Parallel Programming in Python:

Determining the Most Efficient Way to Perform CPU-Intensive Multiprocessing Operations

Software Requirements Specification

Version 1.0

4/12/2021

Thomas Hickey

Lead Software Engineer

Prepared for:

NSU CS4233 — Professional Development in Computer Science

Instructor: Dr. Ted Ward

Spring 2021

**Table of Contents**

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Overview 1

2. General Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Characteristics 2

2.4 General Constraints 2

2.5 Assumptions and Dependencies 3

3. Specific Requirements 4

3.1 Functional Requirement 4

3.2 Use Cases 4

3.2.1 Use Case #1: 4

Using the software to determine the overhead of each method 4

3.2.2 Use Case #2: 5

Using the software to determine the efficiency of each method with different data loads 5

4. Analysis Models 6

4.1 Sequence Diagram 6

4.2 Activity Diagram 7

4.3 State-Transition Diagrams (STD) 8

# Introduction

## 1.1 Purpose

The purpose of this Software Requirement Specification is to demonstrate the necessary requirements in a program that is designed to test the multiprocessing operations available in the Python programming language for efficiency. The intended audience for this document is my instructor for this class, Dr. Ted Ward, and also anyone evaluating my work performed in the CS4233 class in the Spring 2021 semester at Northeastern State University.

## 1.2 Scope

(1) The software product being used for this project is Microsoft Visual Studio Code with the Python programming package installed.

(2) The final product will test the multiprogramming package of Python when operating in parallel for CPU-intensive applications. It will evaluate which of 3 available programming options is the most efficient: simplex pipes, duplex pipes, and the multiprocessing module built-in to Python known as the multiprocessing pool.

(3) The application of this software will be that it will provide conclusive data that can be used to determine which of three mechanisms available to a programmer for multiprocessing in the Python programming language show the highest efficiency, as measured by overall processing time, when the subprocesses themselves are performing very CPU-intensive tasks.

## 1.3 Overview

The rest of this document will attempt to further explain and specify the requirements for this software package. The 2nd section will attempt to make these requirements clearer and easier to understand, while the 3rd section will attempt to state the requirements with absolute specificity for the purpose of guiding the creation of the final product. Finally, the 4th section will include analysis models used in the development of these specific requirements.

# 

# 2. General Description

## 

## 2.1 Product Perspective

This software will fill an existing gap in knowledge regarding parallel programming with the Python programming languagein that, although there is plenty of documentation available to demonstrate how to implement these different methods of using the multiprocessing package in Python, there is currently almost no documentation available to show what the most efficient choice would be for various situations.

## 2.2 Product Functions

The software will perform CPU-intensive functions in parallel using each of the three specified methods of parallel programming in sequence while recording the processing time for each.

## 2.3 User Characteristics

The eventual user of this software would be any programmer that would need to make use of parallel programming using the Python programming language for CPU-intensive tasks. They would need to have a Python terminal or other software pre-installed that would allow them to run Python scripts. This would allow them to configure the software to match their needs and run a test to see which multiprocessing module would be most efficient for their needs.

## 2.4 General Constraints

The following are constraints that limit my options for designing this software:

* I will focus entirely on situations where subprocesses are running CPU-intensive tasks, and will only include two different tasks that are designed to test the ability of the channels between the main process and the subprocesses. Unfortunately, no two tasks could possibly encompass the entire scope of possible uses of the multiprocessing model.
* Averages will have to be obtained over multiple subprocess iterations, which may serve to cover some situational differences that could otherwise be explored. I will attempt to offset this possibility as much as possible by making a sufficient number of runs through the task cycles to be representative of a true average.
* I only have access to one PC to use for testing this software, so there will be no way to determine with absolute certainty how much effect this will have on the efficiency of these different algorithms and what sort of results could be obtained using a different operating system or hardware setup at this time.

## 2.5 Assumptions and Dependencies

The following are assumptions that were made in the design of this software to allow for it to fulfill the stated requirements:

* The chosen CPU-intensive tasks will accurately represent a reasonable average of all CPU-intensive tasks that could be called upon to be performed with the multiprocessing module in Python.
* By averaging many different runs of the software algorithm any possible inconsistencies that could arise from operating system demands or other unpredictable background CPU usage will be accounted for.
* That my PC will demonstrate a realistic average performance that will remain reasonably accurate when scaled up or down to match which of the almost unlimited number and combinations of hardware and software are being used by another programmer.
* That the relative efficiency of each method of moving data between the main process and the subprocesses can be evaluated using the processing time required for each method performing the same task.

# 3. Specific Requirements

## 

## 3.1 Functional Requirement

3.1.1 Introduction

* The first and only functional requirement of this software is that it be able to record the processing time for two different CPU-intensive tasks when they are run in parallel in different combinations using the three different multiprocessing methods specified above.

3.1.2 Inputs

* The user should be able to specify how much data is being processed by each parallel process, how many times each task is run, as well as how many times each method of multiprocessing is being used. The user will also be able to specify how many loops the overall script will be executed.

