Prototyping assignment

Name	Student number
Paul David Beck	365772
Hammad Abdullah	453082

The program consists of three components: server, client and sensore.
Github: https://github.com/PaulsBecks/fc-reliable-messaging

component	file name	start command
sensor	sensore.py	python3 sensor.py <name>
client	client.py	python3 client.py <name>
server	server.py	python3 server.py

Sensor

The sensor is generating data. It periodically reads the cpu and memory usage and writes it to a file. The file name is generated by the only input the sensor program receives. Next to the sensor data it logs an identifier and the current time in nanoseconds into a file (Fig 1). We make the assumption that not more than one log is written to the file per nanosecond. Otherwise our program will fail.

Client

The client reads the generated data from the file the sensor is writing to and tries to send it to the server (Fig 2). If it was able to send the log successfully, it prints an acknowledgement for every log line (Fig 3) and removes the lines from the file (Fig 2). As reading and writing to a file are blocking operations the sensor is not able to add any more lines while the transmission is ongoing and hence only the data already sent to the server gets deleted.
If the connection to the server breaks while transmitting it will not receive the acknowledgement and hence will not delete the data. In case the data is processed by the server, but the acknowledgement is not received by the client the data is sent twice. Hence the server needs to take care of duplicates.

Server

The server listens for incoming connections (Fig 4). If a new http post request is accepted it reads the body and parses it (Fig 5). It checks in the dedicated log file if the id of the last log is greater than the incoming one. If this is the case the information is already logged and hence will be discarded this time. Finally an acknowledgement is sent to the client (Fig 6).
We have to make sure that data is received in FIFO order. If this is not the case our program will fail. This means that log lines need to be read in the same order as they are written. Also, log lines need to be received at the server in the same order they are sent. Both are valid as the client reads the lines from the local log in FIFO order, sends them in FIFO order and reading from the local log file and sending to the server is done synchronously and non parallel.

We do not add proof that we ran the server on a remote AWS machine and the client locally as it can be seen in the video and simple images wouldn't add any more proof to it.

Appendix

```
1   paul 13.2 64.2 1612091235112730000
2   paul 3.1 64.3 1612091236118373000
3   paul 4.0 64.3 1612091237123550000
4   paul 3.2 64.5 1612091238129827000
5   paul 2.2 64.5 1612091239135948000
6   paul 6.2 64.2 1612091240142074000
7   paul 3.2 64.2 1612091241148157000
8   paul 3.4 64.3 1612091242154263000
9   paul 2.0 64.3 1612091243160399000
10  paul 3.6 64.3 1612091244166490000
11  paul 6.7 64.1 1612091245172450000
12  paul 8.1 64.3 1612091246174554000
13  paul 12.9 64.8 1612091247178695000
14  paul 13.2 64.7 1612091248184247000
15  paul 7.2 64.6 1612091249187168000
16  paul 12.2 64.5 1612091250193018000
17  paul 9.2 64.5 1612091251198529000
```

Fig 1: Sensor data generated by the command *python3 sensor.py paul*. From left to right separated by spaces: sensor identifier, cpu usage percentage, virtual memory usage percentage, time in nanoseconds.

```
16  log_file = 'client-'+name+'.log'
17
18  print("Start client.")
19  while True:
20    # check if data is in file if so send it line by line to the server
21    try:
22      with open(log_file, 'r+') as f:
23        for line in f:
24            response = requests.post(BACKEND_URL, line.rstrip().encode('utf-8'))
25            print(response.text)
26        f.truncate(0)
```

Fig 2: The client loops endlessly (ll. 19). Every loop it opens the log file (ll. 22) and sends every line to the server (ll. 24). The response will be printed (ll. 25) and finally once all logs are sent, they are removed from the local log file (ll. 26).

```
❯ python3 client.py paul
Start client.
ACK 16120229667130520000
ACK 16120229677184880000
ACK 16120229687242430000
ACK 16120229697301460000
ACK 16120229707340710000
ACK 16120230346070970000
ACK 16120230356127880000
```

Fig 3: The client printing acknowledgements of transmitted logs. Next to ACK the timestamp of the successful transmitted log is printed.

```
46  def run(server_class=HTTPServer, handler_class=Server):
47      print("Start server on port 8000.")
48      server_address = ('', 8000)
49      httpd = server_class(server_address, handler_class)
50      httpd.handle_request()
```

Fig 4: The server listens on port 8000 for incoming connections.

```
11      def do_POST(self):
12          content_length = int(self.headers['Content-Length'])
13          data = self.rfile.read(content_length).decode('utf-8')
14          dataSplit = data.split(" ")
15          peerId = str(dataSplit[0])
16          count = str(dataSplit[-1])
17          peerLogFile = "server-"+peerId+".log"
```

Fig 5: The server accepts post requests and parses its content.

```
18          def send_response():
19              self.send_response(200)
20              self.send_header("Content-type", "text/html")
21              self.end_headers()
22              self.wfile.write(("ACK "+count).encode("utf-8"))
```

Fig 6: The send_response method returns the acknowledgement to the client.