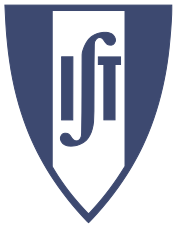

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

PSis 2011/2012

MEEC - MEAer

Prof. Luís Silveira

Yet **A**nother **S**imple **C**alculator

14 de Maio de 2012

64956 Bruno Santos *bruno.dos.santos@ist.utl.pt*
74570 Flávio Schuindt *flavio.silva@ist.utl.pt*

Conteúdo

1 Introdução **1**

2 Cliente **3**

2.1 Chamada 3

2.2 Interface 3

2.3 Estrutura lógica 5

2.4 Meios de comunicação 5

2.5 Desenvolvimento futuro 5

3 Servidor **6**

4 Bibliografia **7**

1 Introdução

Este relatório destina-se a cobrir a implementação de uma calculadora numa estrutura de cliente - servidor implementado segundo as normas ANSI C99 e POSIX.

Na nossa abordagem, optámos por uma implementação monolítica de *IO* síncrono para o cliente, e uma estrutura altamente paralelizada e completamente assíncrona para o servidor. Estas opções serão discutidas ao longo deste relatório, assim como algumas funcionalidades previstas, mas que não chegámos a implementar.

O desenvolvimento deste projecto foi constantemente registado num repositório *git* online que pode ser consultado em **********bitbucket.org.

2 Cliente

2.1 Chamada

A invocação do cliente segue os requisitos do enunciado, bastando indicar o endereço do servidor e a porta de escuta. A falha em passar dois argumentos corresponde a uma falha directa, mas a correcção destes parâmetros não é feita senão mais tarde quando e se o utilizador decidir iniciar sessão.

```
yascC.exe <hostname> <port>
```

Opcionalmente o utilizador poderá ainda fornecer um ficheiro de texto com comandos a executar cuja sintaxe deverá ser idêntica à usada na linha de comandos. Para esse efeito deverá ser dado o directório do programa como parâmetro extra, precedido de `-f`. A verificação da extensão de ficheiro de texto não é verificada, pelo que o ficheiro usado não tem de ser necessariamente `.txt`.

```
-f <filename.txt>
```

Ainda no processo de chamada o utilizador pode configurar duas flags adicionais: `-l` e `-g`.

A primeira destina-se a suprimir o output da linha de comandos e registá-lo na totalidade num ficheiro `log.txt`, excepto erros críticos que continuam a ser dirigidos para o terminal. Esta flag não pode ser alterada em qualquer outro ponto da execução do cliente e destina-se apenas a permitir uma melhor análise do output do programa, sobretudo em situações em que um ficheiro de comandos é usado.

A segunda flag permite apenas que o modo *Debug* seja iniciado no arranque, mas o utilizador pode em qualquer momento alternar entre os dois modos de output.

Com a excepção do endereço do servidor e da porta de escuta, nenhum destes parâmetros é obrigatório nem tem ordem fixa.

A execução do cliente num directório que não contenha `/doc` não permite o acesso ao ficheiro de ajuda durante a execução.

2.2 Interface

A interface com o cliente é feita à custa dos comandos:

- Controlo de sessão:
  - I
  - K
- Operações em stack:
  - $+$, $-$, $*$, $/$, $\%$

- ◇ Controlo de stack:
  - · P
  - · T
  - · R
- ◇ Outros:
  - · G
  - · HELP
  - · EXIT

Os comandos `I` e `K` permitem iniciar e terminar, respectivamente, a ligação ao servidor. Esta operação pode ser feita em qualquer ponto ponto da execução do cliente, e depois de terminada a sessão o cliente pode voltar a iniciá-la sob a penalidade de ficar bloqueado na fila do `listen()`. No caso de uma sessão ter sido previamente iniciada, `I` deverá limpar o stack com retorno positivo `V0`.

As operações aritméticas, todas elas efectuadas sobre números inteiros de 32 bits, retornam sempre `V0` excepto em caso de overflow, underflow ou divisão por 0.

Para visualizar e controlar o stack, tem-se ainda os comandos `P`, `T` e `R`. Estes permitem saber o tamanho do stack, resultados parciais ou números acabados de introduzir e resultados totais, respectivamente. O modo mais simples de reinicializá-lo é, como foi dito, com o comando `I`.

A introdução de valores no stack é feita simplesmente introduzindo números inteiros em base decimal ou, em alternativa, em base octal na forma *0###* ou na base hexadecimal na forma *0x###*.

Por fim, os restantes comandos destinam-se a ligar e desligar o modo *Debug*, a abrir um ficheiro na própria linha de comandos, e a fechar o cliente. Para a leitura do ficheiro de ajuda, tem ainda de ser tido em conta o directório de execução, conforme explicado na secção 2.1, e vale referir que `EXIT` chama internamente `K` por forma a avisar o servidor de que fechou sessão.

O output gerado é sempre precedido de sinalética própria do tipo de mensagem em questão:

◇ `>>` Erro que quebra funcionalidade do cliente.
◇ `: :` Output normal que inclui algumas mensagens de erro.
◇ `DEBUG:` Output extra do modo *Debug*.

Num erro crítico o cliente aborta com a mensagem de erro, mas em qualquer outro caso o cliente deverá apenas informar do erro e continuar a executar.

Para maior detalhe relativo à interface com o utilizador, pode ser consultado o manual do cliente.

2.3 Estrutura lógica

O cliente é constituído por um único fio de execução durante todo o seu tempo de execução. Permanece num ciclo quási infinito conforme é mostrado no trecho de código seguinte.

De notar que no código abaixo, `fin` identifica a linha de comandos ou o ficheiro de comandos passado ao programa.

```
while( fgets(line,MAX_LINE,fin) != NULL ) {
    parse_line(line);
}
```

Neste ciclo o programa limita-se a analisar o input e a tratá-lo de forma serializada por ordem de introdução. Tal inclui a apresentação de mensagens relevantes que obtenha do servidor.

Este tratamento em série do input leva a que numa situação de bloqueio à espera do servidor se reflicta num cliente inutilizado.

2.4 Meios de comunicação

O cliente faz uso de um único socket para comunicação com outro processo, potencialmente remoto: o servidor. Usa o protocolo IPv4 por facilidade de teste na máquina de desenvolvimento, e todas as operações executadas sobre esse socket são síncronas.

Esta abordagem facilita a interface com o utilizador, sendo mais fácil gerir qual a resposta de cada comando individual. Uma abordagem assíncrona exigiria um buffer de mensagens trocadas entre cliente e utilizador e um maior esforço na sua apresentação.

Por outro lado, a notação polonesa inversa sobressai sobretudo pela sua fácil interpretação síncrona, pelo que esta implementação adequa-se bem aos requisitos do projecto.

2.5 Desenvolvimento futuro

3 Servidor

4 Bibliografia