Faculdade de Engenharia da Universidade do Porto
MIEIC - SOPE
2020/2021

Relatório Mini-Projeto 2
Client

Turma 6 - Grupo 3

Adelaide Santos	up201907487
Bruno Mendes	up201906166
Eunice Amorim	up201904920
Rita Mendes	up201907877

Criação das *threads* cliente

O cliente necessita de fazer vários pedidos ao servidor com o máximo de paralelismo. Desta forma, lança-se uma nova *thread* a cada novo pedido, num intervalo pseudo-aleatório de 10 a 50 ms (usando delays mais baixos, na ordem dos microssegundos, provocaria potencialmente *overflow* de *file descriptors* abertos).
Como se está a tentar simular condições de concorrência, o número de *threads* tende para infinito, não havendo uma estrutura que permita guardar todos os *id's* para no final fazer *join*. A solução encontrada para tal foi a criação de threads como *detached (*atribuindo-lhes a responsabilidade de libertar os próprios recursos aquando da terminação) e terminando a main thread sem terminar imediatamente o processo: com pthread_exit.

Tal colocou outro desafio: o fecho da fifo pública. Nestas condições, não se saberia ao certo em que momento fechá-la: as *threads* "trabalhadoras" vivem por si mesmas, e terminam potencialmente depois de a main thread terminar. Tal resolveu-se com uma uma chamada à função *atexit()* definindo o fecho da fifo pública como última tarefa a ser executada, depois da terminação regular do processo, ou seja, todas as threads terminarem.

Temporizador do tempo de execução

Como argumento para o programa, é necessário definir um tempo em que a execução irá ocorrer. Desta forma, decidiu implementar-se um temporizador que é carregado inicialmente com o tempo de execução, passado como argumento de linha de comandos. Seguidamente, em cada momento em que seja necessário tomar uma decisão sobre a continuidade do programa, basta consultar se o tempo inicialmente usado já se esgotou.

Comunicação entre Servidor e Cliente

A comunicação entre Servidor e Cliente é feita com base em fifos, um pública pela qual os pedidos seguem até ao Servidor e várias privadas (uma por cada *thread* cliente) pela qual o Servidor dá resposta. Para isso, utiliza-se a estrutura Message fornecida.

Por outro lado, decidiu-se incluir no tempo passado por argumento a espera pela criação da fifo pública por parte do Servidor. Ou seja, se no tempo fornecido o Servidor não o conseguir fazer, ocorrerá *timeout***.**

Escrita na FIFO pública

Na sequência da comunicação entre threads do cliente e do servidor, torna-se necessário, de acordo com as especificações, a escrita do pedido numa *FIFO* pública por parte do cliente. Deste modo, inicialmente, pensou-se em encapsular essa escrita usando uma primitiva de sincronização, evitando assim a escrita concorrente. Contudo, após alguma pesquisa, concluiu-se que esta será uma operação atómica. A escrita numa FIFO é atómica sempre que o volume de dados a escrever é inferior a PIPE_BUF. Ora, por POSIX.1-2001, PIPE_BUF é de pelo menos 512 bytes, ou seja, valor superior ao tamanho da mensagem a transmitir[1]. Sendo atómica, não será necessário o uso de primitivas de sincronização.

Registo de operações

O registo de operações é feito com base no esquema pedido através da escrita em stdout com *printf*.

Participação de cada elemento

O esforço de trabalho foi dividido por todos de igual forma.

#teamWork

[1] https://linux.die.net/man/7/pipe
https://www.gnu.org/software/libc/manual/html_node/Pipe-Atomicity.html