**MicroService:**

Microservice is a new way of writing big applications where the whole service is divided into small independent services. Just like in our OS which does multiple tasks. Like OS, the task can be completely different in microservices. In OS we can run web, word, run the visual studio, listen to music all at the same time and all those programs share memory and CPU. The same does the services in a microservice architecture. Like as we know different programs running in our computer can have multiple processes, and those processes would be independent of Each other; the same can happen in a microservice architecture. In Microservice we can define framework, languages, HTTP request, business logic, mapping as a different process. Some have a long life while some have short. My analogy is described more in detail below.

**Shared Resources**

**Long Term Storage:** In the computer System, different processes may have to access data from storage. In Web system a lot of our Http requests, depending on the requests, need to access data from the database. Lots of different requests can access a database at the same time. Context makes sure that the database is not changed while another request in accessing or writing data on the database. It is just liking avoiding the race condition in the OS.

**I/0:** Like OS, in the web system we get our data/request from the user. Data can be some form while requests can be a mouse click or some command. And the output can be rendering a page that may or may not use database services.

**Communication:** In OS there are lots of processes and they must communicate with each other as well as outside devices. In our Microservice, we represent different services as a program, and the process happening inside the (microservices/program/services) as a process in OS. There are different tools in microservice that helps different (microservice- process/process) communicate with each other. Microservice-process(MP) communicates with a database with the help of context, MP communicates with the user with the help of a controller, communicates with other MP with the help of API and Mapper depending on if the MP is in same service or different services just like a process in OS.

**CPU:** Likewise processes do lots of computing in OS. Our process in Application needs to carry different business computations. Sometimes one request may need to use 10 context/database. And calculate the result. Another call from the different users also needs to be processed and it may be using one, none, or many database and context. There may be millions of users and each may have fired a call. And all the processing happens in the server, but they are all independent. It is just like we see in the OS, process sharing CPU resources.

**Memory:** In the context of Microservice I will relate memory with the package. It is very useful and used by every object in the server which needs to access all the time. As said in the book in section 3.3.3 the parent and child need to share the same memory space; the same can be related in the microservice. The state of the package must be maintained when we transfer the state/object-state from one stage/process/MP to another.

**Process Lifecycle:**  In our analogy, we have different processes. We have processes with a very long life like the framework to the processes like very short life like the mapper, context-caller, controller, request, service/business logic, etc. These all have their life and one is created by others or is dependent on others. Not every process has to be dependent; two HTTP requests fired from different microservice or even the same microservice do not have to be dependent and talk to each other. Framework life is long, in a program one version of the framework, languages can be installed for a long time as compared to a single HTTP request. But someday both have to be updated and old will die. The new framework may request differently. Likewise, one request can generate different processes like mapping, reading database, writing and these all will die after a short time. It is more like a short term and long term processes.

**Scheduling:** From our book in section 3.2.2 we have found there are 3 types of schedule. A long, short and medium. We have also differentiated the processes in microservices into three types. For long term-scheduling processes, we kept our HTTP request and business logic which are called very few times as compared to the business logic that are auto called like auto-refreshing pages which are our short-term scheduled process. In the long term, the HTTP request is our input bound process and business logic is our computational bound process. For middle-term scheduling, we have kept out save in cookies or draft function. The draft may or may not be available based on service, but cookies are always available. These cookies are loaded when you visit the service next time this perfectly matches the swapping functionality in medium-term scheduling.

**IPC:**  In the Computer system process need to share resources for various reasons[1]. Same in the microservices. Even though microservices are independent there may be a situation where they must talk to each other or even share a copy of the database. By copy I mean services will have their own database, but they might have to share sometimes. There is a different technique of sharing, but this type of sharing is more related to shared memory in a database. In our analogy, we share not an only the database but also the same framework and language. But there is another situation that will not work by sharing a common database image. What if another independent service is written in python and has a no-SQL database. Then things might be different. Then we need an API calling technique that can be related to message passing. Even though message passing does not provide better performance in terms of computing, but it provides better performance in the working team. So, each team can work in their preferred stack. In OS shared memory suffers from coherency same happens with sharing a copy of the database among services. It breaks the policy of independent service and causes overload on mapping.

The backend of Web or any Application - System

Microservices Environment – OS

Web Server CPU – CPU of System

Different independent micro-services – Applications

.net framework- process / parent

.net core- process / child

Entity core – process/child

Entity framework – process/parent

Azure-Database – long term storage

Packages – Memory

Draft storage – Swapping / Medium-term scheduler

Http call: Long term scheduler

Service and business logic – CPU-bound Process

Accessing database and mapping – I/0 bound process

Short term process – Auto Refreshing page and context based on the status of (Entity=>Entity = Class representing database and its properties)

API call => Message Passing

Sharing image of common database => Shared Memory

[1] aA. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts essentials*. Hoboken, NJ: John Wiley & Sons, 2014.