

TRABALHO DE APROUNDAMENTO 2

Rafael Santos-98466, Hugo Domingos-98502



**universidade
de aveiro**

**TRABALHO DE
APROUNDAMENTO 2**
DEPARTAMENTO DE
ELTRÓNICA, TELECOMUNICAÇÕES E
INFORMÁTICA

Rafael Santos-98466, Hugo Domingos-98502

9/05/2020

Conteúdo

1	Introdução	1
2	Metodologia	2
2.1	Comunicação Servidor-Cliente	3
2.1.1	Servidor	3
2.1.2	Cliente	4
2.1.3	Processo de distribuição	7
3	Resultados	8
3.0.1	Listagem dos clientes	8
3.0.2	Processo de ordenação	8
3.0.3	Exportação dos dados	11

Capítulo 1

Introdução

Este trabalho de aprofundamento foi-nos proposto no âmbito da unidade curricular de Laboratórios de Informática.

Este projeto trata-se de um processo de ordenação de várias pessoas que participam num certo evento. Cada um vai obter aleatoriamente um número de ordem .

Quando cada um revelar o seu número de ordem todos poderão comprovar que não há números coincidente.

Capítulo 2

Metodologia

Neste projeto, tal como aconselhado pelos docentes da unidade curricular, foram usados sockets TCP para a comunicação entre o servidor e o cliente, sendo também usado o código fornecido pelos docentes tal como `b64.py`, `common_com.py`, e `cifradecifra.py`. Para realizar este projeto foram usados os conhecimentos de sockets, de criptografia e de ficheiros CSV.

2.1 Comunicação Servidor-Cliente

2.1.1 Servidor

Partindo do código inicial fornecido pelos professores, foi-nos necessário numa fase inicial colocar o servidor a funcionar.

```
def main():  
    # Validate the program parameters  
  
    # Set the server's TCP address from the command args  
    porto = int(sys.argv[1])  
  
    address = (socket.gethostname(), porto)  
  
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
    s.bind(address)  
    s.listen()
```

Figura 2.1: Inicialização do Servidor

Na figura 2.1, no que respeita ao "porto", este é definido através da consola e serve para nomear a socket. O parametro "s" é relativo à criação da socket TCP. É utilizado o bind associado à socket "s" para que seja definido o "end point" da comunicação;

2.1.2 Cliente

Com o servidor ativo foi necessário conectar o cliente ao servidor, sendo necessário colocar os parametros na consola.

```
def main():
    # Validate the program parameters
    porto = int(sys.argv[3])

    # Set the server's TCP address from the command args

    address = ((socket.gethostname(), porto ))

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(address)

    # Set the process_id and the client_id from the command args

    process_id = sys.argv[1]
    client_id = sys.argv[2]

    if client_id == "_":
        dump_csv(manager(s, process_id))
    else:
        client(s, process_id, client_id)
```

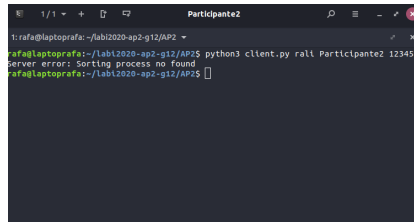
Figura 2.2: Cliente

Para que o cliente entre em contacto com o servidor é necessário recorrer às informações atribuídas ao servidor, tal como o "address" e o "porto".

Para iniciar o cliente é necessário colocar o id do processo (process_id) e o id do cliente (client_id), aquilo que vai identificar o participante.

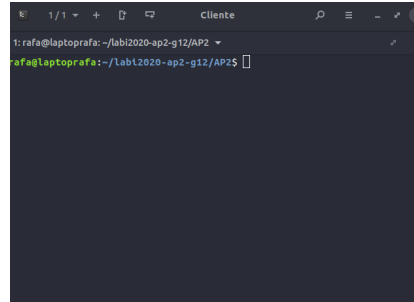
Se o id for "_", vai ser gerado o processo onde poderemos aceder ao número e identidade dos participantes e realizar o processo de distribuição. Caso seja

tentado adicionar um participante mas o cliente ainda não tenha sido criado, irá ser impresso no ecrã :`"Server error: Sorting process no found"`. Se for tentado ser criado um novo processo com mesmo nome do já criado o erro será: `"error': 'Sorting process already exists "`. Caso o participante 1 tenha o mesmo id que o participante 2 deverá ocorrer : `"'error': 'Client already registered in sorting process' "`.



