Instituto Tecnológico de Aeronáutica

Departamento de Engenharia de Software
Divisão de Ciência da Computação
Laboratório de CES-35

Relatório da atividade

Laboratório 3:
Juíz Online

Alunos:
Lucas França de Oliveira, lucas.fra.oli18@gmail.com
Matheus Felipe SBF Rodrigues, matheus.felipesbf@gmail.com
Professora: Cecília de Azevedo Castro César, cecilia@ita.br

2 de outubro de 2017

1 Introdução

Este trabalho tem como objetivo da criação de um juíz online pra ficar a serviço dos professores do ITA. Um juíz online consiste em um sistema online de correção automatizado com feedback instantâneo aos alunos:

1. O professor lança um laboratório e padroniza a formatação de entrada e saída do programa. Desta forma, pode-se criar um sistema automatizado de correção com casos de testes na máquina do professor ou em algum servidor do ITA.

2. O aluno submete o código fonte do um laboratório. O código fonte é compilado, executado com os casos de teste do professor e sua saída é comparada com as saídas corretas.

3. O juíz, ao final da correção, retorna um pequeno relatório gerado automaticamente pelo corretor contendo a porcentagem de casos de teste que tiveram os possíveis vereditos (Aceito, Resposta Errada, Tempo Limite Excedido, Erro em Tempo de Execução, Erro de Compilação).

O aluno não terá, assim, acesso aos casos de teste. Ao receber um veredito inesperado de Resposta Errada, por exemplo, o aluno poderá alterar o código e procurar por erros.

Isso será particularmente útil nas disciplinas de CES-10 e CES-11, em que a a entrada e saída dos laboratórios pode ser padronizada e os problemas abordados possuem solução conhecida. Essas matérias, também, são aplicadas no Curso Fundamental, abrangendo todos os alunos do ano, e possuem uma grande quantidade de laboratórios. A correção automatizada facilitará o trabalho dos professores.

2 Desgin do Juíz

2.1 Comunicação

Foi usada comunicação via protocolo TCP diretamente entre o cliente e o servidor. Esse protocolo foi escolhido devido à sua simplicidade e à existência de blibiotecas na linguagem Java que facilitam o trabalho. O protocolo UDP, que também possui as mesmas facilidades, é menos confiável e é usado somente quando há a necessidade de transmissão rápida de mensagens. Exemplo: em jogos online em tempo real.

O protocolo TCP permite o envio de um buffer de bytes. Essa sequência representa uma string, que será usada para a representação da mensagem em forma de texto. Isso permite certa permissividade com erros, pois caso um byte seja corrompido, a palavra ainda pode ser recuperada pelas outras letras. Como os comandos são pequenos, o tamanho da mensagem continua baixo. A exceção será o código em si, que será enviado diretamente como string no buffer do TCP.

Os comandos são separados por linha, ou seja, pelo caractere $\backslash n$. A primeira palavra, que terá tamanho fixo de 6 caracteres, será um identificador de comando: cada pacote enviado segundo descrito nas seções abaixo será identificado por um identificador de comando único. O resto dos bytes será a string com a informação daquele pacote. Exemplo: o código fonte ou a porta para o envio de dados.

Os comandos podem ser os seguintes:

- *SubReq*: sinaliza o início de uma submissão.

- *SubAck* + porta: acusa o recebimento do requerimento e envia a porta de transferência de dados.

- $SubFin$: sinaliza a confirmação da submissão.
- $CodReq$ + id do problema + número de linhas + código: transfere o código e informações a respeito dele.
- $CodAck$ + id da submissão: acusa o recebimento do código e envia o identificador de submissão.
- $RemReq$: envia o lembrete de correção pronta.
- $RemAck$: acusa o recebimento do lembrete.
- $RemFin$ + veredito: retorna o resultado.
- $RpoReq$ + id da submissão: solicita o resultado de uma submissão e envia o identificador dela.
- $RpoAck$ + veredito: acusa o recebimento do requerimento e o resultado.
- $RpoFin$: acusa o recebimento do resultado.

Cada troca de mensagem possui um timeout de 5 segundos. A correção possui um timeout de 60 segundos. Cada corretor funciona em uma thread distinta da do servidor, o que permite que o juíz atenda a várias submissões simultaneamente. Isso permite que uma exceção ou um loop infinito não derrubem o servidor.

