



## **Faculty of Computers and information**



### **“Squad RTOS”**

### **Graduation Project**

**Student's names**

**Mahmoud Abou-Hawis**

**Adham Emad**

**Rawan Emad**

**Ali Ashraf**

**Esraa Elzebedy**

**Sohaila khalaf**

**Ahmed Ehab**

**Sohaila Mohamed**

**Under supervisor**

**Prof. Dr.Omina El-Barbary**

**July 2023**

**Content:**

<b>Abstraction.....</b>	<b>4</b>
<b>Chapter 1: Introduction.....</b>	<b>5</b>
1.1 What is an operating system?.....	6
1.2 What is a Real time operating system (RTOS)?.....	6
1.3 Difference between RTOS and OS.....	7
1.4 Types of Real Time Operating Systems.....	7
<b>Chapter 2: Real Time Operating System.....</b>	<b>9</b>
2.1 What is Squad RTOS?.....	10
2.2 What is Assembly?.....	10
2.3 What is a Task?.....	12
2.4 What is Super loop? And why are we using it?.....	12
2.5 Interrupts usage.....	14
2.6 Why are we making Real-Time Operating system (Squad RTOS vs Super loop and Interrupts)?.....	18
2.7 Kernel.....	21
2.7.1 Dispatcher.....	21
2.7.2 Scheduler.....	22
2.8 Interrupt handling.....	25
2.9 Memory management.....	27
2.10 How we made Squad RTOS.....	29
<b>Chapter 3: Communication.....</b>	<b>32</b>
3.1 What is an IP address?.....	33
3.2 IP versions.....	32
3.2.1 IPv4.....	35
3.2.2 IPv6.....	36
3.3 OSI and TCP/IP models.....	39
3.3.1 What is OSI model?.....	39
3.3.2 TCP/IP model.....	40
3.3.3 Reasons why OSI model is being used.....	41
3.3.4 OSI layers.....	44
3.4 What is LwIP?.....	51
3.5 UART.....	56
3.5.1 What is UART?.....	56
3.5.2 What is data transmission?.....	57
3.5.3 How does transmission work in UART?.....	58
<b>Chapter 4: Desktop Application.....</b>	<b>61</b>

<b>4.1 Introduction.....</b>	<b>62</b>
<b>4.2 Languages used in Desktop application.....</b>	<b>63</b>
<b>4.2.1 Python.....</b>	<b>63</b>
<b>4.2.1.1 Brief history about Python.....</b>	<b>63</b>
<b>4.2.1.2 How did we use Python to build our Desktop application.....</b>	<b>65</b>
<b>4.3 Libraries used in the Desktop application.....</b>	<b>67</b>
<b>4.3.1 TKinter.....</b>	<b>67</b>
<b>4.3.2 Custom TKinter.....</b>	<b>67</b>
<b>4.3.3 Python Imaging library.....</b>	<b>68</b>
<b>4.3.4 Time library.....</b>	<b>68</b>
<b>4.4 Desktop design.....</b>	<b>69</b>
<b>4.5 APIs (Application Programming Interface) .....</b>	<b>71</b>
<b>4.5.1 what is an API.....</b>	<b>71</b>
<b>4.5.2 why APIs are useful?.....</b>	<b>73</b>
<b>4.5.3 Requests.....</b>	<b>75</b>
<b>4.5.4 OTP Service.....</b>	<b>78</b>
<b>Chapter 5: Security.....</b>	<b>85</b>
<b>5.1 Security Concepts and their importance.....</b>	<b>86</b>
<b>5.2 Different ways of providing security.....</b>	<b>87</b>
<b>Chapter 6: Webpage .....</b>	<b>89</b>
<b>6.1 What is webpage .....</b>	<b>90</b>
<b>6.2 Languages used to create a Web Page.....</b>	<b>90</b>
<b>6.2.1 JavaScript.....</b>	<b>90</b>
<b>6.2.2 CSS.....</b>	<b>91</b>
<b>6.2.3 HTML.....</b>	<b>94</b>
<b>6.3 Content of our Web page.....</b>	<b>96</b>
<b>6.3.1 Login Page.....</b>	<b>96</b>
<b>6.3.2 Face ID page.....</b>	<b>97</b>
<b>6.3.3 Home Page and its services.....</b>	<b>99</b>
<b>6.3.4 LED.....</b>	<b>100</b>
<b>Chapter 7: Conclusion and Future work.....</b>	<b>101</b>

## Abstraction

An operating system (OS) for real-time computing applications that processes data and events with precisely defined time limits is known as “RTOS”. RTOS is different from a time-sharing operating system, like Unix, which controls how system resources are shared in a multitasking or multiprogramming environment using a scheduler, data buffers, or fixed task prioritization. It is important to completely comprehend, and limit processing time needs rather than merely keeping them to a minimum. All processing must take place inside the established boundaries. Event-driven and preventive systems are real-time operating systems. Implying that the OS may adjust the task priority while keeping track of the pertinent priorities of competing tasks. Time-sharing systems swap the job based on clock interruptions, whereas event-driven systems switch the task depending on their priorities.

The production, gathering, and analysis of exponentially more data is being driven by increasing global connectivity, shifting consumer needs for always-available data, and always-on, sensor-enabled workplace systems. IDC predicts that 79.41 zettabytes of data will be generated by 2025, and that approximately 30% of it would need real-time processing made possible by real-time systems.

For companies in robotics, manufacturing, healthcare, and high-precision industries like oil and gas and power that depend on real-time data for continual improvement in safety, efficiency, and dependability, real-time processing is extremely important.

The capacity of a system to manage, prioritize, and execute real-time workloads over non-real-time workloads is a critical component in assuring data processing in real-time for companies in these sorts of sectors.

For instance, modern automakers rely heavily on robots to collaborate on a manufacturing line and construct cars. Robots will exchange components, drill, or weld, or conduct safety inspections—all tasks that call for extreme accuracy and exact time. In this use scenario, a real-time system must be able to process data in a set, predictable period as well as guarantee that important activities, such those relating to safety, are finished before less important ones.

# Chapter 1: Introduction

## 1.1 What is an operating system?

A program that controls computer hardware or acts as an interface is known as an operating system. An operating system loads during the startup phase when you switch on a computer and starts sending orders to various parts.

An operating system (OS) not only enables communication between computer parts but also serves as the user interface. There are other kinds of operating systems, including embedded OS created for dishwashers, HVAC systems, automobiles, and ATM machines. While most people are aware of operating systems used on home computers or game consoles, there are other sorts of operating systems.

Many people think that the first microcomputer, the MITS Altair, was the first to have an operating system. Contrary to popular belief, operating systems have a long history dating back to the 1950s, long before solid-state drives and CD-ROMs were commercially accessible.

## 1.2 What is a Real time operating system (RTOS)?

RTOS stands for real-time operating system, which is a specialized operating system that is designed to provide a reliable and predictable platform for embedded systems that require precise timing and responsiveness.

Unlike general-purpose operating systems, which are designed to provide a wide range of features and functionality, RTOS is designed to prioritize real-time performance and ensure that critical tasks are given priority over non-critical tasks. This is achieved through features such as preemptive multitasking, task scheduling, and interrupt handling.

RTOS is commonly used in applications such as robotics, control systems, and automotive electronics, where precise timing and responsiveness are critical for the proper functioning of the system. It provides a highly customizable and efficient platform for developing embedded systems, with features such as low-level hardware access, optimized algorithms, and comprehensive libraries and tools.

Overall, RTOS is a powerful and specialized operating system that provides a reliable and predictable platform for embedded systems that require real-time performance.

### 1.3 Difference between RTOS and OS:

- While general-purpose operating systems (OS) prioritize the performance of the entire system, real-time operating systems (RTOS) prioritize time-sensitive tasks and ensure that they are completed within predetermined deadlines.
- RTOS supports applications in crucial sectors including aerospace, medical technology, and automotive systems, whereas general-purpose OS supports servers and home computers.
- In contrast to general-purpose OS, which utilizes non-deterministic scheduling methods, RTOS uses deterministic scheduling techniques to ensure that time-critical operations run consistently.

An operating system called a real-time operating system (RTOS) is employed in systems that offer immediate answers to operational problems. It is an operating system used for time-sensitive real-time computing applications. An operating system is a piece of software that manages the hardware and software resources of a computer. It handles input and output and manages files, among other fundamental duties.