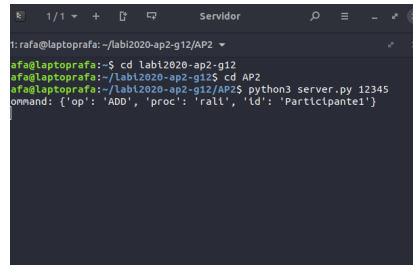
```
1:rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 client.py rali Participante2 12345
Server error: Sorting process no found
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$
```

(a) Sorting process not found



```
1:rafa@laptoprafa:~/lab12020-ap2-g12/AP2$
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$
```

(b) Cliente não criado



```
1:rafa@laptoprafa:~/lab12020-ap2-g12/AP2$
rafa@laptoprafa:~$ cd lab12020-ap2-g12
rafa@laptoprafa:~/lab12020-ap2-g12$ cd AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 server.py 12345
ommand: {'op': 'ADD', 'proc': 'rali', 'id': 'Participante1'}
```

(c) Servidor ativo

Figura 2.3: Sorting process not found

```

t:rafa@laptoprafa: ~/lab12020-ap2-g12/AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 server.py 12345
Command: {'op': 'NEW', 'proc': 'rall'}
Command: {'op': 'NEW', 'proc': 'rall'}

```

(a) Servidor ativo

```

t:rafa@laptoprafa: ~/lab12020-ap2-g12/AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 client.py rall _ 12345
(error: " ")
Commands for ordering on process "rall"
L or l - list the clients
S or s - start ordering
Q or q - quit
prompt:

```

(b) Processo de sorting criado

```

t:rafa@laptoprafa: ~/lab12020-ap2-g12/AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 client.py rall _ 12345
(error: "Sorting process already exists")
Server error: Sorting process already exists
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$

```

(c) error:Sorting process already exists

Figura 2.4: Sorting process not found

```

t:rafa@laptoprafa: ~/lab12020-ap2-g12/AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ cd lab12020-ap2-g12
rafa@laptoprafa:~/lab12020-ap2-g12$ cd AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 server.py 12345
Command: {'op': 'NEW', 'proc': 'rall'}
Command: {'op': 'ADD', 'proc': 'rall', 'id': 'participante1'}
Command: {'op': 'ADD', 'proc': 'rall', 'id': 'participante1'}
Command: {'op': 'LIST', 'proc': 'rall'}

```

(a) Servidor ativo

```

t:rafa@laptoprafa: ~/lab12020-ap2-g12/AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 client.py rall participante1 12345
Server error: Client already registered in sorting process
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$

```

(b) error:Client already registered in sorting process

```

t:rafa@laptoprafa: ~/lab12020-ap2-g12/AP2
rafa@laptoprafa:~/lab12020-ap2-g12/AP2$ python3 client.py rall participante1 12345

```

(c) Participante1 já criado

Figura 2.5: error:Client already registered in sorting process

2.1.3 Processo de distribuição

Feita a conexão entre o servidor e o cliente é necessário colocar os participantes no processo. Iremos chamar ao processo "rali". O utilizador terá a opção de listar os clientes e inicializar o processo de ordenação.

```
rafa@laptoprafa:~$ cd labi2020-ap2-g12
rafa@laptoprafa:~/labi2020-ap2-g12$ cd AP2
rafa@laptoprafa:~/labi2020-ap2-g12/AP2$ python3 server.py 12345
python3: can't open file 'server.py': [Errno 2] No such file or directory
rafa@laptoprafa:~/labi2020-ap2-g12/AP2$ python3 server_tcp.py 12345
Command: {'op': 'NEW', 'proc': 'rali'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u8'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u9'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u10'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u6'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u5'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u7'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u4'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u3'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u2'}
Command: {'op': 'ADD', 'proc': 'rali', 'id': 'u1'}

```

Figura 2.6: Servidor com 10 participantes já listados .

```
1/1 + [?] Cliente
1: rafa@laptoprafa: ~/labi2020-ap2-g12/AP2
rafa@laptoprafa:~$ cd labi2020-ap2-g12
rafa@laptoprafa:~/labi2020-ap2-g12$ cd AP2
rafa@laptoprafa:~/labi2020-ap2-g12/AP2$ python3 client.py rali _ 12345
{'error': ''}
Commands for ordering on process "rali"
L or l - list the clients
S or s - start ordering
Q or q - quit
prompt: 
```

