Adivinha o Número Secreto

Universidade de Aveiro

Mauro Marques Canhão Filho, Patricia Rafaela
da Rocha Cardoso

1.0

Adivinha o Número Secreto

Departamento de Eletrônica, Telecomunicações e Informática (DETI)

Universidade de Aveiro

Mauro Marques Canhão Filho, Patricia Rafaela da Rocha Cardoso
(103411) mauro.filho@ua.pt, (103243) patriciarcardoso@ua.pt

30/05/2020

Resumo

Este relatório tem como objetivo descrever a implementação e a intereção entre um servidor e um ou mais clientes. Para isso, será detalhadamente apresentado o funcionamento/criação de um jogo. O jogo consiste em o cliente adivinhar um número inteiro aleatório entre 0 e 100, o número secreto, gerado aleatoriamente pelo servidor.

Índice

Capítulo 1

Introdução

O objetivo deste trabalho é explicar,enumerar e descrever o desenvolvimento e funcionamento de um servidor que suporte a geração de um número inteiro aleatório (entre 0 e 100), o número secreto, bem como o número máximo de tentativas (entre 10 e 30) concedidas para o adivinhar. E um cliente que permita adivinhar esse número secreto. Ou seja um jogo de adivinha o número secreto. O servidor nunca deverá aceitar dois clientes com a mesma identificação a jogar simultaneamente e deverá criar e atualizar um ficheiro designado por report.csv onde vai escrevendo os resultados dos diversos clientes quando estes terminam o jogo. O cliente pode desistir em qualquer altura e o jogo acaba quando ele adivinha o número secreto ou quando esgota o número máximo de tentativas que dispunha para jogar. Caso o cliente exceda o número de jogadas de que dispunha o jogo será considerado sem sucesso mesmo que ele tenha adivinhado o número. Quando o jogo acaba corretamente o cliente deve escrever no monitor uma mensagem a indicar se adivinhou ou não o número secreto e quantas jogadas efectuou. Por sua vez o servidor acrescenta ao ficheiro a informação relativa ao jogo: cliente; número secreto; número máximo de jogadas; número de jogadas efectuadas; e o resultado obtido pelo cliente (desistência ou sucessso ou insucessso).

Capítulo 2

Metodologia

Neste capítulo será detalhadamente descrito o algoritmo e o funcionamento do progama servidor e do programa cliente bem como a implementação dos testes funcionais e unitários.

2.1 Servidor

O programa servidor consiste em gerar aleatoriamente um número entre 0 e 100 e um número máximo de tentativas entre 10 e 30 para o adivinhar. O programa servidor é constítuido por um dicionário e as seguintes funções: **find_client_id**,**encrypt_intvalue**, **decrypt_intvalue**, **new_msg**, **numberToCompare**, **new_client**, **clean_client**, **quit_client**, **create_file**, **update_file**, **guess_client**, **stop_client** e **main**.

2.1.1 Dicionário e Array

```
    ...
gamers = {'name':[],'sock_id':[], 'segredo':[], 'max':[], 'jogadas':[], 'resultado':[], 'cipherkey':[]}
header = ['name','sock_id','segredo','max', 'jogadas', 'resultado']
    ...
```

Figura 2.1: Dicionário constituído pelos dados dos jogadores e array responsável pela inicialização do header no ficheiro report.csv.

O dicionário "gamers"armazena os dados dos jogadores que estão atualmente com um jogo iniciado. A informação armazenada é baseada na ordem pela qual os clientes se conectam ao servidor. Essa informação é filtrada e distribuída por arrays que contém diferentes campos de identificação. Por exemplo,

se dois jogadores, Mauro e Patrícia estiverem a jogar simultaneamente e se o Mauro se conectou primeiro ao servidor, o seu ID pode ser consultado através de: gamers['sock_id'][0], enquanto o ID da Patrícia pode ser acedido da seguinte forma: gamers['sock_id'][1]. O array "headers"é utilizado para atualizar o cabeçalho do ficheiro report.csv que será gerado pelo servidor.