3.1.3 Processing

* The script will automatically perform the processing of the CPU-intensive tasks after the number of desired loops is specified and will time these processes and record those times without further guidance from the user.

3.1.4 Outputs

* The script will record the processing times for each looped process and each multiprocessing method. It will calculate total times and average times and record this data in a text file for later review from the user.

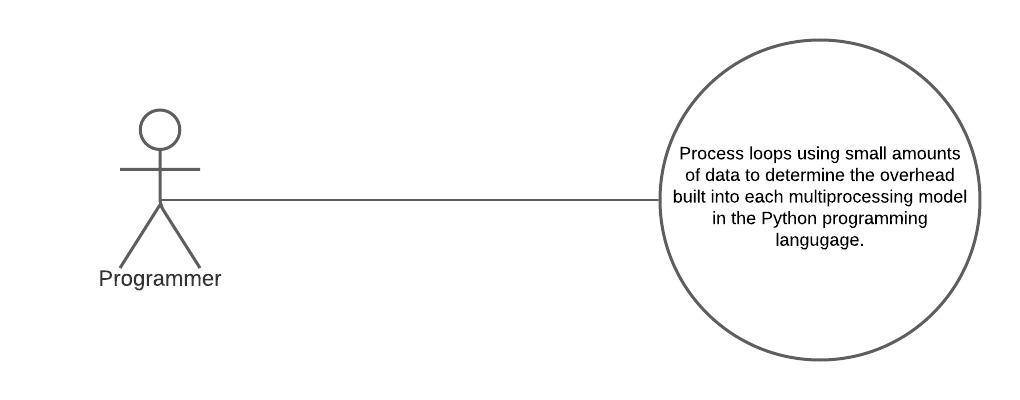
3.1.5 Error Handling

* The script will detect any errors in filename or output being recorded and alert the user in the case of any error.

## 3.2 Use Cases

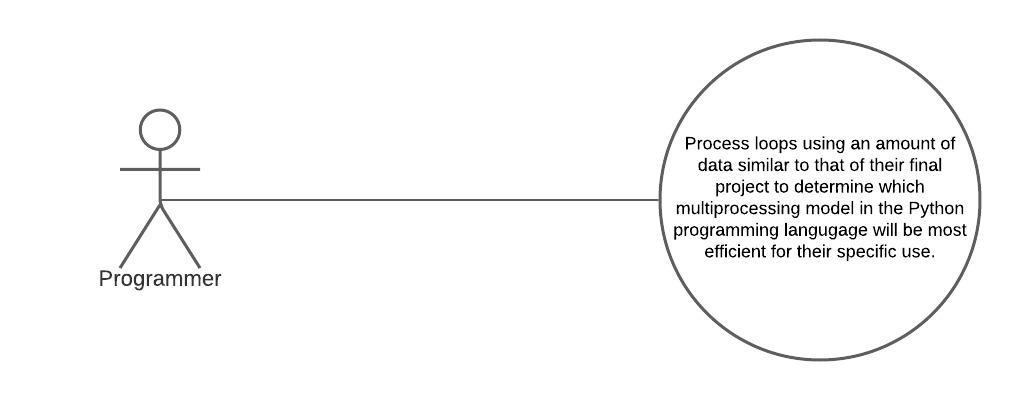
### 3.2.1 Use Case #1:

### Using the software to determine the overhead of each method



### 3.2.2 Use Case #2:

### Using the software to determine the efficiency of each method with different data loads



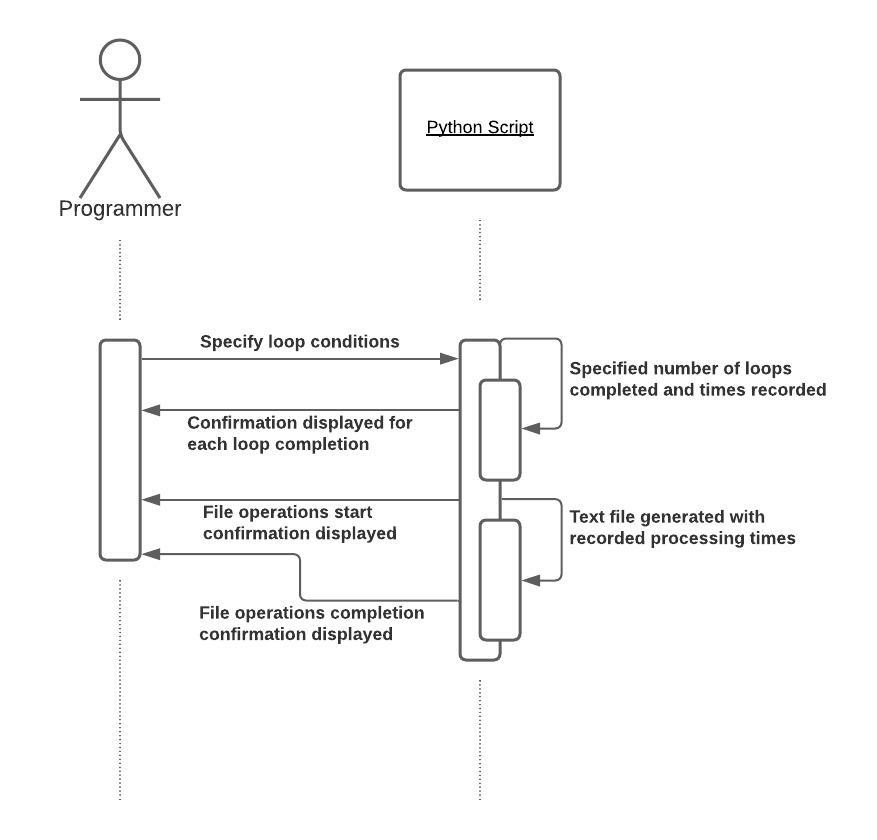
# 

# 4. Analysis Models

## 

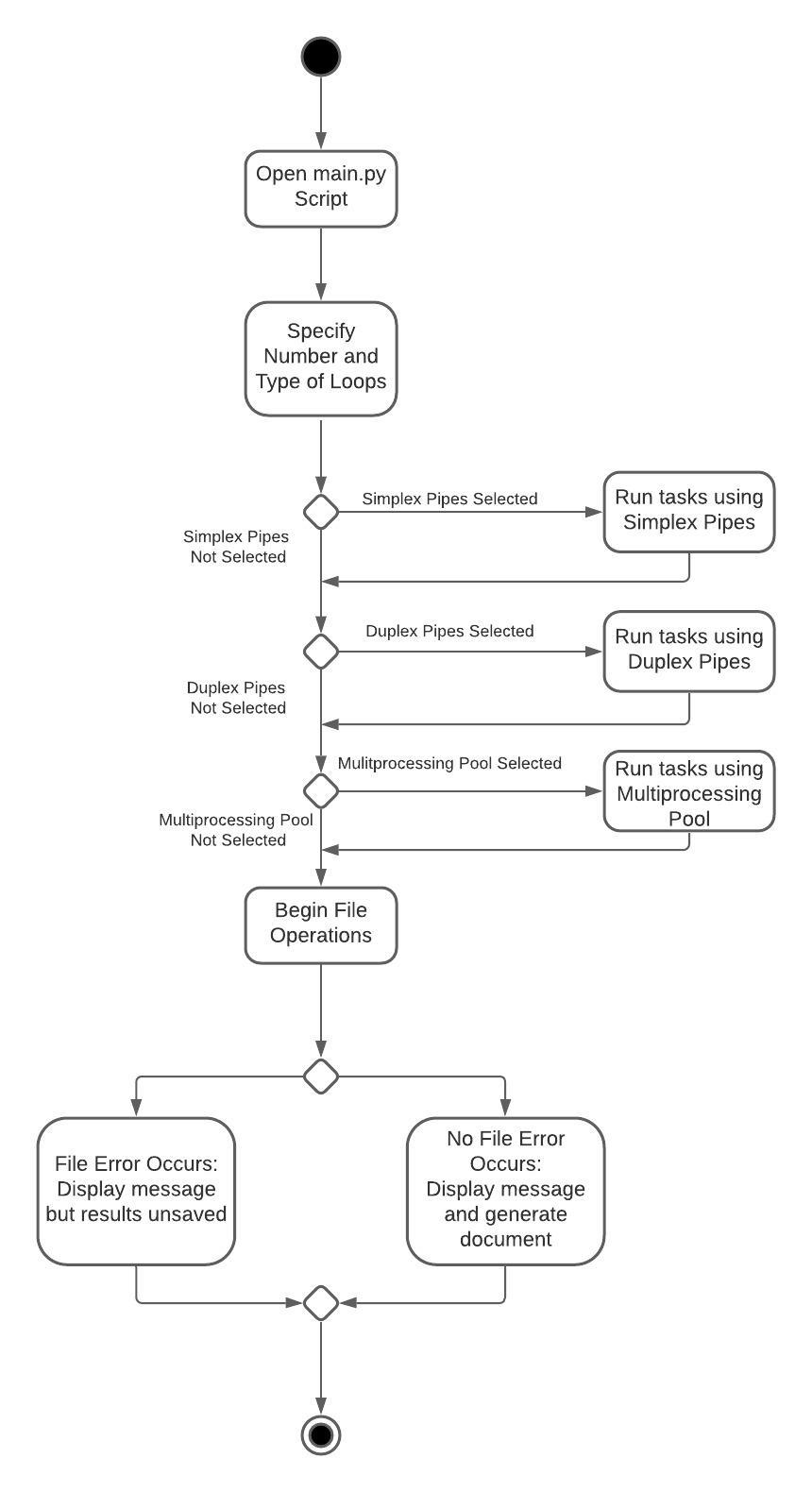
## 4.1 Sequence Diagram

Shows the general interaction between the programmer and the software



## 4.2 Activity Diagram

Shows the general flow of the program from beginning to end depending on the type of multiprocessing being tested



## 4.3 State-Transition Diagrams (STD)

Shows an outline of the main states for the driver script from beginning to end