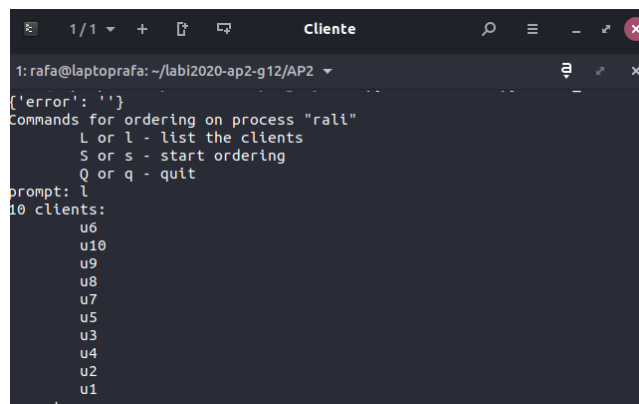
Figura 2.7: Cliente

Capítulo 3

Resultados

3.0.1 Listagem dos clientes

Olhando para o cliente, ao pressionar-mos "l" obtemos a identificação dos participantes por ordem de entrada no processo "rali". *



```
1:rafa@laptoprafa: ~/labi2020-ap2-g12/AP2
[error: '']
Commands for ordering on process "rali"
L or l - list the clients
S or s - start ordering
Q or q - quit
prompt: l
10 clients:
u6
u10
u9
u8
u7
u5
u3
u4
u2
u1
```

Figura 3.1: Participantes ordenados por ordem de entrada no processo.

3.0.2 Processo de ordenação

No processo de ordenação vai ser gerada uma sequência de números de ordem de 1 a N. Estes vão ser encriptados, baralhados e dados a um participante para que escolha um. Este processo vai ser repetido por todos os participantes.

```
def run_sorting(proc):
    # cria uma lista com os ids
    global final_key
    clients = []
    lista = []
    data1 = []
    keys = []

    # cria lista com os nomes dos clients por ordem de conexao
    for key in proc['ids']:
        clients.append(key)

    iterations = len(clients)

    # cria lista com numeros de ordem de 1 a N
    for i in range(1, iterations + 1):
        lista.append(i)

    # envia a lista a um dos clients e recebe uma lista
    for i in range(iterations):
        send_dict(proc['ids'][clients[i]]['endpoint'], lista)
        data1 = recv_dict(proc['ids'][clients[i]]['endpoint'])

    # envia lista cifrada e baralhada por todos os clients a um de cada vez
    # e recebe a lista com o elemento que o client escolheu
    for i in range(iterations):
        send_dict(proc['ids'][clients[i]]['endpoint'], data1)
        data1 = recv_dict(proc['ids'][clients[i]]['endpoint'])

    # recebe o bit commitment e o elemento escolhido anteriormente
    b = []
    c = []
    for i in range(iterations):
        Ci = recv_dict(proc['ids'][clients[i]]['endpoint'])
        Ci.append(Ci)
        Bi = recv_dict(proc['ids'][clients[i]]['endpoint'])
        Bi.append(Bi)
```

(a)

```
# associar o nome de cada client com o par(Ci, Bi)
dic = {}
for i in range(iterations):
    dic[clients[i]] = ("Ci": Ci[i], "Bi": Bi[i])

dic_h = {}
cont = 0
for i in range(iterations):
    send_dict(proc['ids'][clients[i]]['endpoint'], dic)
    continua = recv_dict(proc['ids'][clients[i]]['endpoint'])
    cont.append(continua)

# Verifica se algum client saiu(se saiu acabar o programa)
for i in cont:
    if i != 1:
        print("Alguém saiu do processo... terminando")
        sys.exit(0)

# Pedir a chave de cada client
keys = []
for i in range(iterations):
    key = recv_dict(proc['ids'][clients[i]]['endpoint'])
    keys.append(key)

# Criar os trios(Ci, Bi, Ki) e enviar para o client
for i in range(iterations):
    dic[clients[i]]["K"] = keys[i]

for i in range(iterations):
    send_dict(proc['ids'][clients[i]]['endpoint'], keys)
    send_dict(proc['ids'][clients[i]]['endpoint'], dic)

final_list = []
for i in range(iterations):
    l = recv_dict(proc['ids'][clients[i]]['endpoint'])
    final_list.append(l)
```