Foi implemetado um único executável aue recebe como argumentos de linha de comando uma de 3 opções:

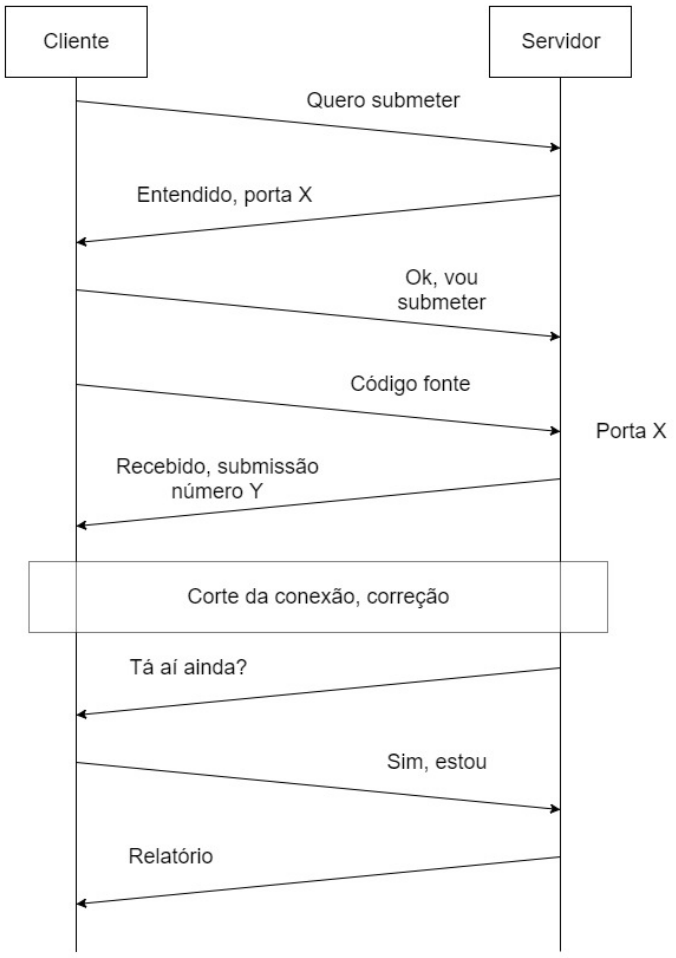
- $server < ip > < port > < addr >$: inicializa um servidor no o IP e porta fornecidos com o banco de dados na máquina no endereço addr.
- $submit < ip > < port > < path > < id >$: submete um código no arquivo descrito por path, solução do problema id, servidor em ip e porta port.
- $report < ip > < port > < sid >$: recupera o veredito da submissão de identificador sid de um servidor no endereço ip e porta port.

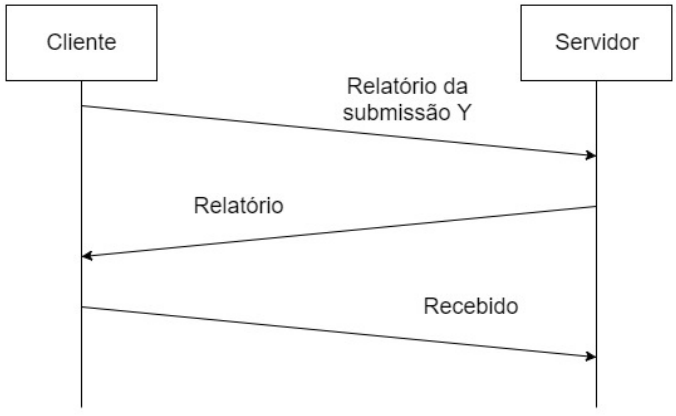
O processo submit é semelhante ao do protocolo FTP no modo passivo. O servidor reserva um porta dedicada ao estabelecimento da conexão. O cliente se conectará com essa porta e enviará um pacote solicitando a submissão. O servidor responderá com uma nova porta para o fluxo de dados. O cliente encerra a comunicação com uma confirmação, concretizando-se o algoritmo dos "3 handshakes".

