**A Brief Description of the Testing Program SchedabilityTestEvaluation**

This document describes the Java project “SchedabilityTestEvaluation”, which contains the analysis of MSRP, PWLP and MrsP. For each protocol, three schedulability analysis are supported: the original analysis (package analysisBasic), our new analysis (package analysisNew) and our new analysis with implementation overheads (IO) accounted for (package analysisNewIO). In addition, we integrated the implementation of the ILP-based analysis via JNI to perform necessary evaluation and comparison. To perform evaluation, a system generation tool is developed to generate random systems with shared resources (package generatorTools), including task generation, priority assignment, allocation and resource usage generation.

Below list the major components of our implementation:

1. Three schedulability analysis (original, new, new with IO) for each protocol.
2. The ILP-based analysis for MSRP and PWLP (i.e., FIFO-NP and FIFO-P).
3. A system generation tool (task generation, priority generation, allocation generation and resource usage generation)

This document uses the new MrsP analysis as an example to demonstrate the implementation of all the analysis. The implementation is allocated in the MrsPNew.java file of the analysisNew package.

To invoke MrsP’s analysis function getResponseTime() should be invoked with a generated system and its shared resources passed as parameters. Then, through a set of iterative calculations (i.e., function busyWindow()), the response time of each task can be obtained to decide whether the given system is schedulable under MrsP.

In function busyWindow(), the response time of each task will be examined by computing the blocking time due to the direct spin delay, interference (with indirect spin delay accounted for) and arrival blocking. Each time when function busyWindow() is executed, the response time of all tasks will be updated. This function will be invoked continuously until (1) the response time of each task in the system is fixed (no response time updates after one execution of function busyWindow()) or (2) there exist at least one task with a response time higher than its deadline.

For each individual task, function directRemoteDelay(), highPriorityInterference() and highPriorityInterference() will be invoked to calculate the total response time based on its previous response time, which by default is the pure computation time of that task. With the response time of each task computed and fixed (in Function getResponseTime()), a decision of whether the given system is schedulable can be decided with a Boolean value returned. To facilitate accounting the blocking time, two helper function is realized: getNoRRemote() and getNoRFromHP(). These two functions return the number of requests to a given resource issued by local high priority tasks and tasks on a remote processor respectively.

Function directRemoteDelay() accounts for the resource accessing time required by the currently-studied task directly, including the potential delay from remote processors. This function goes through each resource that are requested by the currently-studied task. For each resource access, this function checks whether there exists an unaccounted request on each remote processor. If so, the remote request will contribute to the direct spin delay.

Function highPriorityInterference() firstly computes the pure interference from local high priority tasks based on the given response time. Then it invokes function getIndirectSpinDelay() to calculate the cost and potential delay due to high priority tasks accessing shared resources. At last, function highPriorityInterference() returns the total interference

Function localBlocking() computes the arrival blocking that the currently-studied task can suffer due to a local low priority task waiting/executing with a resource that has a priority equal to or higher than that of the current studied task, which includes the potential delay due to requests from remote processors to that resources. To achieve this, Function getLocalBlockingResources() is firstly invoked to return a set of resources that can block this task by its arrival. Then, the blocking time that can cause by each resource will be computed and the highest blocking time will be considered as the arrival blocking of the currently-studied task to capture the worst-case scenario.

The above briefly describes the working mechanism of the new MrsP analysis implementation. This analysis only considers the theoretical response time without any run-time overheads. In practice, additional cost will be imposed due to context switch and operating locks (lock and unlock function). In addition, tasks under MrsP can suffer from additional costs due to migrations. In package utils, the implementation and run-time overheads of each protocol is gathered and reported in file AnalysisUtils.java.

In package analysisNewIO, a run-time overheads-aware schedulability analysis of MrsP is realized, located in file MrsPIO.java (see function getResponseTimeDM()). This analysis is similar with the one described above. However, additional facilities are added to the account for the run-time overheads that a MrsP task can suffer during practice. Detailed description of analyzing the run-time overheads of tasks under MrsP is referred to Section 7- Runtime Overheads (coming soon).

Once a task is released, it incurs the overheads due to context switch so that the cost is added to the task’s response time (see function busyWindow()). In addition, in function highPriorityInterference(), the costs due to context switch should be accounted for in each pre-emption. Further, in each lock and unlock operations during the release of the currently-studied task, the implementation cost of MrsP should also be added into the response time of that task.

One major run-time overheads of tasks under MrsP is caused by migrations. In this analysis, new facilities are added to account for such costs, which are functions migrationCostForSpin（） and migrationCostForArrival(). These functions are invoked by each function that accounts for the blocking time (direct/indirect spin delay and arrival blocking). Both functions will invoke function migrationCost() to calculate the migration cost, which computes for two values: 1. The number of migrations without NP section (via a set of iterative calculations); and (2) the number of migrations with NP sections applied (via ceiling operation). Then the function takes the min value and returns the migration cost.