(b)

```
# Descript a lista final
l1 = []
for i in range(iterations):
    l1.append(base64.b64decode(Ci[i]))

l2 = []
for i in range(iterations):
    final_key = keys[i].encode('utf-8')
    final_key = AES.new(final_key, AES.MODE_ECB)

    for n in range(iterations):
        final_sol = final_key.decrypt(l1[n])
        l2.append(int(str(final_sol, 'utf-8')))

# Associar id ao numero de ordem obtido
d = {}
for i in range(iterations):
    d.append({'id': clients[i], 'Nº de ordem': l2[i]})

# cria lista ordenada por nº de ordem com os ids associados
s_list = sorted(d, key = lambda d: d['Nº de ordem'])
sorted_list = []
for i in range(iterations):
    sorted_list.append(s_list[i]['id'])
print(sorted_list)

# Adicionar o nº de ordem ao trio
for i in range(iterations):
    dic[clients[i]]["Nº de ordem"] = l2[i]

# Enviar ordem final ao client
for i in range(iterations):
    send_dict(proc['ids'][clients[i]]['endpoint'], sorted_list)

return dic, sorted_list
```

(c)

Figura 3.2: Código de Sorting explicado

No que toca à parte encriptar , baralhar e distribuir o nº de ordem pelos participantes foi usada a "def client" do cliente.

```
def client(server, proc, client):
    request = {'op': 'AES', 'proc': proc, 'id': client}
    resp = sendrecv_dict(server, request)

    if resp['error'] != '':
        print('Server error: ' + resp['error'])
        sys.exit(2)

    # Wait for server orders and show the ordering outcome at the end
    lista = recv_dict(server)

    # atrasa 1s o código a seguir para nao ficar tudo baralhado
    time.sleep(0.1)

    # codifica a lista
    iterations = len(lista)
    keys = []

    # cria chave aleatoria
    key = os.urandom(16)
    key = base64.b64encode(key)

    key_a = AES.new(key, AES.MODE_ECB)

    # codificar a lista
    data = []
    for i in range(iterations):
        d = bytes("%16s" % lista[i]), 'utf-8')
        d = key_a.encrypt(d)
        data.append(d)

    # baralhar lista
    random.shuffle(data)
```

(a)

```
# Test
for i in range(iterations):
    st = base64.b64encode(data[i])
    data[i] = st.decode('utf-8')

# enviar a lista ao server
send_dict(server, data)

# recebe lista, retira um e envia a lista sem elemento escolhido
data1 = recv_dict(server)
C1 = data1.pop(0)

send_dict(server, data1)

# comprometer com o numero de ordem que retirou
#
# envia o elemento escolhido
send_dict(server, C1)

# cria compromisso
B1 = SHA256.new()
B1.update(bytes(C1, 'utf-8'))
B1.update(key)
B1_digest = B1.digest()

B1_digest = base64.b64encode(B1_digest)
send_dict(server, B1_digest.decode('utf-8'))

dic = recv_dict(server)

# Compara C1 recebido pelo server com o valor que escolheu e B1 com o bit Commitment
continua = 1
if dic['client'][0] != C1 or dic['client'][1] != B1_digest.decode('utf-8'):
    print("Não diferentes no client " + client)
    server.close()
    continua = -1
```

(b)

```
# Compara C1 recebido pelo server com o valor que escolheu e B1 com o bit Commitment
continua = 1
if dic['client'][0] != C1 or dic['client'][1] != B1_digest.decode('utf-8'):
    print("Não diferentes no client " + client)
    server.close()
    continua = -1

send_dict(server, continua)

# Envia as keys
send_dict(server, key.decode('utf-8'))

# recebe lista com as keys
chaves = recv_dict(server)

new_chaves = []
for i in range(iterations):
    new_chaves.append(chaves[i].encode('utf-8'))

# Recebe o trio e verifica se está correto
trio = recv_dict(server)

if trio['client'][0] != C1 or trio['client'][1] != B1_digest.decode('utf-8') or trio['client'][2] != key.decode('utf-8'):
    print("Não um trio correto no participante " + client)

n_C1 = base64.b64decode(C1)

# Decifra o compromisso
for i in range(iterations):
    final_key = AES.new(new_chaves[i], AES.MODE_ECB)
    msg = final_key.decrypt(n_C1)

    f_C1 = int(msg, 'utf-8')
    send_dict(server, f_C1)

ordem_final = recv_dict(server)
print(ordem_final)
```

(c)

Figura 3.3: Código "def client" explicado.

