1 Logging scheduler level context switches

Context switching is performed by the function `context_switch()` in the file 'kernel/sched/core.c'. This function is called by `schedule()` function while process-scheduling. A hook (`hook_log_context_switch()`) was inserted before this call such that the associated lkm function increments a variable (counts) every time it is called. Thus it should return the number of context switches taking place.

As an experiment for checking correctness, I exported the function `nr_context_switches()` in the same kernel source file. The function returns the **total number of context switches since bootup**. When the kernel module is inserted, this function's return value is recorded. **Periodically**, its return value is taken and the previous value (before 2 seconds) is subtracted from it to get the number of context switches in teh 2-second-interval. This is compared against the number of context switches calculated by the module.

It was observed that the calculated context switches and kernel-function context switches were always almost equal. It was tested in both low load conditions and high load (induced by a c program `process_creator.c` which forked around 20 processes). The occasionally occurring minor mismatch is probably because of the context switches happening between the reading and storing of the two values in the kernel module.

2 Logging Run Queue Length distribution per CPU

The run-queue length of a CPU is given by it's `nr_running` variable. This is obtained by the call `cpu_rq(cpu)->nr_running`, where cpu is the number of the respective cpu. I added a new function `run_q_len(cpu)` in the file 'kernel/sched/core.c' which takes the cpu number as argument and returns its run-queue length. This function was exported and called every 2 seconds in the lkm using a timer. Thus, the queue length of each cpu is printed out every two seconds. The available CPUs are traversed using `for_each_online_cpu()`

macro.

I made a C program which forks new processes periodically. This program is executed to create a process every 2 seconds, to a total of 15 processes. While this is going on, the run queue length is being monitored by the kernel module. Every 2 seconds, the current run-queue length of the available cpus are printed out. It was observed that for each new forked process, it was added to the run queue of a cpu. As the experiment progressed, it was seen that the load was distributed among the cpus equally (when 10 processes were created, the run queue lengths in the two available cpus were 5 each). Now, the experiment is repeated with all the processes given a cpu affinity towards cpu 1. In the output, the run-queue length of cpu 0 was never greater than 1, and the run-queue length of cpu 1 reached 10 when all the 10 processes were forked. So, setting affinity to cpu 1 resulted in all 10 processes in cpu 1, denoting correct estimation of run-queue length.

3 Logging Number of Migrations

Four hooks were inserted in the file 'kernel/sched/core.c' to capture migrations. These are in functions `migrate_swap`, `try_to_wake_up`, `__migrate_task`, and `migrate_task_to`. For testing correctness, the number of migrations were analyzed in low load, high load and increasing load (load created by c program 'process_creator' by forking a number of processes). It was observed that as the load increases, the number of migrations were seen to be lesser. The load was processes on an infinite loop.