2.1.2 Função find_client_id

```
    ...

def find_client_id (client_sock):
    peerName = client_sock.getpeername()
    return peerName[1]

    ...
```

Figura 2.2: Função que retorna o porto ao qual o cliente está conectado.

A partir de cada socket de cliente, é possível extrair algumas informações únicas para o identificar.Neste caso, a função .getpeername() devolve uma sequência ordenada que contém o endereço do host e o porto ao qual o cliente está conectado. O porto, por sua vez, é devolvido pela função find_client_id().

2.1.3 Função encrypt_intvalue

```
    ...

def encrypt_intvalue (client_id, data):
    for i in range(0, len(gamers['sock_id'])):
        if gamers['sock_id'][i] == client_id:
            cipherkey = gamers['cipherkey'][i]

    cipher = AES.new(cipherkey, AES.MODE_ECB)
    data2 = cipher.encrypt(bytes("%16d" % (data), 'utf8'))
    data_tosend = str(base64.b64encode(data2), 'utf8')
    return data_tosend

    ...
```

Figura 2.3: Função para encriptar valores a enviar em formato json com codificação base64.

Cada número inteiro comunicado entre o servidor e o cliente é encriptado por blocos usando a função AES-128 no modo ECB. A encriptação é realizada do seguinte modo:

1. Identificação da chave de cifragem relativa ao cliente atual comparando o ID passado como argumento da função e os IDs presentes no dicionário "gamers";

2. Conversão do inteiro numa string binária de 128 bits;

3. Codificação da string no formato Base64 com o intuito dos criptogramas serem suportados pelo JSON;

4. Devolução pela função do valor codificado e encriptado para que possa ser enviado.

2.1.4 Função decrypt_intvalue

```
    ...

def decrypt_intvalue (client_id, data):
    for i in range(0, len(gamers['sock_id'])):
        if gamers['sock_id'][i] == client_id:
            cipherkey = gamers['cipherkey'][i]

    cipher = AES.new(cipherkey, AES.MODE_ECB)
    data1 = base64.b64decode(data)
    data2 = cipher.decrypt(data1)
    print(data2)
    data3 = int(str(data2, 'utf8'))
    return data3

    ...
```

Figura 2.4: Função para desencriptar valores recebidos em formato json com codificação base64

Cada número inteiro comunicado entre o servidor e o cliente é decriptado por blocos usando a função AES-128 em modo ECB. A decriptação ocorre do seguinte modo:

1. Identificação da chave de cifragem relativa ao cliente atual comparando o ID passado como argumento da função e os IDs presentes no dicionário "gamers";

2. Descodificação dos dados passados à função como argumento no formato Base64 e decriptação do seu conteúdo;

3. Codificação para um valor inteiro;

4. Devolução do valor inteiro pela função.

2.1.5 Função new_msg

```
    ...

def new_msg (client_sock):
    request = recv_dict(client_sock)
    print(request)
    if request['op'] == "START":
        new_client(client_sock, request)
    if request['op'] == "QUIT":
        quit_client(client_sock)
    if request['op'] == "STOP":
        stop_client(client_sock, request)
    if request['op'] == "GUESS":
        guess_client(client_sock, request)
    return None

    ...
```

Figura 2.5: Função chamada sempre que o servidor recebe uma nova mensagem do cliente.

A tarefa desta função é identificar qual a operação requisitada pelo cliente e encaminhá-la para a função que irá processar e responder ao pedido. Caso seja feito um pedido de uma operação fora do alcançe da aplicação não ocorre qualquer comportamento por parte do servidor.

2.1.6 Função numberToCompare

```
    ...

def numberToCompare(client_sock):
    id = find_client_id(client_sock)
    for i in range(0, len(gamers['sock_id'])):
        if gamers['sock_id'][i] == id:
            return gamers['segredo'][i]

    ...
```

Figura 2.6: Função que devolve o número secreto.

Esta função suporta o comando "Guess".

