U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Sistemas Operativos

MP2-v3: Servidor de Aplicações

Marcelo Henriques Couto	up201906086@edu.fe.up.pt
Miguel Azevedo Lopes	up201704590@edu.fe.up.pt
Pedro Miguel Jesus da Silva	up201907523@edu.fe.up.pt
Sofia Ariana Moutinho Coimbra Germer	up201907461@edu.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 segunda parte do MP2 foi solicitado o desenvolvimento da parte do servidor. Para isto, foi preciso recorrer a multithreading e comunicação entre processos através de FIFOs e evitar conflitos entre entidades concorrentes, por via de mecanismos de sincronização.

Requisitos Funcionais

O Servidor S é invocado com o comando: s <-t nsecs> [-l bufsz] em que nsecs é nº (aproximado) de segundos que o programa deve funcionar, bufsz é o tamanho do armazém (buffer) a ser utilizado para guardar os resultados dos pedidos (opcional) e o fifoname é o nome (absoluto ou relativo) do canal público de comunicação com nome (FIFO) por onde serão recebidos pedidos de execução de tarefas. O seu funcionamento, de forma sumarizada, é o seguinte:

- inicialmente, recolhe os argumentos de linha de comando, valida-os e executa as operações fundamentais para o funcionamento do programa. A ordem com que são passados os parametros não é importante, pode ser feita de qualquer maneira;
- os pedido do Cliente são recolhidos pelo thread principal s0 que cria threads Produtores s1, ..., sn e lhes passa os pedidos; cada Produtor invoca a correspondente tarefa em B e coloca o resultado obtido no armazém; depois, termina;
- o acesso ao armazém é gerido usando semáforos, sendo que os threads Consumidor e Produtores ficar bloqueados quando o armazém estiver, respectivamente, vazio ou cheio;
- o (único) thread Consumidor tenta retirar continuamente valores do armazém pela ordem de inserção e enviar cada um ao correspondente thread Cliente que fez o pedido, através do respectivo FIFO privado;
- mesmo não havendo pedidos do Cliente (estando o canal público vazio), o Servidor continua a executar, ficando à espera da chegada de algum pedido, até ao final do prazo de execução especificado (em argumento da linha de comando);
- no final, garante-se que todos os recursos tomados ao sistema são libertados.

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") e o mesmo para "-l", terminando o programa se tal não se verificar.

Threads - Tal como requisitado existem 2 threads que se mantêm ativas ao longo do funcionamento do programa, a que faz a leitura do public fifo e que gera novas threads e a thread consumidora que consome os pedidos do buffer. Para além disso existem as threads produtoras que chamam a biblioteca para executar os pedidos do cliente e depois os colocam no buffer. A gestão do acesso ao buffer é feita usando semáforos que limitam o número de threads que escrevem para o buffer e que impedem que a thread consumidora consuma mais do que os pedidos existentes no buffer.

Buffer - Está implementado sob a forma de vetor. Para gerir em que índice do vetor cada thread deve escrever foi implementado um mutex. Caso o utilizador não especifique o tamanho do buffer ao executar o servidor, foi implementado um valor default de 10.

Linked List - como na fase anterior, os ids dos thread são guardados numa linkedlist, de forma a facilitar o processo de alocação de memória requerida por estes e posterior libertação da memória alocada aos threads.

Timeout - quando o tempo designado para o programa chega ao fim, este ainda processará os últimos pedidos para os quais já tinham sido gerados threads. Aqueles que não tinham sido ainda processados, não o serão, sendo feito o registo de 2LATE, juntamente com o envio da mensagem inicial intacta. Para saber quantos pedidos faltam processar após timeout, é feito uso a uma variável partilhada 'activeRequests', cujo controlo é feito através de um mutex.

Instruções de compilação

Para compilar, basta utilizar o makefile que incluímos na nossa submissão. O resultado da compilação será o servidor, na forma de um ficheiro com o nome "s".

Autoavaliação

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