Capstone Project Specifications

For this capstone you will build two programs, a worker and a controller. The controller acts as a central server managing tasks and potentially multiple workers. Each worker regularly attempts to connect to a server and receive tasks. The worker then executes those tasks and potentially sends back data to the controller if required by the task.

This document will explain the specifications for the worker and controller you need to build, along with a recommended order for implementing those specifications.

Also explained is the structure of the tasks that will be passed to the workers, and the task exchange protocol between workers and controllers.

Task Structure

For this problem you will handle 5 types of tasks. There is a task request and task response JSON format for each task. This table lists the fields and values required for these tasks.

Task	Description	Request Fields	Response Fields
Adjust Callback Interval	Changes the worker's callback interval	name: The task name interval: The new callback interval, measured in seconds	success: True or False
Run System Command	Runs a command on the system's shell	name: The task name command: The command for the system to run	success: True or False output: Output from running the command
Get File	Copies a file from the worker to the controller	name: The task name current_filepath: path to the file on the worker new_filepath: location to place the file on the controller	success: True or False contents: Contents of the file
Put File	Copies a file from the controller to the worker	name: The task name current_filepath: path to the file on the controller new_filepath: location to place the file on the worker contents: contents of the file	success: True or False

Task	Description	Request Fields	Response Fields
Delete Worker	Deletes the worker's file and stops it's process In that order	name: The task name	No response

Note: For the Get File task, when it is supplied to the controller the file contents will be unknown. It is up to the controller to handle getting the file contents and adding them to the task before it is sent to the worker.

Task Exchange Protocol

This protocol directs how all connections between a worker and controller will occur This protocol serves to ensure the worker and controllers fully receive all messages.

This protocol occurs in seven steps, with options to end early in special cases. After the connection is established:

1. The worker sends it's hostname to the controller.
2. The controller responds with the size in bytes of the task request JSON that will be sent to the worker.
    - If the controller has no tasks for the worker then it sends 0 and the worker will then close the connection.
3. The worker sends back the character '1' to indicate it is ready to receive the task.
4. The controller sends the task to the worker.
5. When the worker completes the task, it sends the size of in bytes of its task response JSON.
6. The controller responds with the character '1' to indicate it is ready to receive the task response JSON.
7. The worker sends its task response json and then closes the connection.

Controller Specification

The controller is a server that uses a provided port to handle connections from workers. When a worker connects, the controller then logs the connection. It then searches for any tasks that can be assigned to the worker. It sends one task at a time and as the worker requests them. It logs when each request is send and any responses from the worker if the task requires a response. The controller begins with a default list of tasks that it gives to all workers, but regularly checks a specified file for any additional tasks for specific workers.

Your controller will be implemented in the controller.py file.

Requirements:

- The controller will listen on port 443 by default, but will allow a different port to be specified.
- The controller will maintain a dictionary mapping all known hosts to a list of tasks for each host.
- Upon connecting with an unknown worker, it will add that worker to its dictionary and add a task to set its callback interval to five minutes.
- The controller will log the hostname and ip address each time a worker connects.
- The controller will log the hostname, ip address, task, and task response each time a task is completed by a worker.
- The controller will check for a file named "new_tasks.json" in its directory every minute. If present, this file should contain a list of tasks associated to hostnames it will add those tasks to its task dictionary. The controller will then delete this file.

Worker Guidance

The worker is a client that connects to a controller. It accepts tasks from a controller and executes them, sending back any relevant information. The worker calls back to a controller to request tasks at set intervals.

Your worker will be implemented in the worker.py file.

Requirements:

- The worker will default to a callback interval of 30 seconds.
- The worker will be created with a hard coded ip address and port to connect to.
- The worker will attempt to execute the Task Exchange Protocol upon a successful connection.
  - If the worker receives a task it will attempt to immediately reconnect to the server after completing its task.
  - If the worker does not receive a task it will sleep for its callback interval.
- If the worker cannot connect it will sleep for its callback interval.

Advice

1. Work through this project in small, manageable chunks.
2. It is very helpful to diagram what you want to do on paper before attempting to implement it in code.
3. Implement code to handle the Adjust Callback Interval task first. Once the controller and worker can fully complete that task, then start implementing the rest.
4. For testing while building the implant, use shorter callback times.

5. For testing while building the task logic, take advantage of how the controller creates default tasks for each worker to quickly test each task.

Good Luck! And don't forget to ask for help for the internet, your classmates, and the instructor!