U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Sistemas Operativos

MP2: Servidor de aplicações

Turma 3

Grupo 03

Marcelo Henriques Couto	up201906086@fe.up.pt
Miguel Azevedo Lopes	up201704590@fe.up.pt
Pedro Miguel Jesus da Silva	up201907523@fe.up.pt
Sofia Ariana Moutinho Coimbra Germer	up201907461@fe.up.pt

Introdução

Para o desenvolvimento do MP2 foi nos pedido para desenvolver um programa multithread do tipo servidor-cliente. Neste, um processo cliente multithread envia os seus pedidos de tarefas para o servidor através de um FIFO público, ficando em fila de espera até que seja a sua vez, momento em que esse pedido é passado por um thread produtor à biblioteca para ser executado, sendo o resultado colocado num buffer, que é monitorizado por uma thread consumidor que envia os resultados para o cliente através de FIFOs privados.

Na primeira parte do MP2 foi apenas solicitado o desenvolvimento da parte do cliente. Para isto, foi preciso recorrer a multithreading e comunicação entre processos através de FIFOs.

Requisitos Funcionais

O cliente é um programa com multithreading e funciona com paralelismo, tal como foi pedido.

O comando de invocação do cliente encontra-se correto, só sendo possível passar tempos (parâmetro "nsecs") inteiros positivos..

No cliente a thread principal gera continuamente as threads de pedido com intervalos pseudo-aleatórios, cada um ficando associado a um pedido ao servidor como é pretendido. Um FIFO serve como canal público onde são colocados os pedidos. Cada thread gera o seu número de identificação universal, a carga da tarefa pedida (inteiro aleatório entre 1 e 9) e realiza toda a comunicação que precisa com o servidor, criando e eliminando um FIFO privado na qual recebe a resposta do servidor, esperando em bloqueio até lá.

Quando o tempo descrito em "nsecs" acaba, o cliente termina a sua execução, fazendo com que os threads em espera de resposta desistam e sejam libertados todos os recursos usados no sistema. Se porventura o Servidor fechar o FIFO público, o cliente, como pedido, detecta isso, termina de fazer pedidos à biblioteca e as threads com pedidos colocados aguardam ou o resultado do seu pedido, no caso do servidor já ter realizado o pedido antes de ter encerrado, ou indicação de encerramento quando o pedido já não é atendido.

O registo das operações na stdout encontra-se igualmente implementado, ocorrendo este antes de uma thread de pedido terminar.

Detalhes de implementação

Parse - O parse do comando de invocação do cliente é feito no módulo parser.c. Este possui a função parse() de funcionamento muito simples. Numa primeira fase vê se o número de parâmetros está correto, terminando o programa em caso contrário. Numa segunda fase percorre os argumentos procurando a opção "-t" e vendo se existe um inteiro positivo ("nsecs"), terminando o programa se tal não se verificar. O parâmetro restante é, portanto, o "fifoname".

FIFOs - O módulo fifos.c trata da maioria das funcionalidades relacionadas com os FIFOs do programa.

- A função createFIFO() está encarregue da criação dos FIFOs privados.
- A função 'openPublicFIFO' tenta abrir o fifo público (e desiste após um timeout). O público FIFO abre com a opção O_NONBLOCK para não bloquear a escrita caso o FIFO esteja fechado do outro lado.
- A função writeToPublicFifo() escreve um pedido para o FIFO público e faz log na consola do registo de execução com "IWANT".
- A função readFromPrivateFifo() abre o FIFO privado da thread e lê de lá o resultado do seu pedido, registrando a execução com "GAVUP", "CLOSD" ou "GOTRS" nos momentos devidos.
- A função forcePipesClosure() fecha todos os FIFOs privados com 'file descriptors' diferentes 0, 1 ou 2 e é usada quando o tempo do programa acaba, de modo a fechar as threads que ainda não obtiveram o resultado do seu pedido. As threads que obtiveram resposta já fizeram o registo de execução antes da chamada da função dado que não são afetadas.

Threads e Main - O ficheiro principal main.c possui as funções:

- threadHandler(), cuja função pode ser inferida através do nome (responsável pela atividade de cada thread). Esta cria um FIFO privado (createFifo()) e a message struct com a carga de pedido aleatória de 1 a 9 (createMessageStruct()), escreve o seu pedido (writeToPublicFifo()) e depois lê o resultado no FIFO privado (readFromPrivateFifo())
- handleRequests(), que cria as threads de pedido em loop (o código em threadHandler() é corrido) e invoca-se o usleep() com um valor de tempo aleatório, terminando o loop quando o tempo descrito em "nsecs" acaba. A main() é, portanto, responsável por invocar a função 'parse()', abrir o FIFO público, invocar a função de criação de threads e finalmente libertar os restantes recursos do programa.

Utils - O módulo utils apenas contém apenas funções menores e de utilidade variada, como simplificação dos registos de execução criação da struct da mensagem (bem como a definição da struct).

Linked List - Decidimos que a utilização de uma linked list seria a melhor opção para armazenar os ids dos threads de pedido criados, visto que é difícil prever a quantidade de memória necessária a alocar. Para isso, criámos as funções e estruturas necessárias à implementação de uma.

Notas Extra - O nosso programa não implementa qualquer ferramenta de coordenação dos threads por falta de necessidade. Poucas são as situações em que as threads utilizam uma variável partilhada. Nas situações em que isto acontece, não existem riscos nem race conditions, pois ora estes acessos são apenas para leitura ora a escrita simultânea não apresenta nenhum risco para o programa, como é o caso da 'serverFlag'.

.

Autoavaliação

Marcelo Couto	25%
Miguel Lopes	25%
Sofia Germer	25%
Pedro Silva	25%