**Homework (Simulation) Wan Huzaifah bin Wan Azhar**

In this homework, we’ll use multi.py to simulate a multi-processor CPU scheduler, and learn some of its details.

**Answer:**

1. As it has only one job, it will take time 30 to finish as ‘a’ has run-time of 30
2. The job will run 2 times faster after warmup time of 10. So the job should finish around time 20. The job will run normally from 0 to 9 and speed up 2 times from 10 to 19.
3. If the job’s working size fit or less than cache size, the run time of the job will decrease by 2 each time after warm up time. As shown by -M 300 of question 2.
4. In the second question, the cache will become warm after the warm up time, which defaults is 10 (time 9), after that the cache is warm for the job and can be executed faster. For job ‘a’ of question 2, it becomes warm at time 9 as shown by -C flag.

By changing -w flag to higher or lower, the time it takes for the cache to warm for the job will be higher or lower. For example, if the -w flag is set to 5, the cache will be warm in time 4, and the job will run 2 times faster after that.

1. The multi-processor scheduler takes run-time of 150 finish these three jobs. From the -t flag, the job are switched up between CPU many times that it did not take advantage of cache as cache warm up time only happens after the jobs has changed to another CPU. Therefore, it keeps switching and none of the jobs use cache. But, it is still faster than one CPU.

From -C flag, none of the caches got warmed effectively. Each time a job is warmed, for example a, the CPU switch to job b and more.

1. The time it takes for all jobs to be done is 110. On CPU 0, Job a is using cache effectively as it is the only job in the CPU. On CPU 1, job b and c are alternating and each job is cached into the cache, which makes it run 2 times faster.

The only confusion is, why is job b and c on CPU 1 can both cache at the same time, but previous question (job a,b,c on all CPU) cannot? I think this is due to job b and c has working set of 50 and 50. Therefore, both job can be cached into the cache memory, which defaults to 100. But if job a is in the same CPU, it will displace job b and c’s cache because it is bigger than both.

If job a and job b on the same CPU, the time to finish the job is slower because job a take have higher working set. So, job a and b will compete to cache their job and in turn, no job can use caches effectively.

1. For n = 1, T = 300, n = 2, T = 150, n = 3, T = 100. For each number of CPU added, the time to finish all job decrease by factor of two. Cache is not considered here because working set of 100 means that all job cannot cache due to round robin.

For n = 1 and n = 2 but -M= 100, the Time to finish is the same as earlier. But for n = 3, T = 55 because each job is in one CPU. Therefore, cache can be done on all job and finished faster than smaller cache size.

1. Using per-CPU scheduling option is faster because as one job is done on one CPU, the CPU will pull another job one other CPU and thus, the time it take to finish all job is faster (due to caching). T= 100

Using P = 10 won’t change anything as CPU will check in Time 10 if there is any job to pull, but P = 40 will make all job finish longer because the CPU will take more time to check if there is any job to pull.

Per-CPU approach will decrease time to finish all job as number of CPU increase. It will work better as more CPU can effectively finish all job due to job pulling.