ECE 315 Assignment 1

Arun Woosaree
XXXXXXX

February 3, 2020

1

Hard real-time vs. Soft real-time embedded systems

Similarities

- With both, we want to avoid violating real-time constraints

Differences

- Violations of real-time constraints are undesireable, but tolerated in soft real-time systems while in hard real-time systems, violations of real-time constraints are unacceptable (a violation for a hard real-time system would cause catastrophic failure, or even death.)

When selecting an implementation strategy, it is useful to distinguish between hard and soft real-time systems because you want to select a strategy that is appropriate. You want to consider what might happen in the case of failure to meet real-time constraints. If a violation of these constraints would cause catastrophic damage, such as loss of life, you have a hard real-time system, so your implementation strategy should take that into account. If you have a soft real time system, and failure to meet the real-time constraints is not catastrophic, your implementation strategy does not need to avoid failure to the same extent as it would need to for a hard real-time system.

For example, you probably would have a different implementation strategy for making a pacemaker versus a vending machine. In one scenario, a person's life is dependent, while in the other, a person may or may not get a snack.

2

2.1 Cloud Computing

As per ISO/IEC 17788:2014 :

> “ Cloud computing is a paradigm for enabling network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on-demand. The cloud computing paradigm is composed of key characteristics, cloud computing roles and activities, cloud capabilities types and cloud service categories, cloud deployment models and cloud computing cross cutting aspects that are briefly described in [clause 6 of ISO/IEC 17788:2014]. ”

Advantages:

- The cloud computing provider takes care of purchasing and maintaining hardware, not the user
- Allows for flexibility. The user can scale services as they need on-demand. That way, a user only pays for the resources they use. For example, the user can request more computing resources when their service has more traffic, but when there is less traffic, the user can scale down their operations to save cost
- Reduces barrier of entry for deploying an application. One can deploy a toy project they've been working on for a few dollars, without investing in the up-front cost of purchasing a mainframe, maintaining it, and selling the computer when they're done.

Overall, I think this term is extremely well-defined, considering there is an ISO specification that is 10 pages long, and is very thorough in defining everything. It is, after all the international body for defining standards. . .

2.2 Edge Computing

According to Cloudflare:

> “ Edge computing is a networking philosophy focused on bringing computing as close to the source of data as possible in order to reduce latency and bandwidth use. In simpler terms, edge computing means running fewer processes in the cloud and moving those processes to local places, such as on a user's computer, an IoT device, or an edge server. Bringing computation to the network's edge minimizes the amount of long-distance communication that has to happen between a client and server. ”

Advantages:

- Lower latency (better user experience, streaming and gaming applications for example. Users would experience higher quality streams and faster response times for games)
- Less bandwidth and server resource usage, and therefore lower costs associated with these resources

Overall, it is not very specifically defined. However, most sources seem to agree that the general idea is moving cloud computing resources geographically closer to users.

2.3 Fog Computing

According to the paper "Finding your way in the fog: Towards a comprehensive definition of fog computing" by L. M. Vaquero and L. Rodero-Merino:

> "Fog computing is a scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralised devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of thirdparties. These tasks can be for supporting basic network functions or new services and applications that run in a sandboxed environment. Users leasing part of their devices to host these services get incentives for doing so."

Advantages:

- even lower latency and utilizing resources like bandwidth more efficiently compared to edge computing. It brings the edge closer to the user, kind of like how fog is closer to the ground than clouds
- allows for data to be sent to a local node for faster proccessing and response times. An example application would be sensors in a factory which need to automatically shut down equipment if something goes wrong

Overall, this term is fairly well-defined. There are multiple papers, IEEE articles and the like which give in-depth definitions of fog computing which seem to agree with each other.

2.4 Mist Computing

According to the National Institute of Standards and Technology,

> "Mist computing is a lightweight and rudimentary form of fog computing that resides directly within the network fabric at the edge of the network fabric, bringing the fog computing layer closer to the smart end-devices. Mist computing uses microcomputers and microcontrollers to feed into fog computing nodes and potentially onward towards the centralized (cloud) computing services."

Advantages:

- similar to how fog computing takes edge computing a step further, mist computing takes it a step further to reduce latency, and better use resources like processing power

- no need to connect to the 'cloud' to perform decisions, but overall telemetry data can still be aggregated by connecting to the broader internet Allows for applications like self-driving cars to make decisions locally where decisions have to be made quickly.

Overall, this term is not well-defined. Finding a definition was difficult to begin with. Additionally, the sources that mention mist computing seem to talk more about fog computing than mist computing. However, generally, they agree on the local processing aspect of mist computing.

3

1. Military applications. Say for example, a missile was being launched, and is guided by a computer system. When a missile is launched there were (hopefully) multiple humans who made the decision to do so, and they were sure of their actions. If the system was able to be overridden by humans, that gives the enemy a chance to disarm or take control of the missle, which is not ideal. In this case, the missile should lock out human and override human control to deliver its payload to the intended recipient.
2. Antilock braking systems. Humans are not great at pressing on their car's brakes just enough so that there is static friction, as opposed to kinetic friction. If static friction is maintained, the car can come to a stop quicker and also prevents the car from skidding so that the driver can steer the car more easily and potentially avoid a collision. In this case, the computer forces the brake pedal back up, overriding the human's action of pushing the pedal down. This is a feature that has saved many lives.
3. Safety systems. Imagine a computer at a manufacturing plant which has shut down operations due to one of its sensors reporting a value which results in an unsafe condition. An uninformed worker, or maybe a malpracticing human prioritizing production over safey might try to resume operations without fixing the cause of production being shut down. In this case, it makes sense for the computer to lock out human control for resuming operations until the safety issue is fixed.

4

A pointer can be created in the following ways:

1. You can ask the OS to allocate a region of memory using `malloc()`, `calloc()`, or `realloc()`, and give you a pointer to that region of memory. It is up to the programmer to check if NULL was returned and deal with error handling.
2. The `&` operator is used to obtain a pointer to an existing piece of data

3. math is done on an existing pointer, and the result is stored as a new pointer. (e.g. `char* new_ptr = other_ptr + 8`)
4. A pointer can be declared but uninitialized. A so-called wild pointer can point to anywhere in memory and therefore should not be used. A pointer should be initialized with a value before use.

Throughout a pointer's life, it can be modified and overwritten. An example of a pointer being modified would be a typed pointer being incremented or decremented before or after they are used. (e.g. `*(ptr++)`. Void pointers, which can be used to point to anything can be casted to a different type before being modified (e.g. `ptr = (char* ptr)`) Also, a pointer can be overwritten by simply assigning another pointer value to the existing pointer.

Once the block of memory that a pointer points to is freed using the `free()` function, the pointer can be safely discarded. Actually, the pointer should no longer be used, because the pointer now points to a region of memory that

5

6

7

8

9