2.1.7 Função new_client

```
    ...

def new_client (client_sock, request):
    name = request['client_id']
    sock_id = find_client_id(client_sock)
    if name in gamers['name']:
        response = {'op': "START", 'status': False, 'error': "Cliente existente"}
        send_dict(client_sock, response)
    else:
        gamers['name'].append(name)
        gamers['sock_id'].append(sock_id)
        n = random.randint(10, 30)
        secret = random.randint(0, 100)
        gamers['segredo'].append(secret)
        gamers['max'].append(n)
        gamers['jogadas'].append(0)
        gamers['cipherkey'].append(base64.b64decode(request['cipherkey']))
        print(gamers)
        response = {'op': "START", 'status': True, 'max_attempts': encrypt_intvalue(sock_id,n)}
        send_dict(client_sock, response)
    return None

    ...
```

Figura 2.7: Função que cria um novo cliente no jogo.

Esta função suporta a criação de um novo jogador induzida pela operação START. O seu funcionamento pode ser descrito da seguinte forma:

1. Armazenamento na variável "name"do "client_id"passado para o servidor aquando da inserção pelo utilizador na linha de comandos ao executar o cliente;

2. Identificação do ID(porto ao qual está conectado) do cliente a partir do socket com a função "find_client_id";

3. Envio de uma resposta do servidor para o cliente com status: True; e com o valor encriptado de jogadas máximas que o cliente pode fazer.

Se "name"("client_id"enviado pelo pedido do cliente) já se encontre no dicionário "gamers", o servidor irá relatar ao cliente uma mensagem de status: False; e uma mensagem de erro indicando a já utilização desse nome. Caso contrário, a função adiciona todos os dados necessários do cliente aos arrays do dicionário. É depois, iniciado um jogo.

2.1.8 Função clean_client

```
    ...

def clean_client (client_sock):
    id = find_client_id(client_sock)
    print("numero de gamers: " + str(len(gamers['sock_id'])))
    for i in range(0, len(gamers['sock_id'])):
        print("index: "+str(i))
        if gamers['sock_id'][i] == id:
            gamers['segredo'].pop(i)
            gamers['sock_id'].pop(i)
            gamers['name'].pop(i)
            gamers['max'].pop(i)
            gamers['jogadas'].pop(i)
            gamers['cipherkey'].pop(i)
            return True
    return False

    ...
```

Figura 2.8: Função chamada sempre que é necessário apagar um jogador da lista de jogadores ativos.

Esta função é executada sempre que for necessário excluir um cliente do dicionário "gamers". Isto ocorre quando o cliente se desconecta do servidor, quando termina o jogo ou quando desiste. A função procura pelo cliente no dicionário "gamers"e caso o encontre, exclui todos os dados a ele associados através do seu respetivo índice.

2.1.9 Função quit_client

```
    ...

def quit_client (client_sock):
    if find_client_id(client_sock) in gamers['sock_id']:
        response = {'op': "QUIT", 'status': True}
        send_dict(client_sock, response)
        update_file(find_client_id(client_sock), 'DESISTENCIA')
        clean_client(client_sock)
    else:
        response = {'op': "QUIT", 'status': False, 'error': "cliente inexistente"}
        send_dict(client_sock, response)
    print("CURRENT GAMERS: "+str(gamers))
    return None

    ...
```

Figura 2.9: Função chamada quando o cliente pretende desistir do jogo.

Esta função suporta o pedido de desistência de um cliente - operação QUIT.Primeiro, a função verifica se o cliente que pretende desistir encontra-se realmente em jogo. Para isto, verifica se o ID do socket está presente no dicionário "gamers". Em caso afirmativo, o servidor envia uma mensagem ao cliente com status: True;

e atualiza o ficheiro report.csv(recorrendo à função update_file()) com o resultado "DESISTENCIA". Este resultado indica que a partida foi terminada antes de o jogador adivinhar o número secreto ou antes de atingir o limite de jogadas. Por fim, remove o cliente da lista de jogadores ativos recorrendo à função clean_client. Caso contrário, envia uma mensagem ao cliente com status: False; e uma mensagem de erro que explicita o facto de o cliente não ter sido encontrado entre os jogadores ativos.