The name "RTOS" refers to "real-time operating system," emphasizing this feature: To control planning, RTOS may successfully handle interruptions using priority-based functioning. Contrary to a broad sense OS, an RTOS must meet computational deadlines regardless of how dire the situation is.

A crucial element of an RTOS is its dependability regarding how long it takes to receive and execute a job from an application; this unpredictability is known as "jitter." Alternatively, OS stands for operating system.

The operating system of the entire computer is the most important application that runs on it. Every device that has a CPU and GPU comes with one or more operating systems.

### 1.4 Types of Real Time operating system (RTOS)

#### Hard real-time systems and Soft real-time systems

Hard real-time systems and soft real-time systems are two categories of real-time systems that are differentiated by their degree of tolerance for missed deadlines.

Hard real-time systems are designed to be extremely time-sensitive and must meet strict timing constraints. Failure to meet these constraints can result in catastrophic consequences, such as system failure or loss of life. Examples of hard real-time systems include aerospace control systems, medical devices, and automotive safety systems. In hard real-time systems, missing a deadline can result in system failure, making it critical to ensure that all tasks are completed within the specified time limit. Soft real-time systems, on the other hand, have less strict timing constraints, and missing a deadline may not have catastrophic consequences. Examples of soft real-time systems include multimedia applications, such as video and audio streaming, where occasional delays may be acceptable, but consistent delays can result in a degraded user experience.

### Difference between hard and soft RTOS

HARD REAL TIME OPERATING SYSTEM	SOFT REAL TIME OPERATING SYSTEM
1. Hard response time is required.	1. Soft response time is required
2. Size of data file is small or medium	2. Size of data file is large.
3. It have little laxity and generally provides full deadline compliance	3. It is more flexible and have greater laxity and can tolerate certain amount of deadline misses
4. Safety critical systems are typically hard real system.	4. Linux is an example of softRTOS

**Table 1.1 Difference between hard and soft RTOS**

Overall, the distinction between hard real-time systems and soft real-time systems is based on the degree of tolerance for missed deadlines, with hard real-time systems requiring strict adherence to timing constraints, while soft real-time systems can tolerate occasional delays. [1]



# **Chapter 2:**

# **Squad RTOS**

## 2.1 What is SQUAD RTOS?

Squad RTOS is a real-time operating system designed to provide a reliable and efficient platform for embedded systems. It is specifically tailored for small to medium-sized embedded systems, and its highly customizable architecture can be adapted to meet the specific needs of each project.

One of the key features of Squad RTOS is its preemptive multithreading kernel, which allows multiple tasks to run concurrently while ensuring that critical tasks are given priority over non-critical tasks. This makes it an ideal choice for applications that require precise timing and responsiveness, such as robotics, control systems, and automotive electronics.

Another advantage of Squad RTOS is its comprehensive set of libraries, drivers, and tools that simplify the development process. This helps to reduce development time and costs, while improving the quality and reliability of the final product.

Overall, Squad RTOS is a powerful and flexible real-time operating system that provides an efficient and reliable platform for embedded systems. Its customizable architecture, real-time capabilities, and development tools make it an ideal choice for a wide range of applications in various industries, including aerospace, medical devices, and industrial automation. With its proven track record of success, Squad RTOS is a trusted and reliable choice for developers looking to build high-performance embedded systems.[1]

## 2.2 What is Assembly?

Assembly language, also known as assembler language or symbolic machine code, is any low-level programming language with a close resemblance to the architecture's machine code instructions. It is frequently referred to simply as Assembly and is frequently abbreviated as ASM or asm. One statement per machine instruction (1:1) is the norm for assembly language, however constants, comments, assembler directives, symbolic labels for things like memory locations, registers, and macros are often also available.

The 1947 book Coding for A.R.C. by Kathleen and Andrew Donald Booth contains the first assembly code in which a language is used to represent machine code instructions. An assembler is a tool that transforms assembly code into machine code that can be executed. However, Wilkes,

Wheeler, and Gill used the term to mean "a program that assembles another program consisting of several sections into a single program" in their 1951 book *The Preparation of Programs for an Electronic Digital Computer*, which is generally where the term "assembler" originates from. Assembling refers to the conversion procedure much as it does to assembling source code. Assembly time is the computing phase that occurs when an assembler processes a program.

Each assembly language [nb 1] is unique to a certain computer architecture since assembly depends on the machine code instructions.

Sometimes there are many assemblers available for a given architecture, and other times one assembler is only compatible with a certain operating system. Most assembly languages don't have a specific syntax for operating system calls, and they can be used with any operating system [nb 2] because they give access to all the processor's actual capabilities, which are ultimately what all system call mechanisms depend on. While most high-level programming languages require interpreting or compiling, which are much more difficult processes than assembling, they are typically portable across various architectures in contrast to assembly languages.

Both systems programming and application programming were frequently carried out fully in assembly language during the early years of computers. While still indispensable for some tasks, higher-level interpreted and compiled languages are now used for most programming. The consequences of moving away from assembly language programming were summed up by Fred Brooks in "No Silver Bullet" as follows: "Surely the most effective stroke for software productivity, dependability, and simplicity has been the increased adoption of high-level languages for programming. Most observers attribute that progress to advances in dependability, simplicity, and understandability that increase production by at least a factor of five.

For performance reasons or to connect directly with hardware in ways the higher-level language does not provide, it is common practice today to employ tiny amounts of assembly language code within larger systems developed in higher-level languages. For instance, less than 2% of the Linux kernel source code in version 4.9 is written in assembly, whereas more than 97% of it is written in C.

### 2.3 What is a task?

In Squad RTOS (Real-Time Operating System), a task refers to a unit of execution that can run independently and concurrently with other tasks. A task is essentially a piece of code that can perform a specific function, such as reading data from a sensor, processing a set of instructions, or generating an output signal.

Tasks are created by the application and managed by the Squad RTOS kernel. Each task has its own stack and execution context and can be scheduled to run at a specific time or in response to an event or interrupt. Tasks can communicate with each other through message passing, semaphores, or other synchronization mechanisms.

One of the key benefits of using tasks in Squad RTOS is that they allow for efficient use of system resources by enabling concurrent execution of multiple tasks. This can help improve system responsiveness and reduce latency in real-time applications.

Tasks in a Squad RTOS can have different priorities, which determine their order of execution. Tasks with higher priorities are executed first, and lower priority tasks are executed only when higher priority tasks are blocked or suspended.

Overall, tasks are a fundamental concept in Squad RTOS programming, and understanding how to create, manage, and schedule tasks is essential for developing efficient and responsive real-time applications.[10]

### 2.4 What is Super loop and why are we using it?

A super loop is a simple programming structure that can be used to create tasks and manage their execution. A super loop consists of an infinite loop that repeatedly executes a sequence of tasks, each of which performs a specific function.

To implement tasks with a super loop, you can define each task as a separate function, and then call these functions from within the super loop. Each task can be executed sequentially, one after the other, or concurrently, by using interrupts or other synchronization mechanisms.

Here's a simple example of how to create tasks with a super loop:

```

void task1(void) {
    // Code for task 1
}
void task2(void) {
    // Code for task 2
}
void main(void) {
    while (1) {
        task1(); // Execute task 1
        task2(); // Execute task 2
    }
}

```

**Code 2.1 showing how simple Super loop works.**

In this example, we have defined two tasks, `task1()` and `task2()`, which are called sequentially from within the super loop in the `main()` function. Each task can perform a specific function, such as reading data from a sensor, processing a set of instructions, or generating an output signal.[10]

Note that in this example, the tasks are executed sequentially, which means that `task2()` will not be executed until `task1()` has completed. If you want to execute tasks concurrently, you can use interrupts or other synchronization mechanisms to ensure that each task is executed at the appropriate time.

The super loop structure is a simple and flexible way to create tasks in an embedded system and can be easily adapted to suit a variety of applications. However, it may not be suitable for more complex systems that require advanced scheduling or synchronization mechanisms. In such cases, a Squad RTOS may be a better option for managing tasks and ensuring system responsiveness.

Sure, I'd be happy to discuss the advantages and disadvantages of using a super loop in an embedded system.

### **Advantages of a Super Loop:**

1. **Simplicity:** A super loop structure is easy to understand and implement, making it a popular choice for small and simple embedded systems.
2. **Efficiency:** A super loop can be very efficient, as it avoids the overhead of task switching and context switching that is required in more complex scheduling mechanisms.
3. **Flexibility:** The structure of a super loop is flexible, allowing the programmer to easily add, remove or modify tasks as needed.

