PERFORMANCE OF RT&ES

Scheduling Project

Introduction

The goal of this project is to implement several real-time scheduling algorithms and experiment with their use. The implementation includes rate-monotonic schedule, earliest deadline first, least slack time schedulers. The implementation was done on QNX the purple-box targets.

Design Methodology

For the project, mainly four threads were employed. They are the scheduler thread, idle thread, a time keeper thread and a thread was assigned for each task. The scheduler thread is set to high priority (10). The scheduler thread is responsible for scheduling task for different type of schedulers. Tasks run as per the algorithm and for different algorithms, tasks have different priorities. Scheduler threads calculate the priority of each task based on the algorithm we are using to schedule it. The priorities keep on changing for every task during their execution. So, the scheduler thread calculates the priority at each scheduling point and then schedule accordingly. The time keeper acts as a flag and is used to signal the task thread in every 5 milliseconds, so that it can schedule task for the next period. Pthread conditional wait and signal routines were used for this purpose.

For rate-monotonic algorithm, the scheduler thread calculates the period of all the tasks and then assign highest priority to the one with the shortest period relative to the other tasks. For earliest deadline first, the task thread determines the next deadline relative to the present execution period at each scheduling point. This is done for each task at each scheduling point. Task whose deadline is closest is assigned the higher priority as compared to other tasks. For least slack time scheduler, same procedure is followed. At each scheduling point, i.e. when the time keeper thread signals the scheduler thread, slack time for each task is calculated and then the task with least slack time is assigned highest priority. Since the scheduler that is employed is dynamic, the priority of tasks keep on changing accordingly.

All the tasks have a compute time and deadline and are periodic. Since tasks are periodic, deadline for each task is same as its period. Each task thread is designed for consuming CPU cycles for their execution time. Nanospin, which simulates as a real thread that is running was used for this purpose. The scheduler thread has many functions that are used for checking any missed deadlines, starting a new period for any thread. Logging was used to determine the execution time of each task. This was done by logging the start and end of any task. For overall analysis of the system kernel event log trace was captured. For switching between the algorithms, just changing of a macro is required. The thread_meta_t structure holds all the information about any task.

The Idle thread is used to free up the system when all the tasks are done. The code has to run for several test sets in which some are schedulable, some sets are failure and one identical set. For this purpose, several sets were made.

Task set 1: (1,3,3), (2,5,5), (1,10,10)
Task set 2: (1,7,7), (2,5,5), (1,8,8), (1,10,10), (2,16,16)
Task set 3: (1,4,4), (2,5,5), (1,8,8), (1,10,10)
Task set 4: (3,10,10), (3,10,10), (3,10,10)
Task set 5: (3,4,4), (2,6,6), (3,9,9)

Conclusion

The project helps us to study and know all about the different scheduling algorithms that are used. It is used to analyze the performance of all the three schedulers for different tasks.