3.0.3 Exportação dos dados

No fim do processo de sorting todos os dados gerados (tais como o n° de ordem, key, bit commitment, o elemento cifrado) de cada um dos participantes devem ser exportados para um ficheiro csv(report.csv).

```
def dump_csv(dic):
    # Dump a CSV to a file from data received from the server
    ids = []
    for i in dic.keys():
        ids.append(i)

    for i in range(len(ids)):
        dic[ids[i]]['ID'] = ids[i]

    fout = open('report.csv', 'w')
    fieldnames = ['ID', 'Ci', 'Bi', 'K', 'N° de ordem']
    w = csv.DictWriter(fout, fieldnames = fieldnames)
    w.writeheader()
    for i in range(len(dic.keys())):
        w.writerow({'ID':dic[ids[i]]['ID'], 'Ci':dic[ids[i]]['Ci'], 'Bi':dic[ids[i]]['Bi'], 'K':dic[ids[i]]['K'], 'N° de ordem':dic[ids[i]]['N° de ordem']})
    fout.close()
```

Figura 3.4: dump_proc - servidor.

```
# tem de retornar a sorted list
def dump_proc(sorted_list):
    return sorted_list
```

Figura 3.5: dump_csv - cliente.

Se admitir-mos que entraram no processo 10 participantes, o resultado do ficheiro CSV final será o seguinte:

```
ID,Ci,Bi,K,N° de ordem
u10,DWYUWGK/z1e1byKRMADHQ==,rCcsQbCQTgEWKxKRIQkHCjcQt1MdS7Xvk2LNdLcoF2o=,NELBzSpkR2k+p8ebrQHEvQ==,10
u9,EgFHH7rvFIPHMD57K0XK/w==,rJHE41q1Ey2b4QZQoAHYaxm2chR9shLfwxynnAlruWA=,05Pb2r6y2g6Byz1Ud9D1KA==,8
u8,UfGD8cKkc46ehfPmzobzA==,rS3XAvGF0EgFlcRzkLByLknRgRYp21LAXcDfUFJ1n0=,tGegSC0eduK+3EwSk2g50Q==,4
u7,qAZvZU8glt0ORLK+Wv7byw==,+W+TYhv/dLPhrpWpYpwI/ScxgLvLFqHmVaJ976Fafce=,hZ22I4921wXIEpfnDvvdw==,7
u5,iNRLmc492b13kgrM+lUCZw==,g6jr7ckORZmV4mrk3LZwJ73QR2Ly7LFgU549zoqx9Do=,Z8jDNU/nIsP1qsFZ1HDG2Q==,2
u6,ywQECFYDyhuo10r0ZKhR7Q==,sdC84IYXAQ16UsemA5k9P8ySHuMAlwsXGNC1spMFG1E=,eB0jtyqk+35LG9sPs917Zw==,1
u2,92wVQs20RcJUhoJ1b0QT/Q==,Et2fYvc54qdYWCzy+LE0heqY08FeskKChhb0lc1Jaxg=,T8FqQV7oDSn1Tp7vJC+XRA==,3
u4,6ZChkKSHhm8A6RpMey61Yg==,nIm5WYaeEwd7TE+LGPZyHwZrdyeuIxxTR1P4rCty8g=,eu0wgcIUy4gVGZP3gNdzHA==,6
u3,cPykvzHvuFvmW11p0t0eXw==,m8azt0K4rv1PY71nkAVqJFCK3oNDVxKw6TYRSANcpqw=,wS0zhX0200L4np8ISY8X6g==,9
u1,909Xqg9bTKjvxLp039Louw==,UCGwGRHI9nMLVFeDLNfV5Vd0aG/v2n9CLTQ03WKhbcI=,K0TFHD59oG01z6FMHnVuJw==,5
```

Figura 3.6: report.csv

Contribuições dos autores

Consideramos que houve uma contribuição de 50% para cada um dos dois membros do grupo