### Disadvantages of a Super Loop:

1. Limited concurrency: A super loop structure can only execute one task at a time, which can limit the concurrency and responsiveness of the system.
2. Lack of priority: A super loop does not provide a mechanism for task prioritization, which can be important in systems that require real-time performance.
3. Difficulty in managing timing: A super loop structure can make it difficult to manage the timing of tasks, especially if the tasks have different execution times or require precise timing.
4. Lack of synchronization: A super loop structure does not provide a mechanism for task synchronization, which can be important in systems that require coordination between tasks.

Overall, the choice of whether to use a super loop structure in an embedded system depends on the specific requirements and constraints of the system. While a super loop can be simple and efficient, it may not be suitable for more complex systems that require advanced scheduling or synchronization mechanisms. In such cases, RTOS or other more advanced scheduling mechanisms may be a better option.

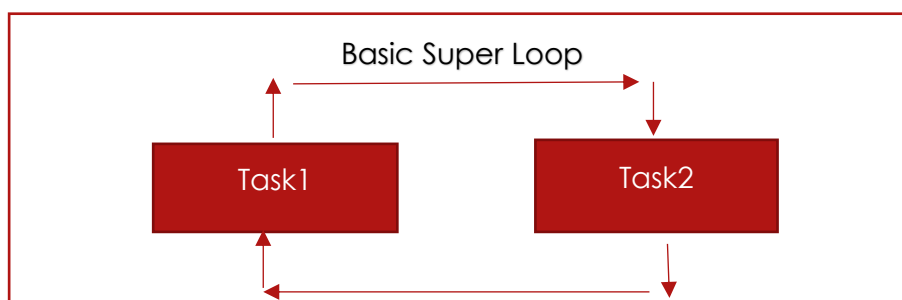


Fig 2.1 showing Basic Super loop.

### 2.5 Interrupt usage

Programming using interrupts is a technique commonly used in embedded systems to handle events and tasks in a timely and efficient manner. Interrupts are signals that are generated by hardware or software events and cause the processor to temporarily suspend the current task and execute a specific interrupt service routine (ISR) that handles the event.

To program using interrupts, the programmer needs to define the ISR for each event and register it with the system. When the event occurs, the

processor interrupts the current task, saves its context, and executes the ISR to handle the event. Once the ISR is completed, the processor restores the context of the interrupted task and resumes its execution.

Programming using interrupts has several advantages over other techniques, such as the super loop. Interrupts provide faster response times and more efficient use of system resources, as the processor can perform other tasks while waiting for an event to occur. Interrupts also allow the system to handle multiple events simultaneously, making it more scalable and adaptable to a wider range of applications.

However, programming using interrupts also requires a higher level of expertise and attention to detail, as the programmer must ensure that the ISR is properly designed, tested, and integrated into the system. Improperly designed or poorly implemented ISRs can lead to performance issues, system crashes, or other unexpected behaviors.[10]

#### **Advantages of interrupt usage:**

- 1- Faster response times: Interrupts provide faster response times to events than other programming techniques, such as the super loop. This is because the processor can immediately suspend the current task and execute the interrupt service routine (ISR) to handle the event, without having to wait for the next iteration of the main loop.
- 2- Efficient use of resources: Interrupts allow the processor to perform other tasks while waiting for an event to occur, which makes better use of system resources and reduces the overall processing time.
- 3- Scalability: Interrupts can handle multiple events simultaneously, making them more scalable and adaptable to a wider range of applications.
- 4- Deterministic behavior: Interrupts provide deterministic behavior, which means that the response time of the system can be accurately predicted and controlled.

#### **Disadvantages of interrupt usage:**

- 1- Complexity: Programming using interrupts is more complex than other programming techniques, such as the super loop. This is because the programmer must design, test, and integrate the ISR into the system, which requires a higher level of expertise and attention to detail.

2- Debugging: Interrupt-driven programming can be difficult to debug since the execution order of the tasks is not determined by the programmer. This can make it harder to isolate and fix bugs in the code.

3- Overhead latency: Interrupts have an overhead latency, which is the additional time taken to perform tasks that are not directly related to their main code execution, such as context switching and interrupt handling. This overhead latency can impact the overall performance and responsiveness of the system.

4- Unpredictability: The execution order of the tasks is determined by the events that trigger the interrupts, which can make the behavior of the system harder to predict and control. This can lead to unexpected or undesirable

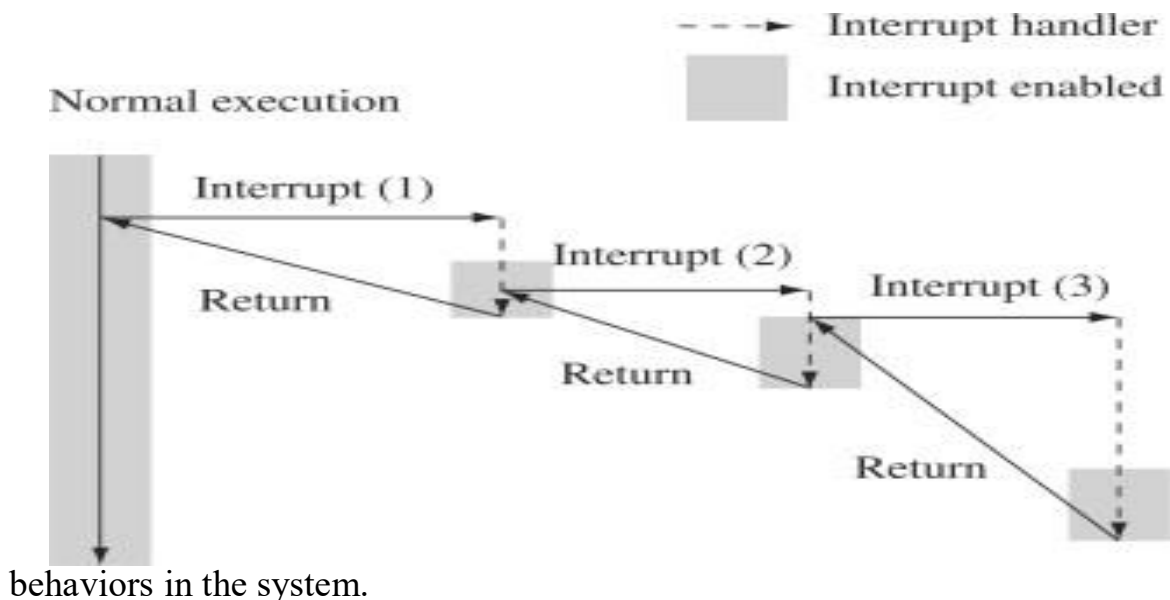


Fig 2.2 showing how interrupts work.

Making system using super loop and interrupts:

To achieve concurrent execution of tasks, you can also use interrupts to trigger the execution of specific tasks in response to external events or input signals.



```

void task2(void) {
    // Code for task 2
}

void isr1(void) {
    // Call task 1
    task1();
}
void isr2(void) {
    // Call task 2
    task2();
}
void main(void) {
    // Register ISR 1 for interrupt source 1
    register_isr(isr1, INTERRUPT_SOURCE_1);

    // Register ISR 2 for interrupt source 2
    register_isr(isr2, INTERRUPT_SOURCE_2);

    // Enable interrupts
    enable_interrupts();

    while (1) {
        // Super loop
    }
}

```

**Code 2.2 showing the implementation of super loop with interrupts to allow the system to finish multiple tasks simultaneously.**

To implement tasks with a super loop and interrupts, you can define each task as a separate function, and then register these functions as interrupt service routines (ISRs) for the corresponding interrupt sources.

When an interruption occurs, the ISR is automatically executed, allowing the associated task to be performed in a timely and efficient manner.

Here's a simple example of how to create tasks with a super loop and interrupts:

In this example, we have defined two tasks, `task1()` and `task2()`, and two ISRs, `isr1()` and `isr2()`, which are registered for the corresponding interrupt

sources. When an interrupt occurs, the associated ISR is automatically executed, allowing the corresponding task to be performed.

Note that in this example, the super loop is empty, because the tasks are executed in response to interrupts. The super loop may still be used to perform other tasks that do not require interrupt-driven execution.

The combination of a super loop and interrupts provides a simple and efficient way to create tasks in an embedded system that requires concurrent execution. However, it may not be suitable for more complex systems that require advanced scheduling or synchronization mechanisms. In such cases, a Squad RTOS may be a better option for managing tasks and ensuring system responsiveness.

