Bilkent University

CS342: Operating Systems

Project 1 Experiment Report

Berdan Akyürek

21600904

Section: 1

1. Introduction

After implementing statclient.c, statserver,c, statclient_th.c, statserver_th.c, and Makefile files and testing them, experiments are conducted in order to understand the running time of a thread and child process. For this, a little modification was done in the client code. After this modification, the server code saves current time after receiving a command from the client and compares it to the time when the response was sent. After each command response, the server displays the current time passed after receiving the request.

Since threads and child processes are created in order to process different files, different file numbers should be tried. For this, a simple python script called generate_files.py was implemented. The code for this script is added at the end of the report.

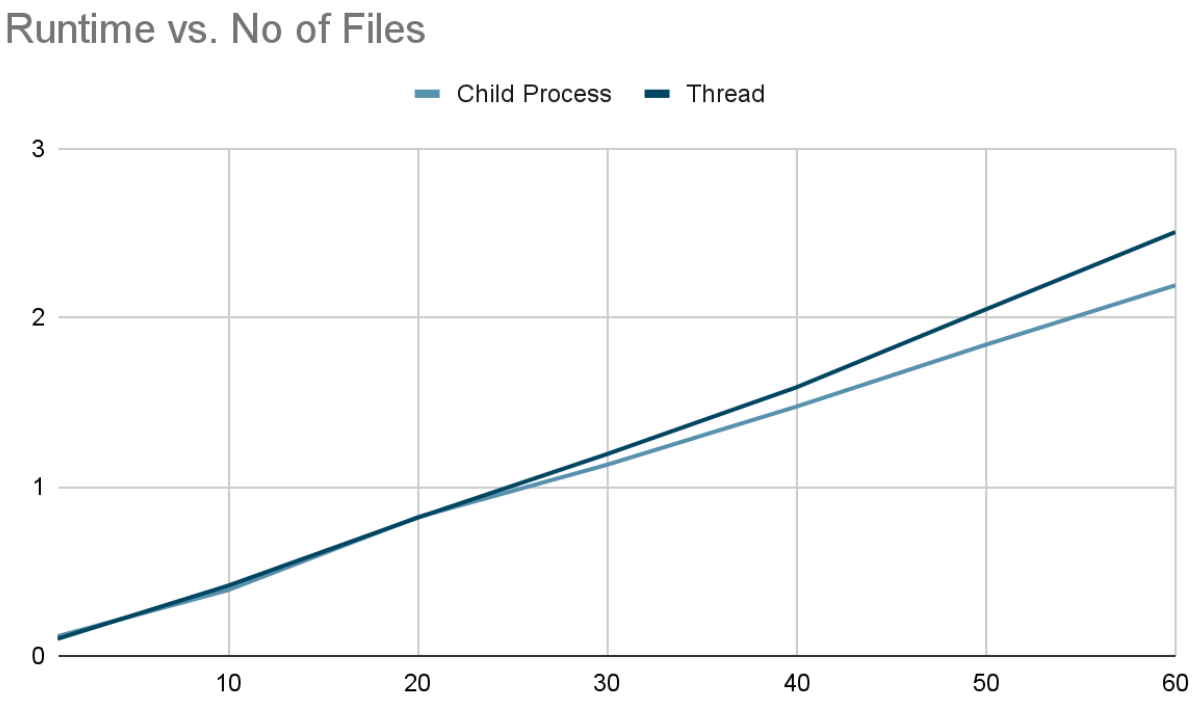
This script takes 2 command line arguments. It is called with a syntax like “python3 generate_files.py x y” where x and y are positive integers. The script removes all “.txt” and “.sh” files in the directory, creates x files where each contains y integers between 0 and 999, creates 2 “.sh” files that run the statserver and statserver_th executables with newly created files. For example, if the script is executed as “python3 generate_files.py 5 100”, at the end of execution there will be 5 files called test0.txt, test1.txt …..test4.txt. Also, there will be a statclient.sh file that contains the command “./statclient_th 5 test0.txt, test1.txt …..test4.txt” and a statclient_th.sh file that contains the command “./statclient 5 test0.txt, test1.txt …..test4.txt”.

In short, after running generate_files.py with two positive arguments, we can run client code with “./statclient.sh” or “./statclient_th.sh”, instead of specifying lots of command-line arguments. This way, it is possible to conduct experiments with large numbers of files without much effort.

2. Experiment Setup

During the experiment process, the max command is executed for 1, 10, 20, 30, 40, 50, and 60 files by keeping the number of integers in a file constant (1.000.000) for both child process and thread implementation. Results are shown on a graph as shown in the next part.

3. Results



As shown in the chart, as the number of files increases, the execution time increases too. However, this increase is faster in threads. This is not as expected. Actually, threads are expected to be faster compared to processes. When the execution time of only thread and process creation, all processes are created in 0.365 seconds but the

same number of threads are created only in 00.1 seconds. This is as expected. Although there is a significant difference between thread and process creation time, with other I/O operations, this difference becomes insignificant.

4. Appendix

1. generate_files.py file content

```
from os import system
import sys
from random import randint

system("rm -rf *.txt *.sh")

no_of_files = int(sys.argv[1])
ints_per_file = int(sys.argv[2])

min = 0
max = 999

for i in range(no_of_files):
   f = open("test%d.txt" % i, "w")
   for j in range(ints_per_file):
       f.write( "%d\n" % randint(min, max))
   f.close()

f = open("statserver.sh", "w")
f.write("./statserver %d " % no_of_files)

for i in range(no_of_files):
   f.write( "test%d.txt " % i )

f.write("\n")
f.close()

f = open("statserver_th.sh", "w")
```

```
f.write("./statserver_th %d " % no_of_files)

for i in range(no_of_files):
   f.write( "test%d.txt " % i )

f.write("\n")
f.close()

system("chmod +x statserver.sh")
system("chmod +x statserver_th.sh")

print("Done")
```