A seguir, o cliente iniciará a conexão com o servidor na porta especificada e fará o envio do código fonte. No final, o servidor responderá que recebeu e encerrará a conexão para fazer a correção. Após a correção, o servidor tentará se conectar com o cliente e, também fazendo os "3 handshakes", enviará o relatório. Nesta última etapa, os 3 envios servem para verificar a presença do cliente sem a necessidade de enviar o veredito. Neste exemplo, o relatório é apenas uma sentença, mas em uma possível aplicação no ITA, poderia ser um arquivo PDF com um relatório gerado automaticamente.

O comando report fará apenas uma iteração dos "3 handshakes", o que é suficiente para o recebimento do relatório.

Na proposta, foi enviado o esquema abaixo para se ilustrar o procedimento. Apenas uma mudança foi feita: não há necessidade de se fazer o corte da conexão durante a correção. Caso ela seja mantida durante a correção, o relatório será enviado automaticamente. Caso ela seja interrompida, o envio final falhará, mas o veredito ainda estará disponível através do identificador de submissão enviado inicialmente.


Cliente
Servidor
Quero submeter
Entendido, porta X
Ok, vou submeter
Código fonte
Porta X
Recebido, submissão número Y
Corte da conexão, correção
Tá aí ainda?
Sim, estou
Relatório


Cliente
Servidor
Relatório da submissão Y
Relatório
Recebido

2.2 Compilação e correção

Este laboratório tem foco na comunicação via TCP, então simplificou-se o máximo possível o funcionamento da correção. Serão admitidos apenas códigos em um único arquivo em C++. A compilação ocorrerá fazendo uma chamada ao GNU g++ 6.3.

É feita uma execução para cada teste usando o standart input e standart output, e a resposta será comparada com um gabarito. Para termos de simplicidade, o problema de teste consiste ler dois números inteiros separados por um espaço e imprimir sua soma. Os números são menores que 100000.

Foram feitos testes de erros em tempo de execução, tempo limite excedido, erro de compilação, resposta errada e resposta certa. As chamadas, tanto de compilação quanto de execução, acontecem com a API de Runtime do Java, que permite a execução de um comando de terminal.

3 Testes e resultados

O código foi implementado em Java utilizando a IDE NetBeans. O IP usado foi o localhost e uma porta maior que 2000. A seguir, segue a saída de execução de uma sequência de submissões, fizeram-se 6 submissões, a 3 e 4 com erros de compilação, apenas a primeira sendo mostrada. Todas as submissões foram feitas em uma única máquina com o IP localhost.

Não foi possível implementar o tratamento de loop infinito, que deveria retornar o veredito "Time Limit Exceeded", devido a complicações de implementação. Uma submissão com loop infinito travará a thread de correção, vide o exemplo abaixo na submissão de índice 0. Foi possível, porém, fazer com que o servidor não caísse neste caso, pois cada corretor executa em uma thread distinta. Processos que encerram excedendo o timeout recebem o veredito normalmente.

O veredito "Runtime Error" é acionado quando a standart error retorna não vazia ou quando o valor de retorno do processo é diferente de 0 (código de erro nulo). No entanto, nos casos de erro em tempo de execução do código do aluno, o Sistema Operacional demora pra tratar o processo e o timeout é estourado, fazendo a submissão receber o veredito de tempo limite excedido, como se pode ver no exemplo abaixo na submissão de índice 5.

Submissão de um código com loop infinito:

```
[SUBMITTER] Code read (11 lines):
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    while(true);
    printf("%d\n", a+b);
    return 0;
}

[SUBMITTER] Connecting request submitter...
[SUBMITTER] Attempting submit request
[SUBMITTER] Received: SubAck
[SUBMITTER] Received: SubAck 1679
[SUBMITTER] Connecting code submitter...
[SUBMITTER] Received: CodAck
[SUBMITTER] Submission id: 0
```

Submissão de um código correto:

```
[SUBMITTER] Code read (10 lines):
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d\n", a+b);
    return 0;
}

[SUBMITTER] Connecting request submitter...
[SUBMITTER] Attempting submit request
[SUBMITTER] Received: SubAck
[SUBMITTER] Received: SubAck 9275
[SUBMITTER] Connecting code submitter...
[SUBMITTER] Received: CodAck
[SUBMITTER] Submission id: 1
[SUBMITTER] Received: RemReq
[SUBMITTER] Received: RemFin
[SUBMITTER] Submitter done, verdict: ACCEPTED
```