### **Advantage of ISR:**

Interrupts allow for a more complex programming model, which can handle a greater number of events and tasks, without overloading the processor. This can lead to more efficient and responsive systems, especially in larger and more complex embedded systems.

### **Disadvantage of ISR:**

However, the increased complexity of interrupt-driven programming can also make it more difficult to debug and maintain the code. This is because the execution order of the tasks is determined by the events that trigger the interrupts, which can be unpredictable and harder to understand than the deterministic execution order of the super loop.

On the other hand, the super loop provides a simpler programming model that is easier to understand and maintain, especially for small and less complex systems. However, this simplicity comes at the cost of reduced efficiency and scalability, which can limit the performance and responsiveness of the system as it grows in complexity.

## **2.6 Why are we making Real-Time Operating system (Squad RTOS vs Super loop and Interrupts)?**

In the context of embedded systems development with Squad RTOS, real-time operating systems (RTOS) offer several advantages over super loops and interrupts. RTOS provides a preemptive multitasking kernel, allowing multiple tasks to run concurrently while ensuring that critical tasks

are given priority over non-critical tasks. This makes it ideal for applications that require precise timing and responsiveness.

Super loops, on the other hand, can be simple to implement but may not be able to handle complex systems with multiple tasks. Interrupts can be used to handle events but may result in high system overhead and increased complexity.

Squad RTOS provides a highly efficient and reliable real-time operating system that offers minimal system overhead and simplified development. This includes features such as optimized algorithms, low-level hardware access, and full control over system resources.

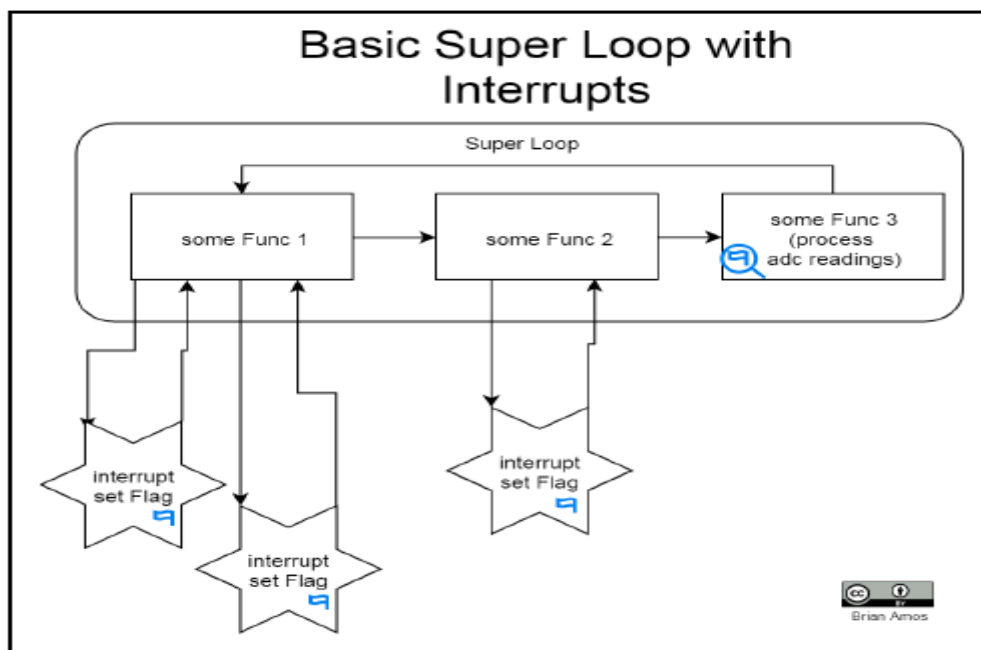


Fig 2.3 showing how a super loop and interrupt are used together.

**The main differences between tasks in Squad RTOS and super loops and interrupts are as follows:**

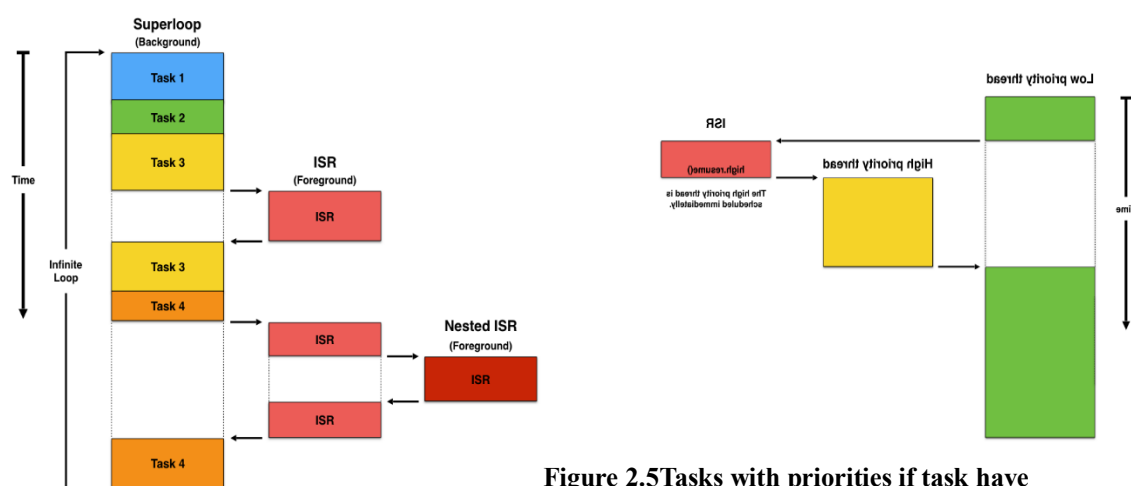
1. Private Stack (Tasks isolated): Each task is assigned its own private stack, which is not shared with any other task in the system. This allows each task to have its own call stack without interfering with the execution of other tasks, unlike a super loop which shares the system stack.
2. Priority Assigned: Each task is assigned a priority, which enables the scheduler to make decisions on which task should be running at any given time. The goal is to ensure that the highest priority task in the system is always doing useful work.

Advantages of Interrupts and super loop working together:

- 1- Memory protection: Each task's private stack is protected from other tasks in the system, preventing memory corruption and other errors that can occur when multiple tasks share the same stack.
- 2- Improved robustness: By having its own stack, each task is less likely to be affected by stack overflows or other stack-related errors that can occur when multiple tasks share the same stack.
- 3- Better context switching: Context switching between tasks is faster and more efficient when each task has its own stack. This is because the processor does not have to save and restore the entire stack for each task switch, but only the necessary parts of the stack for the current task.
- 4- Improved debugging: Debugging is easier when each task has its own stack, as it is easier to isolate and identify errors that occur within a specific task.

Overall, while super loops and interrupts can be useful for simple systems, Squad RTOS provides a more reliable and efficient platform for larger and more complex systems that require precise timing and responsiveness.

Super loops Tasks with priorities if task have Priority higher than current thread it will work.[1]



## 2.7 Kernel

### 2.7.1 Dispatcher

A dispatcher in a real-time operating system (Squad RTOS) is responsible for selecting which task or thread should execute next, based on scheduling policies and algorithms. The dispatcher is a key component of the Squad RTOS kernel and plays a critical role in ensuring that real-time tasks are executed in a timely and predictable manner.

The dispatcher works in conjunction with the scheduler, which is responsible for managing the allocation of system resources, such as CPU time, memory, to different tasks or threads. The scheduler uses scheduling policies and algorithms to determine which task or thread should execute next, based on factors such as task priority, deadline, and available resources. The dispatcher then selects the next task or thread to execute, based on the scheduling decision made by the scheduler.

The dispatcher typically operates at a lower level than the scheduler and is responsible for performing context switches between tasks or threads. When the dispatcher selects a new task or thread to execute, it saves the context of the current task or thread and restores the context of the selected task or thread. Context switching involves saving and restoring the contents of CPU registers, program counter, stack pointer, and other state information, and can have a significant impact on system performance.

In a typical Squad RTOS, the dispatcher uses a priority-based scheduling algorithm, where tasks or threads with higher priority are executed first. The dispatcher may also use other scheduling policies, such as round-robin scheduling or earliest deadline first (EDF) scheduling, depending on the specific requirements of the system or application.

The dispatcher must be designed to respond to scheduling decisions in a timely and predictable manner, to ensure that real-time tasks are executed in a timely and reliable manner. The dispatcher must also be designed to handle interrupts and other asynchronous events, which can affect scheduling decisions and require immediate attention.

