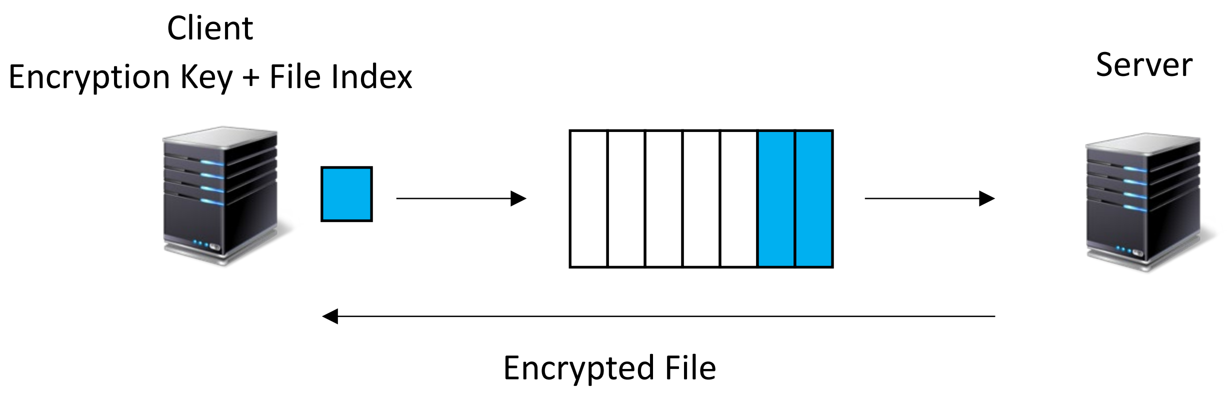
LINFO2241 : Project phase 1 (v1.2)

Imagine you have a company with customers using one of your products, a client-server application. When looking at the performance of such an application, one is typically interested in two problems:

- What response time do my customers experience?
    - If the response time is high, what is the reason for this?
    - Is the CPU or the disk in the server too slow?
    - Or maybe the network connection?
    - What about memory usage?
- Assuming I get new customers in the future, how many customers can my application handle before the response time becomes too high?



In this project, you will analyze the performance (e.g. response time) of a simple client-server application that you will implement in C. The client requests an encrypted file from the server. The files are of fixed size, given as an argument to the server. In practice, you will pre-generate 1000 files with the sizes of the file taken in the argument. They will be stored in volatile memory. The size of the file will always be the square of a power of 2, enforce this in the parameter parsing. The client gives the index of a file (0 to 999 included) and the key. The key will be used to encode the file with block-by-block matrix multiplication. For instance, with a key of size 2x2 and a file of size 4x4:

Key:

A	B
C	D

File:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Encrypted File:

A*1+ B*5	A*2+ B*6	A*3+ B*7	A*4+ B*8
C*1+ D*5	C*2+ D*6	C*3+ D*7	C*4+ D*8
…			

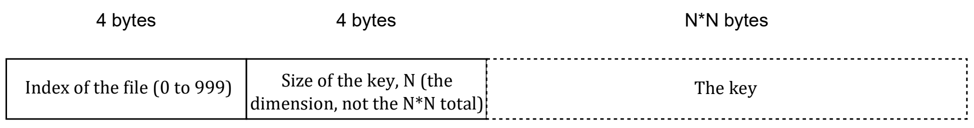
You will multiply the key with the 4 coloured boxes using matrix multiplication. The key size must divide the file size, up to the file size itself. Enforce the rule, returning error codes if it is not the case. Therefore you don’t have to worry about corner cases like the key not perfectly fitting into the file.

You will use TCP sockets. The protocol works as follow. The client sends a request with the following format. The first 4 bytes are an unsigned integer which is the index of the file to receive (0 to 999). The next 4 bytes sent by the client composes a single unsigned integer, specifying the size of the square matrix to come (2 in the example). Then will follow N*N bytes (4 in the example) describing the key. Line by line and left to right.

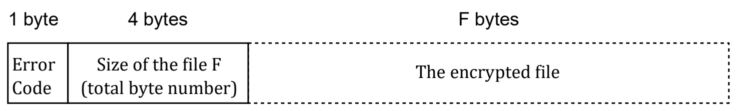
The server then returns an error code in the form of a byte, normally 0 when there’s no error. Then the file size (as a total byte number, so 16 in the example) as an unsigned 4-byte integer, and then the server closes the connection after transmitting those bytes. The server must be multi-threaded using a single socket (at least in the first phase). That is you can’t create as many threads as there are connections, use a pool of threads instead, and/or a FIFO queue.

Pay attention to the endianness. The unsigned integer fields must follow the network byte order (big-endian).

The client's request


4 bytes
4 bytes
N*N bytes
Index of the file (0 to 999)
Size of the key, N (the dimension, not the N*N total)
The key

The server's response



The project will be split into 4 phases.

- The goal of the first part of the work (now) is to measure the performance of the server in a systematic way. Do not beat yourself about the encryption in the first phase, as the second phase will be about the optimization of the processing of the first one. Follow the mathematical multiplication blocks by blocks.
- In the second phase, you will optimize the processing in a cache-aware fashion, and show how the performance of multiple versions of the algorithm can be affected by the placement of data in the cache and the instructions. You will use profiling techniques to explain these improvements.
- In the third phase, you will optimize the processing using SIMD and other instruction-level parallelism techniques. This part will be done individually.
- In the final phase, you will use queuing theory to modelize your system.

