BONUS TASKS IMPLEMENTATION MIND-MAP:

Bonus #1: Assume the filename `commands.txt` file no longer fits in memory. Tweak your solution to allow for this.
Solution: The following code snippet can take care of a file that does not fit in memory by reading and processing each command (line-vise) rather than loading all the commands in memory.

```
with open("commands.txt") as f:
    for line in f:
        process_line(line)
```

Bonus #3: Change the API so that instead of passing a file formatted with two sections listing valid commands and the command list, you come up with an alternative scheme of conveying this information. Maybe you want to pass these as arrays? What are the pros and cons of your chosen API relative to the original one we provided?
Solution: Rather than reading the commands line by line, we can change the code to accept two arrays. So we will use sys.argv[n] for inputs and sys.argv[n+1] for valid commands. However, such an exercise will be disadvantageous because if either array is too big, it will not fit in memory while reading a file line by line will not stretch memory constraints.

Bonus #6: Assume that invalid, "malicious", or erroring command strings could exist in the `VALID_COMMANDS` section now; if a "valid" command turns out to be invalid, treat it as though it were invalid in the first place.
Solution: Malicious command strings will attempt to write into the /tmp folder. The following snippet in command parser should take care of malicious commands.

```
command = queue.get(True)
if '> /tmp' in command or 'rm ' in command or 'rmdir' in command:
# do not execute
continue
```

Another method to account for this is to use subprocess.check_output instead of subprocess.run. In this way, we can make sure that we don't execute the commands and check what the command does before executing.