The dispatcher is a crucial part of real-time operating systems, and it is imperative to carefully design and implement it to guarantee the timely and predictable execution of real-time tasks. The dispatcher works in tandem

with the scheduler and other components of the RTOS kernel to ensure that real-time systems fulfill their performance and reliability objectives.

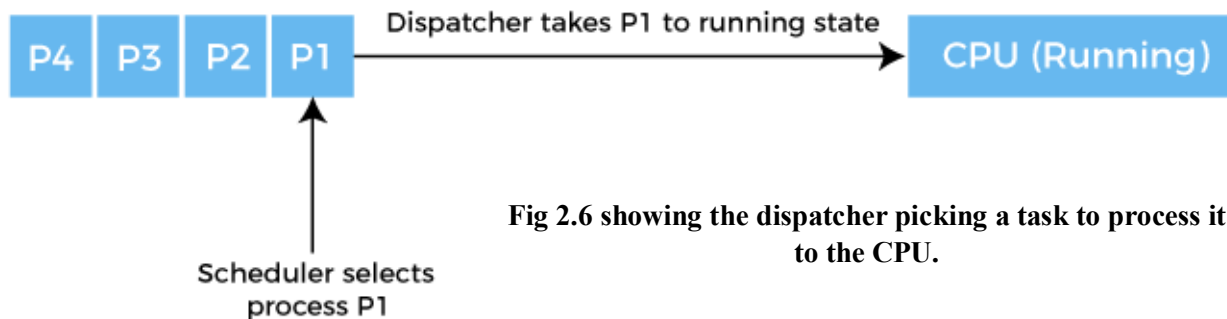


Fig 2.6 showing the dispatcher picking a task to process it to the CPU.

### 2.7.2 Scheduler:

The scheduler is a component that manages the allocation of system resources, such as CPU time, memory, and I/O devices, to different tasks or threads with real-time requirements.

#### Scheduler responsibility:

The scheduler is responsible for ensuring that tasks or threads are executed in a timely and predictable manner, according to their criticality and deadline requirements.

#### Squad RTOS scheduler overview:

Squad RTOS schedulers typically use scheduling policies and algorithms that prioritize tasks based on their criticality and deadline requirements. For example, tasks with hard real-time requirements, such as control tasks in a robotic system, may be given higher priority than tasks with soft real-time requirements, such as user interface tasks.

mannerism schedulers may also include features such as preemption, where a higher-priority task can interrupt a lower-priority task, and interrupt handling, where the scheduler can respond to hardware or software interrupts in a timely manner.

Real-time schedulers are critical components of real-time systems, where tasks or threads must be executed in a predictable and reliable manner, often with strict timing requirements. By managing the allocation of system resources and prioritizing tasks based on their criticality and deadline requirements, real-time schedulers can ensure that real-time systems meet their performance and reliability goals.

**Squad RTOS Scheduler overview:**

Squad RTOS schedulers typically use scheduling policies and algorithms that prioritize tasks based on their criticality and deadline requirements. For example, tasks with hard real-time requirements, such as control tasks in a robotic system, may be given higher priority than tasks with soft real-time requirements, such as user interface tasks.

Real-time schedulers may also include features such as preemption, where a higher-priority task can interrupt a lower-priority task, and interrupt handling, where the scheduler can respond to hardware or software interrupts in a timely manner.

Real-time schedulers are critical components of real-time systems, where tasks or threads must be executed in a predictable and reliable manner, often with strict timing requirements. By managing the allocation of system resources and prioritizing tasks based on their criticality and deadline requirements, real-time schedulers can ensure that real-time systems meet their performance and reliability goals.

**Preemptive-based:**

Preemptive-based scheduling with round robin is a type of CPU scheduling algorithm that combines the features of both preemptive and round-robin scheduling.

Preemptive scheduling means that the operating system can interrupt a running process and switch to another process at any time, based on priority or other scheduling criteria. This allows higher-priority processes to be executed before lower-priority processes, which can improve system responsiveness and reduce wait times.

Round-robin scheduling is a time-sharing algorithm that assigns a fixed time slice to each process in a circular queue. When the time slice expires, the current process is interrupted and moved to the back of the queue, and the next process is executed. This ensures that each process gets a fair share of the CPU time and prevents any single process from monopolizing the CPU.

In preemptive-based scheduling with round robin, the time slice assigned to each process is smaller than in pure round-robin scheduling, typically in

the range of a few milliseconds. This allows the operating system to switch between processes more frequently, based on their priority or other scheduling criteria. If a higher-priority process becomes available, the operating system can preempt the current process and switch to the higher-priority process immediately, ensuring that critical tasks are completed as soon as possible.

Preemptive-based scheduling with round-robin is a versatile and effective algorithm that strikes a balance between the requirements of multiple processes and the timely execution of critical tasks.

### **Advantages of both preemption round-robin scheduling:**

Both preemption and round-robin scheduling can provide fair access to system resources among all tasks, preventing starvation.

They can both be efficient and effective in meeting the performance and timing requirements of different types of tasks.

They can both be combined with other scheduling techniques to create hybrid scheduling algorithms that take advantage of the strengths of multiple techniques.

### **Disadvantages of both preemption round-robin scheduling:**

Both preemption and round-robin scheduling can increase system overhead and complexity by adding scheduling and context-switching overhead.

They can both result in priority inversion and race conditions, which can lead to missed deadlines or system failure.

They may not be suitable for all types of systems or applications and may require careful consideration of the specific requirements of the system.

In summary, the selection of a scheduling technique, whether preemption or round-robin, should be based on the needs of the system or application. By thoughtfully weighing the pros and cons of each technique, and exploring the possibility of hybrid scheduling algorithms, the system can be optimized to achieve its performance and reliability objectives while minimizing unnecessary overhead and complexity.[1]



## 2.8 Interrupt handling:

Interrupt handling is a critical component of Squad RTOS, a real-time operating system, that is responsible for managing the handling of hardware and software interrupts. Interrupts are signals from hardware devices or software events that temporarily halt the normal execution of a task, allowing the system to respond to the interrupt. Interrupt handling involves several components, including the interrupt service routine (ISR), interrupt controller, and interrupt dispatcher.

The interrupt service routine (ISR) is a small piece of code that is executed when an interrupt occurs. The ISR is responsible for handling the interrupt, performing any necessary operations, and returning control to the interrupted task. The ISR is typically designed to be as small and efficient as possible, with minimal overhead, to ensure that it can respond quickly to interrupts.[1]

The interrupt controller is a hardware component that is responsible for managing the flow of interrupts in the system. The interrupt controller receives interrupts from hardware devices and software events and determines which ISR should be executed in response to the interrupt. The interrupt controller can also prioritize interrupts based on their importance and can mask or disable interrupts to prevent interference with critical operations.

The interrupt dispatcher is a component that manages the scheduling of interrupt service routines. The dispatcher determines the priority of each interrupt and schedules the corresponding ISR to run based on the interrupt priority. The dispatcher is responsible for ensuring that the system responds to interrupts in a timely and efficient manner, without introducing delays or conflicts.

Interrupt handling can introduce overhead and complexity into the system, as interrupts must be handled quickly and efficiently to ensure that the system remains responsive and reliable. Interrupt handling can also introduce synchronization issues, as multiple tasks may attempt to access shared resources concurrently. To mitigate these issues, Squad RTOS provides a range of features and mechanisms to manage interrupt handling, including interrupt priorities, interrupt masking, and interrupt nesting.

Interrupt priorities allow the system to prioritize interrupts based on their importance, ensuring that critical events are handled quickly and efficiently. Interrupt priorities are typically assigned based on the type of interrupt and the importance of the event that triggered the interrupt. For example, a

system may assign a higher priority to interrupts from critical hardware devices, such as a watchdog timer or a real-time clock, than to interrupts from less critical devices, such as a keyboard or a mouse.

Interrupt masking allows the system to temporarily disable interrupts to prevent interference with critical operations. Interrupt masking is typically used when a critical operation, such as a system update or a data transfer, is in progress, and interrupts could cause the operation to fail or introduce errors. Interrupt masking can also be used to ensure that interrupts are handled in a specific order, to prevent conflicts or race conditions.

Interrupt nesting is a feature that allows the system to handle multiple interrupts simultaneously or in rapid succession. Interrupt nesting allows the system to prioritize and schedule interrupts based on their importance and to avoid conflicts or delays. Interrupt nesting can be challenging to implement and requires careful design and testing to ensure that it is reliable and efficient.

