# Clustered Scheduling System User Guide

## I. Writing a Task

1. Include the task.h header file, which contains the following:



1. Define task\_t task, which is declared in task.h.
   1. The task struct must contain an init function pointer, a run function pointer, and a finalize function pointer (see task\_t definition above). Each of these function pointers has the same parameters and return type as a main function.
   2. The init function pointer is called once when the task starts, and the finalize function pointer is called once when the task finishes. These function pointers should perform one-time setup and cleanup work and may be NULL if they are not needed.
   3. The run function pointer is called once per period of the task. This function pointer should perform the real-time work of the task and must point to a valid function.
   4. All three function pointers receive the exact same arguments. The arguments are passed to all three functions to provide greater flexibility to tasks. You will specify the arguments in a configuration file (see “Creating a Configuration File,” below).

Example Task:



1. Do NOT provide a main function for your task. You will link your task with a main function provided by the clustered scheduling system (see “Compiling and Linking a Task”).
2. To create a parallel task, write your task with OpenMP.
   1. The provided object files and the instructions in “Compiling and Linking a Task,” below, assume your tasks use OpenMP.
   2. You may also use a different parallel runtime, such as Intel’s CilkPlus, but you will need to remove the OpenMP specific function calls (needed to make OpenMP suitable for real-time) from the task\_manager.cpp and utilization\_calculator.cpp source code and recompile everything for your parallel runtime.
   3. If you switch to a parallel runtime other than OpenMP, make sure your parallel runtime uses a greedy (or near greedy) scheduler.

## II. Compiling and Linking a Task

1. Make sure you have the following header file, Python scripts, archive, object file, and executable (source code and makefile are also available if you would like to make modifications):

task.h

cluster.py

lib\_cluster.py

libclustering.a

task\_manager.o

clustering\_launcher

1. Compile your task. Then link your task with the OpenMP flag, task\_manager.o (which contains the main function), and the –lrt, -lm, and –lclustering libraries.

Example:

Type the following in the terminal to compile and link a task consisting of just the single source file sample\_task.cpp (add more source files as necessary):



## III. Creating a Configuration File

The configuration file specifies the executable name, the arguments, and the real-time semantics of each task in the task set.

1. The configuration file should have a “.rtpt” extension (real-time parallel task set).
2. The first line of the configuration file should list the number of the first and last core that should be used by the clustered scheduling system (inclusive range) separated by a space.
3. For each task, provide two lines. The first line for each task should contain the executable name followed by the arguments that should be passed to the task program, separated by spaces.
4. The second line for each task should contain the empirically measured work and span of the task, the period and relative deadline of the task, the release time of the task relative to the other tasks at launch, and the number of iterations that the task should be executed.
   1. Note that the relative deadline should be identical to the period since only implicit deadline tasks are currently supported.
   2. All values, with the exception of the number of iterations, should be specified as two integers representing the seconds and nanoseconds components of the time. All values should be separated by a space.

Example sample\_config\_file.rtpt:



## IV. Running a Task Set

Run the clustering\_launcher executable.

* 1. Pass the name of the configuration file as the only argument. Do NOT include the “.rtpt” extension in the name passed to clustering\_launcher.

Example:

./clustering\_launcher sample\_config\_file

## V. Profiling a Task

Since you must specify the work and span of each task in the configuration file, we have provided a program to help you profile your tasks. This program will run a task repeatedly and report the maximum running time along with a histogram of all running times.

1. Make sure you have the following header file, archive, and object file (source code and makefile are also available if you would like to make modifications):

task.h

libclustering.a

utilization\_calculator.o

1. Compile your task. Then link your task with the OpenMP flag, utilization\_calculator.o (which contains the main function), and the –lrt, -lm, and –lclustering libraries.

Example:

Type the following in the terminal to compile and link a task consisting of just the single source file sample\_task.cpp (add more source files as necessary):



1. Run the profiler as follows:
   1. Specify the number of the first and last core (inclusive range) to bind the task to.
   2. Specify the width of the histogram buckets (in units of time) using two integers to represent the seconds and nanoseconds components of the time.
   3. Specify the number of times to run the task.
   4. Specify the arguments for the task.

./sample\_task\_utilization first\_core last\_core bucket\_width\_sec bucket\_width\_nsec num\_repetitions arg1 arg2 ...