Submissão de um código incorreto:

```
[SUBMITTER] Code read (10 lines):
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d\n", a+b+1);
    return 0;
}

[SUBMITTER] Connecting request submitter...
[SUBMITTER] Attempting submit request
[SUBMITTER] Received: SubAck
[SUBMITTER] Received: SubAck 4722
[SUBMITTER] Connecting code submitter...
[SUBMITTER] Received: CodAck
[SUBMITTER] Submission id: 2
[SUBMITTER] Received: RemReq
[SUBMITTER] Received: RemFin
[SUBMITTER] Submitter done, verdict: WRONG ANSWER
```

Submissão de um código com erro de compilação:

```
[SUBMITTER] Code read (10 lines):
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d\n", a+b+1);
    return 0
}

[SUBMITTER] Connecting request submitter...
[SUBMITTER] Attempting submit request
[SUBMITTER] Received: SubAck
```

```
[SUBMITTER] Received: SubAck 8831
[SUBMITTER] Connecting code submitter...
[SUBMITTER] Received: CodAck
[SUBMITTER] Submission id: 3
[SUBMITTER] Received: RemReq
[SUBMITTER] Received: RemFin
[SUBMITTER] Submitter done, verdict: COMPILATION ERROR
```

Submissão de um código com erro em tempo de execução:

```
[SUBMITTER] Code read (10 lines):
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a, b;
    scanf("%d %d", 0, &b);
    printf("%d\n", a+b+1);
    return 0;
}

[SUBMITTER] Connecting request submitter...
[SUBMITTER] Attempting submit request
[SUBMITTER] Received: SubAck
[SUBMITTER] Received: SubAck 7515
[SUBMITTER] Connecting code submitter...
[SUBMITTER] Received: CodAck
[SUBMITTER] Submission id: 5
[SUBMITTER] Received: RemReq
[SUBMITTER] Received: RemFin
[SUBMITTER] Submitter done, verdict: TIME LIMIT EXCEEDED
```

Solicitação da resposta com identificador 0:

```
[REPORTER] Connecting request submitter...
[REPORTER] Attempting report request
[REPORTER] Received: RpoAck
[REPORTER] Reporter done, verdict: NOT ANSWERED YET
```

Solicitação da resposta com identificador 0:

```
[REPORTER] Connecting request submitter...
[REPORTER] Attempting report request
[REPORTER] Received: RpoAck
[REPORTER] Reporter done, verdict: NOT ANSWERED YET
```

Solicitação da resposta com identificador 1:

```
[REPORTER] Connecting request submitter...
[REPORTER] Attempting report request
[REPORTER] Received: RpoAck
[REPORTER] Reporter done, verdict: ACCEPTED
```

Solicitação da resposta com identificador 2:

```
[REPORTER] Connecting request submitter...
[REPORTER] Attempting report request
[REPORTER] Received: RpoAck
[REPORTER] Reporter done, verdict: WRONG ANSWER
```

Solicitação da resposta com identificador 3:

```
[REPORTER] Connecting request submitter...
[REPORTER] Attempting report request
[REPORTER] Received: RpoAck
[REPORTER] Reporter done, verdict: COMPILATION ERROR
```

Solicitação da resposta com identificador 5:

```
[REPORTER] Connecting request submitter...
[REPORTER] Attempting report request
[REPORTER] Received: RpoAck
[REPORTER] Reporter done, verdict: TIME LIMIT EXCEEDED
```