Interrupt handling is also important for ensuring the safety and security of the system. Interrupt handling can prevent errors and failures caused by hardware or software events and can detect and respond to security threats or attacks. Interrupt handling can also provide diagnostic information and error messages that can be used to identify and resolve system issues.

When an interrupt occurs in an ARM-M processor, the processor saves the current execution context onto the stack, including the program counter (PC), current processor status register (PSR), link register (LR), and stack pointer (SP). In addition to these registers, the operating system may choose to save all the general-purpose registers (R0-R12) onto the stack as well.

Saving all the general-purpose registers allows the operating system to preserve the state of the interrupted program and ensure that no data is lost during the interrupt. This is particularly important in multi-threaded or multi-tasking environments where multiple programs may be executing simultaneously, and interrupts may occur frequently.

By saving all the general-purpose registers onto the stack, the operating system ensures that the interrupted program can be resumed exactly as it was before the interrupt occurred, without the need for the program to explicitly save and restore its own state. This can save time and reduce complexity in programming, as well as improve overall system performance and stability.

[1]

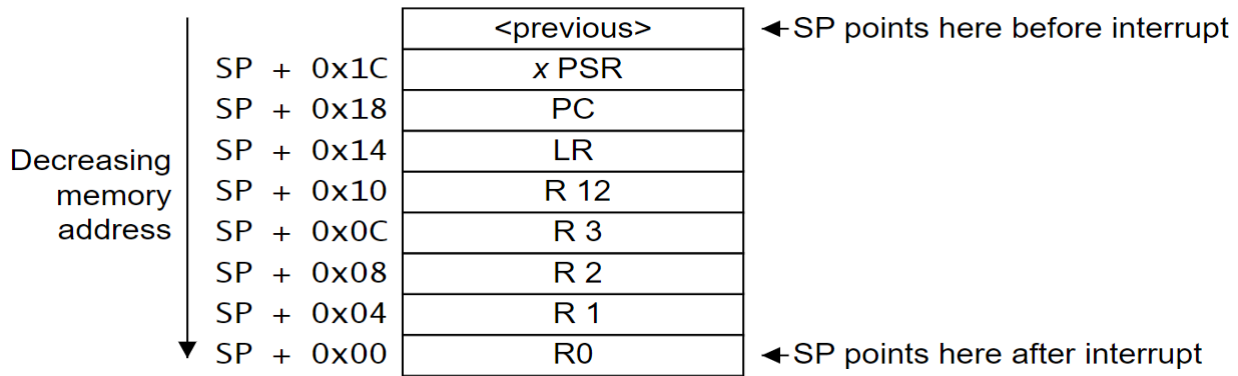


Fig 2.7 After normal interrupt

EXC_RETURN
CONTROL
R4
R5
R6
R7
R8
R9
R10
R11
R0
R1
R2
R3
R12
LR
PC
xPSR

Fig 2.8 After normal interrupt and using Squad RTOS

In summary, interrupt handling is a complex and critical aspect of operating system design, requiring careful consideration of system requirements, hardware capabilities, and software implementation. By effectively managing interrupts, Squad RTOS can provide fast, reliable, and efficient responses to external events and requests, enabling a wide range of applications and use cases. Interrupt handling is also essential for ensuring the safety and security of the system and can provide valuable diagnostic information and error messages.

## 2.9 Memory Management:

Memory management is an essential component of any operating system that is responsible for managing the allocation, utilization, and release of system memory. The operating system must manage the limited memory

resources available in the system and allocate these resources efficiently to support the execution of multiple programs simultaneously.

Memory management involves various tasks, such as memory allocation, memory protection, memory swapping, and garbage collection. Effective memory management ensures that the operating system can run multiple programs simultaneously without running out of memory or causing memory-related errors.

Memory allocation is the process of assigning memory to programs or processes. The operating system must provide a mechanism for allocating memory dynamically as processes are created, and freeing up memory when processes are terminated. Memory allocation can be done using various algorithms, such as first-fit, best-fit, and worst-fit, depending on the size of the available memory and the requirements of the processes.

Garbage collection is the process of reclaiming memory that is no longer being used by a program or process. This is accomplished by identifying memory that is no longer referenced by the program and freeing it up for use by other processes.

Effective memory management can improve overall system performance by reducing the amount of time spent managing memory resources and minimizing the risk of memory-related errors, such as memory leaks and segmentation faults. Memory leaks occur when a program fails to release memory when it is no longer needed, leading to a gradual loss of available memory and eventually causing the system to crash. Segmentation faults occur when a program attempts to access memory that it is not authorized to access, leading to system instability and potential data loss.

Memory management is particularly important in real-time systems, where response time and predictability are critical. In real-time systems, memory allocation and deallocation must be performed in a predictable and deterministic manner to avoid delays and ensure that critical processes have sufficient memory resources available.

In summary, memory management is a critical component of any operating system that is responsible for managing the allocation, utilization, and release of system memory. Effective memory management can improve overall system performance and stability, minimize the risk of memory-related errors, and ensure that critical processes have sufficient memory

resources available. By providing efficient and reliable memory management mechanisms, operating systems can support the execution of multiple programs simultaneously, enabling a wide range of applications and use cases.[1]

## 2.10 How we made Squad RTOS.

### Thread status:

- The thread is running state can ask for resource (RTOS call).
- RTOS makes the thread go into a waiting state when resources are not available.
- waiting state is associated with the event for which the thread is waiting.
- IRQs can trigger some events and RTOS can move a thread from waiting to ready state.
- Thread is switched from running state to ready state when an IRQ triggers high priority thread into ready state.

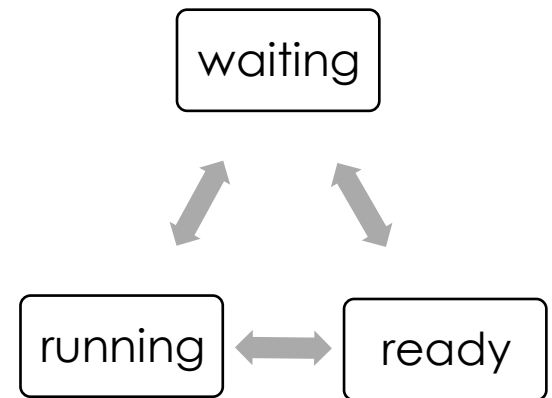


FIG 2.9 showing the status of the thread.

### ARM CM4 OS SUPPORT FEATURES:

- Shadowed stack pointer: MSP is used for the OS kernel and interrupt handler; PSP is used for application tasks.
- SysTick timer.
- SVC and PendSV exceptions.
- Unprivileged execution level: security model that restricts the access rights of some applications.
- Exclusive access: useful for semaphore and mutual exclusive.[11]

