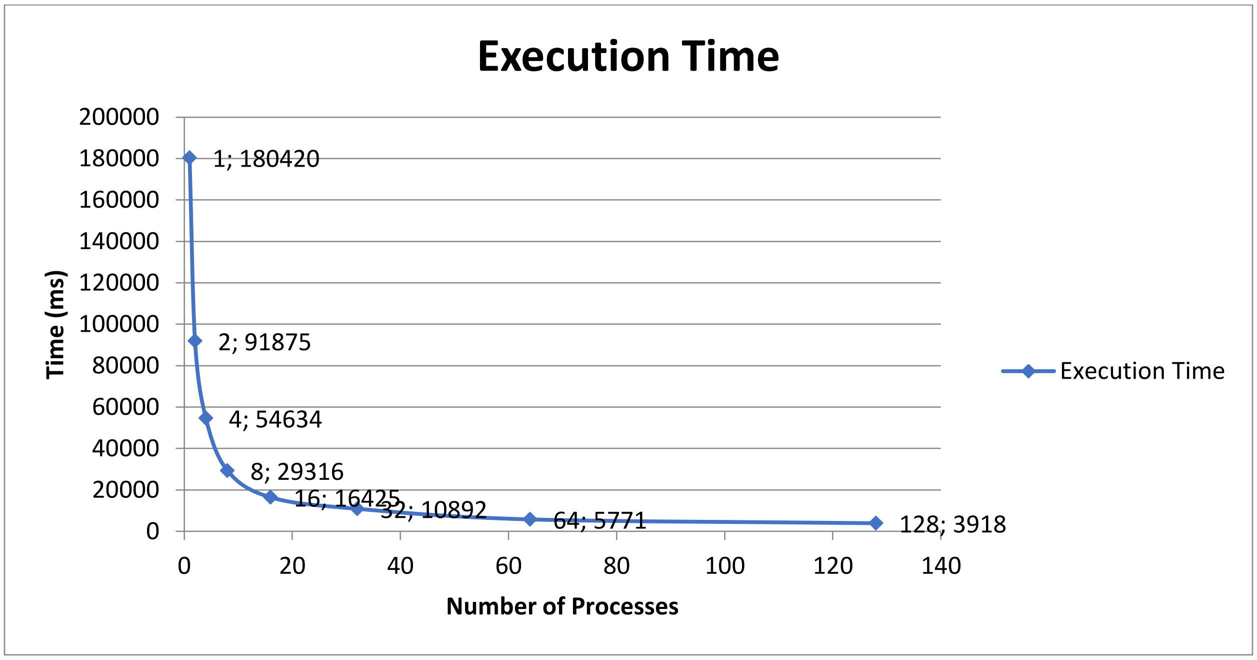
Ex2

Itamar Chuvali

200048734

2.3



2.4

As mentioned in the question we are only running a single core, therefore we cannot expect the time processes to actually run in parallel. The reason we are seeing faster execution time as the number of processes increase is that the real time it takes to return a OK, ERROR, UNKNOWN signal from the URL is the most time consuming operation. During this operation, the process can begin to **wait** for a response from server. While it waits, the other processes can also execute (one by one) and go into wait mode. With a single process, each call to check_url needs to execute sequentially and can only continue to the next URL once it has received a response. This is also the reason why it takes longer than 2 seconds with 128 processes. Though the wait for a response is a significant part of that, the overhead is higher as there is a fork, fopen, read and write system call per process.

2.5

The graph would change a lot, yes. The waiting period would be near non-existent and therefore the increase in processes would essentially increase the time as is would be calling unnecessary system calls. A single process would be able to run the code as it would be a check within the program.

3.

The first fork forks a child. (2 processes)

The “if not a fork” then checks the process to see if it’s a child and forks it (3 processes)

The “if fork” then checks whether the PID is not 0, which it isn’t for either process, we then fork all 3 of the processes. (6 processes)

Within that if, 2 of the processes are parents so they fork (8 processes)

The last fork forks all 8 for a total of **16 forks**

3.2.

If fork succeeds, we have 2 processes, parent and child. The child will enter the “if” statement and the program will exit by the return statement. The parent will then get to the kill statement and send SIGKILL to the child. However if fork fails, the root process will send SIGKILL to every all other processes it has access to.

3.3

The successful “execlp” command will end the child process ensuring that “printf("LINE J");” will never be reached.

3.4

NEW - impossible. Since process made operations before “worker_checker” function call, it cannot be new.

READY - after called “check_url”, the process is waiting for response. When he receives it he is ready.

WAITING - when we are waiting for answer from server.

RUNNING - when process is running in “worker_checker()” function.

TERMINATED - if error occurred while reading to file or writing to file, we exit the program.