Operating Systems

Pedro Sobral
pmsobral@ufp.edu.pt

Bruno Gomes
bagomes@ufp.edu.pt

João Viana
jviana@ufp.edu.pt

March 2023

Universidade Fernando Pessoa

Faculdade de Ciências e Tecnologias

Objective:

Develop a simplified HTTP server and test clients, using the knowledge acquired in the Operating Systems curriculum unit, in order to improve the performance and scalability of applications.

1. Problem definition

Starting from a reference example made available from an HTTP server and client, it is intended that students implement, in a concurrent programming paradigm, the mechanisms to use more effectively the resources available on the machine. The system performance metric relates to the total response time from the customer's point of view (period elapsed between sending the HTTP request and receiving the response). In this way, the server should be based on a multi-process implementation (Phase 1) or multitasking (Phase 2), while the client should evolve towards becoming a benchmarking tool.

2. Requirements

For the 1st phase of submission, the following requirements will be considered:

A. (20%) Adapt the client code to receive http requests by argument N in batches of size M. The processes of each batches should run concurrently. The program only advances to the next batch after receiving response from all requests already shipped.
    a. ./ client 1000 10 -> 1000 requests, 10 to 10
    b. ./ client 1000 1000 -> 1000 competing requests
    c. ./client 1000 1 -> 1000 sequential requests (serial execution)

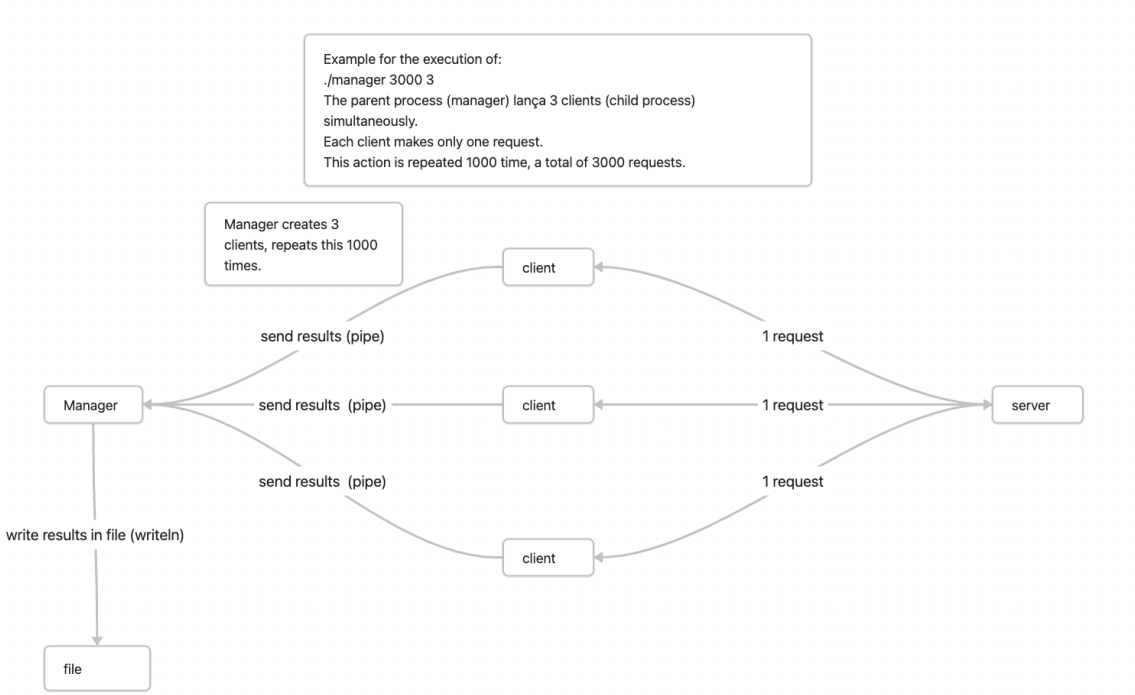
    The program should post M n/m child processes times. The m children must run in a concurrent manner, each of which is responsible for collecting the HTTP response code (rc), and the elapsed time (t) and concatenating with their process identifier

(pid), batch sequence number (bsn), and request sequence number (rsn) in the batch context. Before finishing, each of the child process must write to the same shared file, a line with the execution result in the format:

pid;bsn;rsn;rc;t

B. (10%) Generate order execution report. Total time, average time per order, minimum and maximum times. The result must be printed on the terminal before finishing the client execution

C. (20%) Change requirement A so that client processes, alternatively writing the line to a file shared with the execution result, *communicate pipes with the parent* process (manager), which is responsible for writing to the file. The writing and reading of the pipe should be implemented using the functions *readn*[2] e *writen*[2].



D. (15%) Implement customer *Graceful Shutdown* function. The parent process should install a service routine for the SIGINT signal. When this signal is received, the parent process is responsible for terminating all child processes and presenting the execution statistics (requirement B) collected so far.

E. (35%) Make the HTTP server a multi-process application:

1. Adapt the server code to launch a process to fulfill each request received from clients (creating child processes "*on demand*"). The server is waiting

for a connection from a client. Once you accept a new connection, the parent process launches a new process to fulfill the request. After the response is sent, the child process closes the connection and terminates.

2. An improved version of the requirement involves exploring a solution where the parent process maintains a pool of previously released processes. As soon as a new request is received, the parent process is responsible for forwarding it to one of the available child processes (see *call to waitpid()* system and supported options).
3. To ensure the stability of the solution, the parent process should trigger the creation of new child processes for insertion into the pool whenever they end unexpectedly. The verification of processes that have ended unexpectedly should be implemented using the installation of a signal response routine (SIGCHLD).

3. Notes

Throughout the project, the student must use exclusively calls to the POSIX system (*open, read, write*,...) and not the c libraries functions(*fopen, fread, fwrite*,...). In all system calls, any error conditions must be tested using the perror *() function*.

This assignment will be carried out individually or in groups of two students. The assignmet (README and source code) must be submitted by the date indicated in the elearning system and will be presented and defended by the student on a date to be designated by the teacher. For the calculation of the final note, a weighting of 50% will be assigned to each of the submission phases.

4. Bibliography

[1] *Advanced Programming in the UNIX ® Environment* - Signal - 10.14 sigaction Function

[2] *Advanced Programming in the UNIX ® Environment* - Advanced I/O - 14.7 readn and writen Functions

[3] *Advanced Programming in the UNIX ® Environment* - Network IPC - 16