### USING PENDSV TO PARTITION AN INTERRUPT:

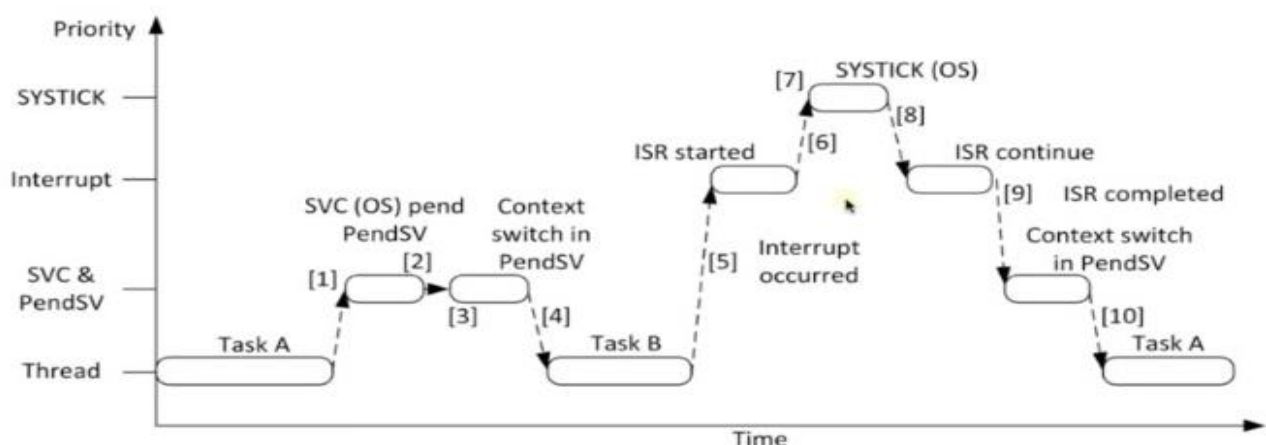
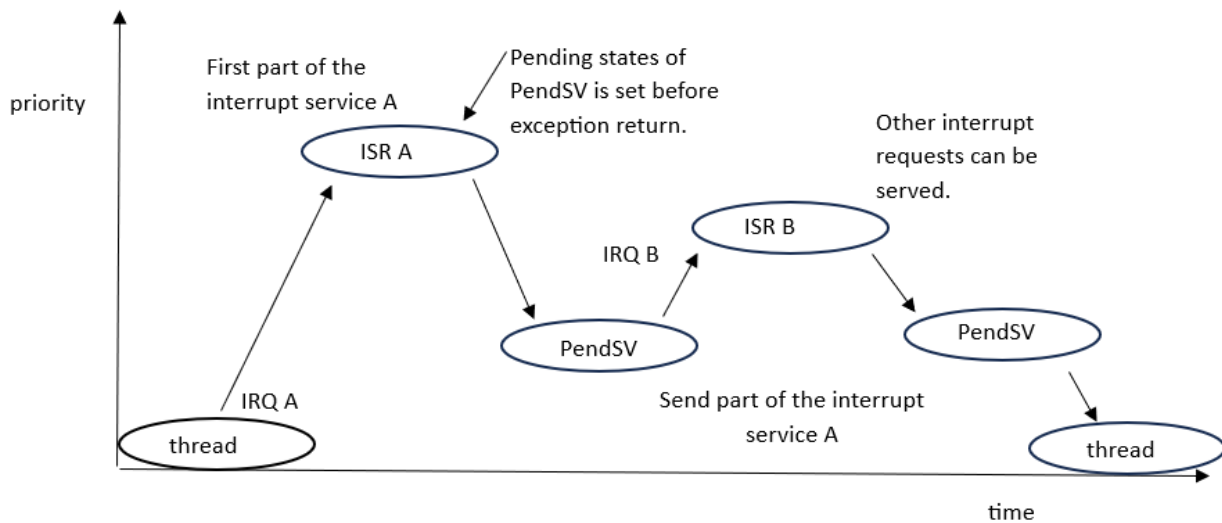


Fig 2.10 Showing PENDSV steps.

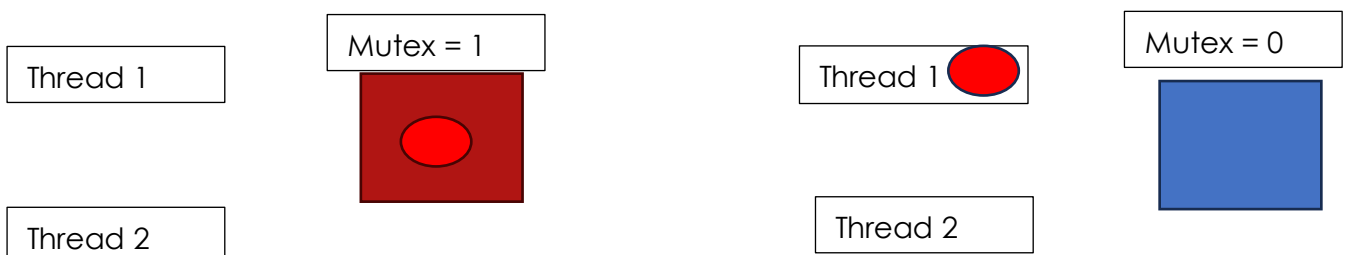
## USING PendSV Dor context switch:



**Fig 2.11 Showing PENDSV Dor context switch steps.**

## Mutex:

- Flag that can be tested and updated automatically.
- Mutex has two states, 1 (no thread in the critical section) and 0 (one thread in the critical section)
- Each thread checks the mutex before entering the critical section, if the mutex is 1, the thread sets the mutex to 0 (mutex lock) and enters the critical section and back to 1 (mutex release) after leaving.[10]



**Fig 2.12 Showing how Mutex work.**

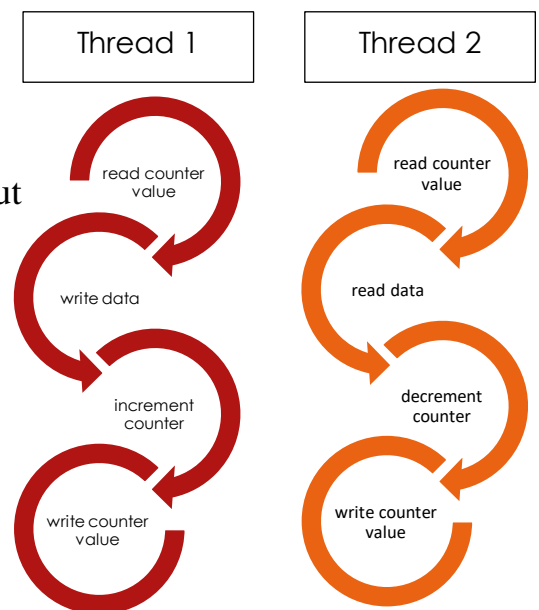
## MAILBOX:

- Used for interthread communication, unlike semaphores, which provide only synchronization with information exchange.
- Using shared buffer one thread can fill the buffer (producer), the other empties it (consumer)

- The consumer needs to wait when the buffer is empty, and the producer needs to wait when the buffer is full.
- Shared buffer needs to be updated by the producer and consumer mutually exclusively, so mutex is needed for mutual exclusive access to the buffer and the semaphore for counting number of messages in the buffer.
- The buffer has limited size, producer can't fill the buffer infinitely, so another semaphore is needed to count the empty slots in buffer.

### MUTUAL EXCLUSION:

- Two or more threads may want to use non-sharable hardware (e.g., serial port), or accessing shared memory simultaneously.
- Atomic operation can solve the problem, but no processor can update the memory location with one instruction.
- One way to use critical sections (threads cannot preempt) when accessing shared resources.
- The Critical section is the piece of code which accesses shared resources.
- Critical sections can be implemented by disabling interrupts, which could increase.[10]



**Fig 2.13 Showing how Threads working together.**

# Chapter 3: Communication stack



### **3.1 What is an IP address:**

The term "IP" refers to a set of guidelines that control how data is carried via computer networks, including the internet. IP oversees delivering data packets accurately and directing them from one device to another.

Each device connected to a computer network that makes use of the Internet Protocol is given a numerical label known as an IP address. The IP address performs two primary tasks: it identifies the host or network interface and gives the device's location within the network.

For example, IP addresses are used to enable communication between devices on a local network, route traffic across the internet, and locate devices for geolocation services. They are a crucial component of the internet's architecture and are utilised by several protocols, including HTTP, DNS, and TCP/IP.

IP addresses are essential for facilitating communication between devices on a network in addition to identifying devices and supplying location information. The IP address of the target device is included in the packet header when a device delivers data across the network. This address is used by routers and other networking equipment to direct packets to their intended destinations.

The Internet Service Provider (ISP) of a device or the network administrator are normally responsible for allocating IP addresses. Static and dynamic allocations of IP addresses are the two primary categories. An IP address that is manually allocated to a device and is known as a static IP address does not change over time. On the other hand, a dynamic IP address is given by a DHCP server and is subject to change.

IP addresses are a crucial component of the internet's architecture and are utilised in a wide variety of protocols and applications. They provide for the routing of data via the internet, network device communication, and the identification and placement of devices for a variety of uses.

One of the most crucial protocols used in computer networking, IP (Internet Protocol), is a foundational protocol of the internet. It oversees giving network devices specific addresses, directing data packets between networks, and guaranteeing dependable data delivery.

**Here are some justifications for why IP is crucial:**

1- Addressing: IP assigns an individual logical address, or IP address, to every device connected to a network. This makes it possible for devices to connect with each other across networks, making the internet a global network of networks.

2- Routing: IP is responsible for routing data packets between networks, using routing protocols such as OSPF (Open Shortest Path First) and BGP (Border Gateway Protocol). This allows data to be transmitted efficiently and reliably across the internet.

3- Reliability: IP offers techniques to guarantee error-free and dependable data transmission. This comprises congestion control, error detection and repair, and packet fragmentation and reassembly.

4- Compatibility: Almost all computer networking hardware and operating systems use IP, a standardized protocol. This makes it simple for various device kinds to talk with one another and guarantees that data may be delivered over various network architectures.

