README.md

Aluno: Gustavo de Castro Biage (18200424)
Disciplina: Computação Distribuída
Semestre: 20202

> Refererência: Opção de socket SO_REUSEPORT

Instruções de execução

Para a implementação cliente-servidor, pode-se testar a funcionalidade da aplicação de maneira manual ou de maneira automática, assim, a maneira manual apresentada fará uso do executável `calculadora` e `servidor`, enquanto a execução de testes mais automáticos fará uso de `cliente`, `multi_cliente` e `servidor`.

Para o funcionamento do servidor e das requisições, foi escolhida uma porta arbitrária, assim, utiliza-se a porta 8089, que pode ser modificada nos arquivos `src/servidor.c` e `template/cliente.h` se necessário.

Execução manual

A execução manual será bem similar ao proposto no enunciado da atividade, assim, necessita-se da abertura do servidor antes de executar a calculadora. Recomenda-se a execução do servidor em um terminal e da aplicação em outro. Assim, primeiro demonstra-se a compilação e execução do servidor.

```
foo@bar:~/.../src$ make servidor
foo@bar:~/.../src$ ./servidor
```

Após iniciar o servidor, pode-se compilar e executar a aplicação, permitindo a realização de operações. Para finalizar o funcionamento, deve-se escrever *"sair"* como entrada, deste modo, como exemplo demonstra-se a solução para as expressões *1+2*, *3*3* e *9+4*.

```
foo@bar:~/.../src$ make calculadora
foo@bar:~/.../src$ ./calculadora
1+2
r: 3
3*3
r: 9
9+4
r: 13
sair
```

Execução automática

Na execução automática, o `cliente` realizará uma série de requisições para solucionar uma expressão passada como argumento, assim, como o objetivo é de somente demonstrar e testar a capacidade de atender várias requisições de maneira simultâneas, as expressões serão resolvidas da esquerda para a direita, sem levar em consideração a prioridade de operadores.

Portanto, como exemplo mostramos o comportamento da expressão *1+2*3+4*, onde será resolvida a primeira requisição *1+2*, com retorno *3*; seguido da requisição *3*3*, com retorno *9*; e por fim, haverá a requisição *9+4*, retornando 13.

Dessa forma, o executável `multi-cliente` simplesmente inicializa-se várias instâncias de `cliente`, logo, na sua execução, será criada um número de *threads*, com base na quantidade de argumentos fornecidos, cada *thread* fará uma chamada de sistema para um novo processo de `cliente` e esperará o fim de sua execução. Com base nessas informações, no `makefile`, encontra-se um conjunto de comandos para testar a solução das expressões *1+2*3+4*, *2+3-9/2* e *4+5-5+4*1*.

Abaixo apresenta-se uma versão mais legível dos comandos executados pela regra `testar` do `makefile`. O primeiro comando executa o servidor em background e armazena-se o `PID` de seu processo no arquivo `server.pid`. No segundo comando, executa-se o `multi_cliente` com as três expressões mencionadas anteriormente e espera-se o fim de sua execução. No último comando, termina-se o processo com o `PID` armazenado em `server.pid`, que seria o processo do servidor.

```
testar: servidor multi_cliente
        @./servidor & echo $$! > resources/tmp/server.pid ;
        @./multi_cliente 1+2*3+4 2+3-9/2 4+5-5+4*1
        @cat resources/tmp/server.pid | xargs kill
```

Podemos executar esta regra do `makefile` com o comando `make testar`. Então, espera-se os seguintes valores apresentados de resposta da requisição. Claro que os mesmos podem estar em uma ordem diferente, pois foram requisitados em paralelo, porém, todos estes resultados devem estar presentes.

```
foo@bar:~/.../src$ make testar
Server waiting...
resultado = -2
resultado = 8
resultado = 13
finalizando servidor
```

A mensagem *"Server waiting..."* será escrita pelo `servidor`, assim que ser ligado. As mensagens com os resultados das expressões serão escritas cada um pelo seu `cliente` independente. A mensagem *"finalizando servidor"* será colocada no terminal pelo próprio `makefile`, informando que o processo do `servidor` será terminado forçadamente.

Dentro do arquivo `src/servidor.c`, conseguimos descomentar as linhas `printf("THREAD START\n")` e `printf("THREAD END\n")`. Por consequência, sempre que o servidor aceitar uma nova conexão e criar uma nova *thread* para supri-la, será impresso uma mensagem de início e fim. Logo, pode-se enxergar no terminal a resolução em paralelo das requisições. Como apresentado abaixo:

```
foo@bar:~/.../src$ make testar
Server waiting...
THREAD START
THREAD START
THREAD START
THREAD END
THREAD END
resultado = -2
resultado = 8
THREAD END
resultado = 13
finalizando servidor
```

Um dos problemas encontrados por este método de teste é que, mesmo depois de o processo ser terminado, o `socket` utilizado pelo servidor continua ocupado. Deste modo, após realizar um novo teste logo em seguida, haverá problema para se estabelecer novas conexões. Portanto, foi adicionado o `SO_REUSEPORT` como opção de `socket`, que permite que múltiplas aplicações ouçam a mesma porta. Assim, a realização de testes subsequentes não resultam em problemas de conexão. Dessa forma, a utilização desta opção pode não ser a melhor escolha em uma aplicação real, visto que fere a segurança e qualquer outra aplicação pode passar a ouvir a mesma porta do servidor em funcionamento. Contudo, para o âmbito de apresentar o sistema em operação, esta opção foi utilizada e não gera nenhum risco crítico.