Log do servidor:

```
[SERVER] Server online
[SERVER] Awaiting submit request
[SERVER] Request found
[SERVER] Received: SubReq
[SERVER] Received: SubFin
[SERVER] Submit port reserved: 1679
[CORRECTOR 1679] Awaiting code request
[SERVER] Awaiting submit request
[CORRECTOR 1679] Code request found
[CORRECTOR 1679] Received: CodSub
[CORRECTOR 1679] Received 11 lines for problem 1001
[CORRECTOR 1679] Compiling code
[CORRECTOR 1679] perfoming 3 tests on submission 0
[SERVER] Request found
[SERVER] Received: SubReq
[SERVER] Received: SubFin
[SERVER] Awaiting submit request
[SERVER] Submit port reserved: 9275
[CORRECTOR 9275] Awaiting code request
[CORRECTOR 9275] Code request found
[CORRECTOR 9275] Received: CodSub
[CORRECTOR 9275] Received 10 lines for problem 1001
[CORRECTOR 9275] Compiling code
[CORRECTOR 9275] perfoming 3 tests on submission 1
[CORRECTOR 9275] Received: RemAck
[SERVER] Submit port freed: 9275
[SERVER] Request found
[SERVER] Received: SubReq
[SERVER] Received: SubFin
[SERVER] Awaiting submit request
[SERVER] Submit port reserved: 4722
[CORRECTOR 4722] Awaiting code request
[CORRECTOR 4722] Code request found
[CORRECTOR 4722] Received: CodSub
[CORRECTOR 4722] Received 10 lines for problem 1001
[CORRECTOR 4722] Compiling code
[CORRECTOR 4722] perfoming 3 tests on submission 2
[CORRECTOR 4722] WA, expected 5, found 6
[CORRECTOR 4722] Received: RemAck
[SERVER] Submit port freed: 4722
[SERVER] Request found
[SERVER] Received: SubReq
[SERVER] Received: SubFin
[SERVER] Awaiting submit request
[SERVER] Submit port reserved: 8831
[CORRECTOR 8831] Awaiting code request
[CORRECTOR 8831] Code request found
[CORRECTOR 8831] Received: CodSub
```

```
[CORRECTOR 8831] Received 10 lines for problem 1001
[CORRECTOR 8831] Compiling code
[CORRECTOR 8831] Received: RemAck
[SERVER] Submit port freed: 8831
[SERVER] Request found
[SERVER] Received: SubReq
[SERVER] Received: SubFin
[SERVER] Awaiting submit request
[SERVER] Submit port reserved: 8462
[CORRECTOR 8462] Awaiting code request
[CORRECTOR 8462] Code request found
[CORRECTOR 8462] Received: CodSub
[CORRECTOR 8462] Received 10 lines for problem 1001
[CORRECTOR 8462] Compiling code
[CORRECTOR 8462] Received: RemAck
[SERVER] Submit port freed: 8462
[SERVER] Request found
[SERVER] Received: SubReq
[SERVER] Received: SubFin
[SERVER] Awaiting submit request
[SERVER] Submit port reserved: 7515
[CORRECTOR 7515] Awaiting code request
[CORRECTOR 7515] Code request found
[CORRECTOR 7515] Received: CodSub
[CORRECTOR 7515] Received 10 lines for problem 1001
[CORRECTOR 7515] Compiling code
[CORRECTOR 7515] perfoming 3 tests on submission 5
[CORRECTOR 7515] TLE
[CORRECTOR 7515] Received: RemAck
[SERVER] Submit port freed: 7515
```

4 Conclusão

O protocolo TCP se mostrou simples e seguro de ser usado. Não houve nenhuma dificuldade no projeto em relação ao seu funcionamento, sendo a correção automatizada a parte mais trabalhosa. A linguagem Java e suas API's simplificaram enormemente o trabalho com sua interface de alto nível.

Os testes foram todos bem sucedidos a exceção dos casos de loop infinito e de erro em tempo de execução. Estes casos não foram solucionados pois não eram o foco deste trabalho e tampouco apresentaram risco à integridade do software.

A princípio, havia sido solicitada pela professora que a comunicação fosse feita apenas entre os dois computadores dos membros da dupla. No entanto, devido à alta taxa de bugs no projeto, tornou-se mais simples cada um poder simular toda a comunicação em uma só máquina. Era inviável depender um do outro para a realização de testes.

A atividade se mostrou bastante interessante, pois envolveu aprendizado não apenas do TCP, mas da sua API em Java, além de como se fazer chamadas de linha de comando pela API de Runtime do Java. O projeto, também, possui aplicação prática para a graduação do ITA, como fora citado na introdução. Para a aplicação real, dever-se-ia implementar uma interface amigável, possivelmente um website, para o acesso e para a submissão. Para segurança, seria necessário um cadastro. SQL seria uma boa opção para o banco de dados, pois possui uma interface com o Java.