2.1.10 Função create_file

```
    ...

def create_file ():
    if path.exists('report.csv') == False:
        with open('report.csv', 'w') as fileCSV:
            writer = csv.DictWriter(fileCSV, fieldnames=header)
            writer.writeheader()
    return None

    ...
```

Figura 2.10: Função que cria um ficheiro report.csv quando o servidor é inicializado.

No momento em que o servidor é inicializado é chamada a função "create_file"para que seja criado um novo ficheiro report.csv caso ainda não exista no diretório em que o server.py se encontra de maneira a que o servidor não reinicie o ficheiro toda a vez que for inicializado. Depois, escrever o cabeçalho no ficheiro com base no array "header".

2.1.11 Função update_file

```
    ...

def update_file (client_id, result):
    with open('report.csv', 'a') as fileCSV:
        writer = csv.DictWriter(fileCSV, fieldnames=header)
        for i in range(0, len(gamers['sock_id'])):

            if client_id == gamers['sock_id'][i]:

                di = {  'name': gamers['name'][i], 'sock_id': gamers['sock_id'][i],
                        'segredo': gamers['segredo'][i], 'max': gamers['max'][i],
                       'jogadas': gamers['jogadas'][i], 'resultado': result}
        writer.writerow(di)
    return None

    ...
```

Figura 2.11: Função que atualiza o ficheiro report.csv quando um jogo é terminado.

Esta função atualiza o ficheiro report.csv com os dados de um jogador quando um jogo é terminado(com sucesso, sem sucesso ou desistência). Para isso, abre o ficheiro no modo "a"(append) para adicionar dados sem escrever sobre aqueles que já lá estavam.Assim, procura pelo index "i"tal que o sock_id é igual ao client_id passado como parâmetro da função. Por fim, escreve todos os itens na posição "i"dos arrays do dicionário "gamers".

2.1.12 Função guess_client

```
    ...

def guess_client (client_sock, request):

    if find_client_id(client_sock) in gamers['sock_id']:
        segredo = numberToCompare(client_sock)
        jogado = decrypt_intvalue(find_client_id(client_sock),request['number'])

        if jogado == segredo:
            response = {'op': "GUESS", 'status': True, 'result':"equals"}
            send_dict(client_sock, response)
        if jogado > segredo:
            response = {'op': "GUESS", 'status': True, 'result':"larger"}
            send_dict(client_sock, response)
        if jogado < segredo:
            response = {'op': "GUESS", 'status': True, 'result':"smaller"}
            send_dict(client_sock, response)
        for i in range(0, len(gamers['sock_id'])):
            if find_client_id(client_sock) == gamers['sock_id'][i]:
                gamers['jogadas'][i] = gamers['jogadas'][i] + 1
    else:
        response = {'op': "GUESS", 'status': False, 'error': "Client inexistente"}
        send_dict(client_sock, response)

    return None

    ...
```

Figura 2.12: Suporte da jogada de um cliente - Operação GUESS.

Para que a função possa funcionar corretamente, temos que averigurar se o cliente que está a jogar tem realmente uma sessão iniciada no jogo. Se ele se encontrar no dicionário "gamers"prosseguimos com o GUESS.Caso contrário, o servidor envia uma mensagem ao cliente com o status: False; e uma mensagem de erro a indicar que o cliente não se encontra na lista de jogadores ativos. Consideremos agora o caso em que o cliente tem um jogo iniciado. Primeiro, procuramos o valor do número secreto deste cliente através da função "numberToCompare()", que será armazenado na variável segredo. Depois, descriptografamos o número inserido pelo jogador(que é passado na mensagem enviada do cliente ao servidor e que depois é encaminhada para a função pelo parâmetro "request") que é armazenado na variável "jogado". Se o número for igual ao número secreto, o servidor envia uma mensagem ao cliente com status: True e result: "equals", a indicar que o jogador acertou no número. Se o número for maior que o segredo, o servidor envia uma mensagem ao cliente com status: True e result: "larger"a indicar que o jogador introduziu um número

superior ao número secreto. Se o número for menor que o segredo, o servidor envia uma mensagem ao cliente com status: True e result: "smaller", a indicar que o jogador introduziu um número mais pequeno que o número secreto. Por fim, atualiza no dicionário "gamers"o número de jogadas efetuadas.

