Advanced Programming (2024)
Assignment 6

Emil Valentin Ramsbæk, TDH424
Simon Brodersen, RZN875

October 24, 2024

Contents

1 Introduction

In general, we have followed the assignment's instructions and believe our implementation to be correct. We have taken some assumptions about implementation such as that a worker cancels a job, if it is running said job when it is shut down. Furthermore, we assume that a worker never accidentally crashes during execution and thus always finishes it's assigned work successfully. With these assumptions in mind, we believe that everything is functional.

2 Tasks

2.1 Adding Workers

When adding workers we make use of the Genserver module. This then runs the function workerMain, which loops infinitely unless it gets a workerShutDown message at which point it does no recursive call. This way when the worker is shut down it exits execution.

With this we believe our implementation is functional and correct.

2.2 Job Cancellation

When the server receives a cancel job, it refers the message to the worker currently running the job if one exists. As every job runs as a thread within the worker, then the worker is able to kill the thread during execution, which means that a job can be canceled even when it is running.

With this we believe our implementation is functional and correct.

2.3 Timeouts

For the timeout we have chosen to enforce the timeouts centrally in the SPC thread, this is done in the handleMsg which iterates over the running jobs

2.4 Exceptions

The worker receives a WorkerMsgJovTimedOut message and it handles this by killing the thread running the job and then notifying the SPC thread that the job has timed out

2.5 Removing Workers

When removing workers we took the assumption, that if a worker is removed while executing a job, then that job is canceled. This is a choice we have made instead of adding the job back to the pending queue. With this assumption in mind we believe our implementation is functional and correct. We make use of the jobDone function and its inner check of busy workers to make sure the state is updated correctly when the worker is removed.

3 Questions

3.1 What happens when job but no worker

In this case the jobs are enqueued in the JobsPending list, and wait indefinitely for a worker. We have tested this with "Add multiple workers" where we first add 4 jobs, check the status of these jobs before adding workers. These jobs are then pending, and we then add 4 workers after which all the jobs now have the status JobRunning.

3.2 Timeouts in SPC or workers?

We have implemented it such that when SPC sees that a worker contains a thread which is timeout, it sends a message to the worker about the issue.

The pros of this approach is that the server has an overview of the changes happening to all workers even if they work in concurrency. The cons are, that the server has to spend its computational time going over all the jobs running whenever it loops. If there are a ton of JobsRunning this could take a long time, which might be a significant cost if the server is larger. An observable effect would be this delay in the SPC servers response time.

3.3 Test worker works after crash.

We have a test called "job crash worker work" tests where we add a job that crashes. Makes sure it finishes and that it did indeed crash. Then we add a new job afterwards and that job still finishes as expected. Also note we only ever add a single worker.

3.4 Worker crash somehow?

This would most likely make the SPC server non-functional. If a worker crashes and the server does not realise, then it might still send jobs to the worker and expect the worker to finish the job. Then when the worker never finishes the Job the server will mark it as timed out and attempt to tell the worker to kill the thread. Which again the worker will not do, and the server will just assume that the worker did it. In general the SPC server assumes that the worker does its given work.