U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Client Server App

Mini-Projeto 2B

Sistemas Operativos 2020/2021

Turma 7 Grupo 1

Adriano Soares	**up201904873**@edu.fe.up.pt
Diogo Maia	**up201904974**@edu.fe.up.pt
Francisco Cerqueira	**up201905337**@edu.fe.up.pt
Pedro Pereira	**up201905508**@edu.fe.up.pt
Vasco Alves	**up201808031**@edu.fe.up.pt

Índice

Contexto

Requisitos Funcionais

A aplicação relativa ao servidor, requerida nesta segunda entrega, é capaz de recepcionar diversas tarefas de diferente carga em paralelo do cliente, comunicando-as à biblioteca disponibilizada, e comunicando o resultado, através da criação de threads produtores, que irão lidar com a comunicação com a biblioteca, e threads consumidores, que irão comunicar o resultado ao cliente.

Esta recebe o número de segundos que o programa deve permanecer em execução, o nome (absoluto ou relativo) do canal público de comunicação com nome (FIFO) por onde o Cliente envia pedidos ao Servidor e, opcionalmente, o tamanho do buffer que guarda os pedidos já respondidos.

O formato requerido será então:

s <-t NSECS> [-l BUFFERSIZE] <FIFONAME>

1) NSECS: número de segundos que o programa deve ficar ativo;
2) BUFFERSIZE: tamanho do buffer que armazena os pedidos prontos a serem processados pelo thread consumidor;
3) FIFONAME: caminho relativo ou absoluto do canal público de comunicação com nome (FIFO) por onde o Cliente envia pedidos ao Servidor.

Estrutura Geral do Código

Main Thread

A thread principal (*main()*) é responsável por inicializar todas as variáveis necessárias para o processo, como por exemplo o descritor do FIFO público, o estado do servidor, o buffer, a mutex e verificar também os argumentos passados pelo utilizador. Esta substitui o handler de Alarme e aciona-o de modo a medir o tempo de execução do programa. Após esta inicialização, procede-se à criação da thread consumidor.

De seguida, é verificada periodicamente a escrita de mensagens por parte do cliente na FIFO pública. Em caso afirmativo, é criada uma thread produtora para cada mensagem.

Após o encerramento do servidor, esta espera por todas as threads produtoras e thread consumidoras para impedir perdas de memória e finaliza o trabalho do servidor como um todo libertando o espaço de memória previamente alocado para as variáveis necessárias.

Producer Thread

A thread produtora é responsável por invocar as tarefas na Biblioteca, escrever na mensagem o resultado devolvido pela execução da tarefa e armazená-la no buffer. O resultado escrito na mensagem depende da Biblioteca e se o servidor está ou não aberto. Se o servidor estiver aberto, este pede o resultado à Biblioteca de acordo com a tarefa do pedido, senão, este apenas envia o pedido como está para o buffer (com resultado -1).

Para sincronizar a gestão de dados entre as threads produtoras e o buffer, foi utilizado uma mutex. Quando este pretende escrever no buffer, ele dá *lock* à mutex e tenta escrever. Caso o buffer esteja cheio, ele espera até poder escrever, senão escreve e dá *unlock* à mutex, terminando a thread.

Consumer Thread

A thread consumidora está encarregue de enviar as respostas do servidor ao cliente, através de uma FIFO privada criada pelo cliente. Com a implementação de uma fila foi nos permitido organizar as respostas do servidor de uma forma simples e eficaz. Desta forma, podemos enviar as respostas por ordem de obtenção de resposta da Biblioteca. Após receber a mensagem com a resposta do buffer, esta thread está encarregue de determinar se pode enviar a resposta e registar a operação de acordo com o sucedido.

A segunda função desta thread é abrir o descritor de escrita da FIFO privada do cliente. Através desta abertura e da sucessiva tentativa de escrita na FIFO, o servidor consegue determinar se o cliente ainda está a receber as respostas e, caso não consiga, o pedido é registado como FAILD.

Caso a escrita seja bem sucedida, o pedido pode ser registado de duas maneiras: se no pedido a resposta for um valor diferente de -1, o thread produtor conseguiu responder ao pedido do cliente e este é registado como TSKDN; caso este seja igual a -1, o thread produtor não processou a resposta e o pedido é registado como 2LATE.

Esta thread funciona até que todos os pedidos que passaram pela thread produtora sejam registados, quer o FIFO público do servidor esteja aberto ou não.

Utils

Para além das threads previamente referidas, existem duas estruturas que foram importantes para o desenvolvimento do produto:

- A *queue*, que nos permitiu manipular de uma maneira simplificada o buffer que armazena as mensagens atendidas.
- A *message*, que foi fornecida previamente pelo professor.

Principais Dificuldades

A principal dificuldade que encontramos foi na abertura e leitura/escrita nos FIFOS públicos e privados, respetivamente.

Ao tentarmos abrir o FIFO público sem a flag O_NONBLOCK ativa o programa ficava bloqueado à espera da abertura deste FIFO por parte do cliente, e caso isso não acontecesse o alarmHandler era ativado, e quando o programa retomava ao estado que se encontrava permanecia bloqueado na abertura deste. De modo a contornar esta adversidade decidimos ativar a flag O_NONBLOCK e utilizamos a função poll() com a flag POLLIN ativa (com delay imediato) permitindo-nos saber quando o cliente insere informação para o FIFO. Neste caso o read() retorna um inteiro positivo quando existe informação para ler, e -1 quando não há informação para lermos, com *errno* = EWOULDBLOCK. Quando o alarmHandler é ativo, fechamos o FIFO público, o read() retorna 0 simbolizando EOF (o fechar do FIFO por um dos lados), não permitindo a leitura de mais solicitações.

O mesmo acontecia com a thread consumidora, esta ficava bloqueada quando tentávamos abrir um FIFO privado quando o cliente não estava aberto. Deste modo acionamos a flag O_NONBLOCK, em caso de erro na escrita e na abertura (o cliente não está ativo) o pedido é registado como FAILD.

Contribuição Percentual

Nome	**Percentagem**
Adriano	20%
Diogo	20%
Francisco	20%
Pedro	20%
Vasco	20%