2.1.13 Função stop_client

```
    ...

def stop_client (client_sock, request):
    if find_client_id(client_sock) in gamers['sock_id']:
        response = {'op': "STOP", 'status': True}
        send_dict(client_sock, response)
        for i in range(0, len(gamers['sock_id'])):
            if find_client_id(client_sock) == gamers['sock_id'][i]:
                gamers['jogadas'][i] = decrypt_intvalue(gamers['sock_id'][i],request['attempts'])
                if gamers['segredo'][i] == decrypt_intvalue(gamers['sock_id'][i],request['number']):
                    update_file(find_client_id(client_sock), "SUCCESS")
                else:
                    update_file(find_client_id(client_sock), "FAILURE")
        clean_client(client_sock)

    else:
        response = {'op': "STOP", 'status': False, 'error': "cliente inexistente"}
        send_dict(client_sock, response)
    print("CURRENT GAMERS: " + str(gamers))
    return None

    ...
```

Figura 2.13: Função responsável por encerrar o jogo.

Representa o suporte do pedido de terminação de um cliente - a operação STOP. Esta operação ocorre sempre que um jogo é terminado ou porque o jogador acertou no número secreto ou porque efetuou mais jogadas dos que as que possuía. Para que um jogo seja encerrado, o cliente precisa estar na lista de jogadores ativos, ou seja, no dicionário "gamers". Se o cliente não se encontrar ativo no jogo, a função envia-lhe uma mensagem com status: False e uma mensagem de erro a indicar que o cliente não se encontra na lista de jogadores ativos. Caso o cliente esteja ativo no jogo, o servidor envia-lhe uma mensagem com status: True, a indicar que a finalização do jogo foi processada. O processamento do encerramento do jogo dá-se da seguinte forma:

1. O servidor atualiza no dicionário "gamers"o número de jogadas efetuadas pelo jogador. Para isso, deve descriptograr o número inteiro enviado pelo cliente com auxílio da função "decrypt_intvalue()";

2. O servidor verifica se o último número jogado pelo utilizador(que também deve ser descriptografado) coincide com o número secreto. Caso seja, atualiza o ficheiro report.csv com os dados do cliente e o resultado final "SUCCESS". Caso contrário, atualiza o ficheiro report.csv com os dados do cliente e o resultado final "FAILURE";

3. Elimina o cliente da lista de jogadores ativos através da função "clean_client()".

2.1.14 Função main

```
    ...
def main():

    if len(sys.argv) != 2:
        sys.exit("Deve passar o porto como argumento para o servidor")
    try:
        int(sys.argv[1])
    except ValueError:
        sys.exit("Porto deve ser um numero inteiro")
    if int(sys.argv[1])<0:
        sys.exit("Porto deve ser um numero inteiro positivo")

    port = int(sys.argv[1])



    server_socket = socket.socket (socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind (("127.0.0.1", port))
    server_socket.listen (10)

    clients = []
    create_file ()

    while True:
        try:
            available = select.select ([server_socket] + clients, [], [])[0]
        except ValueError:

            for client_sock in clients:
                if client_sock.fileno() == -1: client_sock.remove()
            continue

        for client_sock in available:

            if client_sock is server_socket:
                newclient, addr = server_socket.accept ()
                clients.append (newclient)

            else:

                if len (client_sock.recv (1, socket.MSG_PEEK)) != 0:

                    new_msg (client_sock)
                else:
                    clients.remove (client_sock)
                    clean_client (client_sock)
                    client_sock.close ()
                    break

if __name__ == "__main__":
    main()
```

Figura 2.14: Função que permite o funcionamento correto de todo o servidor.

2.2 Cliente

Capítulo 3

Resultados

Capítulo 4

Análise

Capítulo 5

Conclusões

Contribuições dos autores

Acrónimos