Virtual environment

You must develop and test the system on your own local machine.

However, the **FINAL** graphs must be generated on the intelXX machines of the Intel room (https://wiki.student.info.ucl.ac.be/Mat%C3%A9riel/SalleIntel). As you are not root on those machines, you cannot directly create special network devices and induce delays. The solution is to create 3 virtual machines on a single intelXX machine, a client, a server and a "router" machine connecting the 2 first machines that will simulate varying network conditions such as network delay.

All the necessary software is already installed on the Intel machines. If you want to try at home, you need the `qemu` hypervisor and `qemu-kvm` accelerator.

Download the Debian base image with:

```
wget https://cloud.debian.org/images/cloud/buster/latest/debian-10-
nocloud-amd64.qcow2
```

Create 3 overlay images for the 3 machines with:

```
qemu-img create -b debian-10-nocloud-amd64.qcow2 -F qcow2 -f qcow2
client.qcow2 20G
```

Launch the server with accelerated QEMU (with KVM, a must!) :

```
qemu-system-x86_64 -m 4096 -drive file=client.qcow2 -nographic -
enable-kvm
```

The user of the machine is root, there is no password.

Exit the VM with Ctrl + A – x

The VM server can be launched with multiple CPUs with : --smp cpus=4.

Please be carefull to killall your VMs when leaving with `killall qemu-system-x86_64` !

VM networking

Add the following parameter to each machine to create a network interface that will provide Internet access for your convenience : `-netdev user,id=mynet0  -device virtio-net-pci,netdev=mynet0`

Then, you will need to create an "internal" network. To the router machine, add 2 interfaces :

```
-netdev socket,id=left,listen=:1234 -device virtio-net-
pci,netdev=left
```

```
-netdev socket,id=right,listen=:1235 -device virtio-net-
pci,netdev=right
```

To the client machines, add :

```
-netdev socket,id=left,listen=:1234 -device virtio-net-
pci,netdev=left
```

To the server machine, add :

```
-netdev socket,id=right,listen=:1235 -device virtio-net-
pci,netdev=right
```

You now have a network with a router in between. You must set the “internal” IP addresses by yourself with “ip addr …”. You can add some delay with netem. See https://wiki.linuxfoundation.org/networking/netem .

Configure the router and gateways in a way that allows client to see server and vice versa.

Phase 1

Measure the average response time of your system and other relevant metrics you see fit. Obviously, the response time will depend on how many requests the server receives and how " difficult" the requests are.
To do measurements, you must not send requests to the server by hand. Instead, you should implement a small client application that sends requests at random times. For the inter-request time, choose some distribution you see fit. To send multiple requests to the server at the same time, you can either write a client application that opens multiple TCP connections, or the client only opens one TCP connection and you start the client application several times in parallel.
The client should not wait for a response before sending the next request. It's perfectly possible that a request is sent while another request is still being processed. Try to go up to 50 to 100 client connections. Show measurement results for combinations of

- different request rate,
- different matrix sizes

…

Think about different difficulties and how they would affect the performance of the server in different ways (e.g. more CPU intensive vs more network intensive). Plan carefully your experimental design following what we saw in the course.

Choose reasonable values for the parameters: obviously, a request rate of 1 request/day does not produce interesting response time results. Show plots and discuss the results. What is the most important factor (CPU, network, …) in the response time, depending on the chosen parameters? Here are some tools to measure the network and CPU load on Linux:

- http://www.binarytides.com/linux-commands-monitor-network/
- https://www.tecmint.com/command-line-tools-to-monitor-linux-performance/

You must use those tools inside the VMs, of course.

Expected Output

You should hand in the source code of your implementation (no binaries) on INGInious and a written report on Moodle. Both submission site will open 1~2 weeks before the deadline, as groups cannot be changed after the submission sites are open. The deadline for both is October 16th, 23:59.
The report should contain:

- a description of the implementation, including the client
- a description of your measurement setup, even if it is the Intel machines for everyone
- a description of the workload you used in your experiments, i.e. how the clients generate the requests. Include a description of the experimental design and all the variables that you thought of. The factors you selected. The one you left out and their possible impact.

Don’t forget to include your names, email addresses, NOMA and group number. Keep your report concise. Don’t waste space on long introductions, a table of contents, etc.

You must provide a Makefile that will compile the client and server with “make client” and make server”, the generated binaries will have the same names as the target.

- The command line of the client should be “./client -k 128 -r 1000 -t 10 127.0.0.1:2241” to generate requests with keys of size 128 at a mean rate of 1000 requests/seconds for 10 seconds to the server 127.0.0.1 on TCP port 2241.
- The command line of the server should be “./server -j 4 -s 1024 -p 2241” to open the server with 4 threads and pre-generate files of 1024*1024 bytes, listening on the TCP port 2241.

If you think something is missing, discuss it on the forum.

Evaluation Criteria for the first part

- Quality of code (comments, correctness, etc.)
- Quality of report (reasoning, language, readability, clearness of presentation)
- Quality of the experimental design. Workload description. Choice of variables, choice of factors. Explanation of the limits of the measurements. (I.e. : What did you not evaluate? What design choices affect the performance?)
- Quality of the measurements. Choice of visualization, choice and correctness of summarization