5- Routing: Using routing protocols like OSPF (Open Shortest Path First) and BGP (Border Gateway Protocol), IP is responsible for directing data packets between networks. This makes it possible for data to be consistently and effectively sent via the internet.

6- Scalability: IP is made to be scalable, which enables it to accommodate numerous devices on a network and manage enormous volumes of data traffic. As a result, billions of devices and people may access the internet worldwide.

IP is an essential part of the internet and computer networking because it is a crucial protocol for enabling communication and data transport across networks. It is a crucial protocol for contemporary communication systems due to its distinct addressing, routing, dependability, compatibility, and scalability capabilities.[11]

### **3.2 IP versions:**

There are two types of IP addresses: IPv4 and IPv6. IPv6 and IPv4. IPv6 is a more recent version that was created to solve IPv4's restrictions and give a greater number of unique IP addresses. IPv4 is the older and more extensively used version. The four decimal integers in an IPv4 address are

32-bit numbers separated by dots. The IPv4 address 192.168.0.1 is a good illustration. But as more devices become connected, IPv4 addresses are getting harder to come by, which prompted the creation of IPv6. IPv6 addresses allow for a substantially higher number of unique addresses since they are 128-bit values written in hexadecimal format.

### 3.2.1 IPv4

The fourth version of the Internet Protocol, known as IPv4, is referred to as Internet Protocol version 4. The most used IP version, IPv4, is the basis for most modern internet connectivity.

Since IPv4 addresses are 32-bit numbers expressed in dotted-decimal format, they are made up of four sets of decimal numbers, each ranging from 0 to 255, separated by periods. An IPv4 address would appear something like this: 192.168.0.1, for instance.

4.3 billion unique IPv4 addresses are available in total, which was first believed to be enough for all internet-connected devices. However, IPv4 addresses are becoming scarce due to the quick increase in the number of devices connected to the internet.

Because of its simplicity, effectiveness, and compatibility, IPv4 has been extensively embraced. It has been applied in several applications and protocols, including TCP/IP, UDP, and ICMP, and it offers a dependable and effective method of data routing across the internet.

The restricted availability of IPv4 addresses, however, prompted the creation of IPv6, which offers a significantly higher number of unique addresses and more functionalities. While IPv4 will likely still be used soon, IPv6 is already in use and is expected to become more crucial as more devices connect to the internet.

#### **Here are some other information regarding IPv4:**

1- Address format: IPv4 addresses are made up of 32 bits, which are then broken up into four groups of 8 bits each. The decimal numbers 0 to 255 that make up each octet are divided into groups of eight by periods. The IP address 192.168.0.1, for instance, comprises four octets: 192, 168, 0, and 1.

2- Private addresses: IPv4 contains several reserved address ranges used for private networks. These addresses are only used for communication within a

local network and are not globally routable. The three private address ranges that are most frequently used are 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16.

3- Subnetting: By dividing IPv4 addresses into smaller subnets, network managers may build more streamlined networks with higher levels of performance and security. By taking bits from the host section of the IP address and utilizing them to build a subnet mask, subnetting is accomplished.

4- Network Address Translation (NAT): This method enables devices connected to a private network to share a single public IP address. Home routers and firewalls frequently employ NAT to enable several devices to access the internet using a single connection.

5- Address exhaustion: Due to the IPv4 address shortage, IPv6 was created, which uses 128-bit addresses and offers a substantially higher number of unique addresses. While IPv6 adoption is happening gradually, IPv4 will probably still be in use for some time.

In general, IPv4 is a widely used and dependable protocol for data routing over local networks and the internet. The next version of IP, IPv6, was created and adopted because of the restricted number of addresses that are now a serious problem.[11]

### 3.2.2 IPv6

The most recent version of the Internet Protocol is known as IPv6, or Internet Protocol version 6. IPv6 was created to remedy IPv4's shortcomings, notably the scarcity of IP addresses, and it offers a significantly bigger address space.

Hexadecimal-coded 128-bit values are used to represent IPv6 addresses. Eight sets of four hexadecimal digits each are separated by a colon. An IPv6 address, for instance, would be like this: 2001:0db8:85a3:0000:0000:8a2e:0370:7334.

The nearly 3.41038 unique addresses offered by IPv6 are much higher than the 4.3 billion addresses provided by IPv4 in total. This makes it possible for a lot more devices to be linked to the internet and gives address assignment more freedom.

**Other new features and enhancements included in IPv6 besides the expanded address space include:**

- 1- Stateless Address Autoconfiguration: Without a DHCP server, IPv6 devices can autonomously give themselves a special IP address.
- 2- Simplified Header: The IPv6 header is more straightforward than the IPv4 header, enabling quicker processing and better network performance.
- Extension Headers: IPv6 includes extension headers that allow for additional functionality and flexibility.
- 3- Security: IPv6 includes built-in support for IPsec, which provides secure communication over the internet and local networks.
- 4- Multicast: IPv6 includes support for multicast, which allows a single packet to be sent to multiple devices simultaneously.

As more devices connect to the internet, IPv6 is being adopted gradually and is anticipated to gain significance. Although many devices and networks still use IPv4, IPv4 will be used for the foreseeable future as the switch to IPv6 will take some time.

**Here are some additional details about IPv6:**

- 1- Address format: The 128 bits in IPv6 addresses are split up into eight groups of 16 bits each. Four hexadecimal numbers are used to represent each group, which are separated by colons. The IP address 2001:0db8:85a3:0000:0000:8a2e:0370:7334, for instance, contains eight groups: 2001, 0db8, 85a3, 0000, 0000, 8a2e, and 0370.
- 2- Address types: There are other kinds of addresses in IPv6, including unicast, multicast, and anycast addresses. One-to-one communication is conducted using unicast addresses, one-to-many communication is conducted using multicast addresses, and one-to-nearest communication is conducted using anycast addresses.
- 3- Routing: IPv6 employs a hierarchical routing scheme that enables more effective and scalable data routing over local and wide-area networks. An IPv6 address's prefix is used by IPv6 routers to identify the network, and its suffix is used to identify the host.
- 4- Address distribution: IPv6 addresses are distributed to regional Internet registries (RIRs) by the Internet Assigned Numbers Authority (IANA),

which subsequently distributes addresses to Internet service providers (ISPs) and businesses. A hierarchical approach to IPv6 address distribution enables effective and scalable address management.

5- Transitional methods: The switch from IPv4 to IPv6 will take place gradually over time. Several transition mechanisms that enable interoperability between IPv6 and IPv4 networks are included in IPv6 to ease the transition. These methods include dual-stack operation, translation, and tunnelling.

In general, IPv6 outperforms IPv4 and offers a significantly bigger address space, better speed, and extra functionality. While the switch to IPv6 will take some time, it is anticipated that its significance will increase as more devices are connected to the internet and as the need for distinctive IP addresses rises.[11]

**In comparison to IPv4, IPv6 performs better in several ways:**

1- Simplified Header: The IPv6 header is more straightforward than the IPv4 header, enabling quicker processing and better network performance. Unlike the IPv4 header, which can range in size from 20 to 60 bytes, the IPv6 header has a set length of 40 bytes.

2- A wider address space: Compared to IPv4, IPv6 has a substantially larger address space, which makes it possible to address and route devices on local and public networks more effectively. Because there is less need for Network Address Translation (NAT), which can enhance network performance, the address space is larger.

3- Stateless Address Autoconfiguration: Without a DHCP server, IPv6 devices can autonomously give themselves a special IP address. This makes network device configuration quicker and more effective.

4- Extension headers: IPv6 comes with extension headers, which offer more capability and flexibility. Extension headers can be used to include extra information in the packet, such as security, fragmentation, or routing data. This can enhance network performance and lower processing overhead associated with packet processing.

5- Multicast: IPv6 has enhanced multicast functionality, enabling the simultaneous transmission of a single message to several devices. Applications like video streaming, which can boost network efficiency by

lowering the amount of traffic required to convey the same data to several devices, can take advantage of multicast.

Overall, IPv6 outperforms IPv4 in terms of performance thanks to a more straightforward header, a bigger address space, better address setting, more functionality and flexibility, and better multicast support. Data routing via the internet and local networks may become quicker and more effective because of these advancements, which may enhance user experience and network performance.

ICMP (Internet Control Message Protocol) and ARP (Address Resolution Protocol), two protocols that are used for network diagnosis and address resolution, respectively, are included in IP.

End-to-end communication between devices on a network is provided by IP, which works at the network layer of the OSI (Open Systems Interconnection) paradigm. It accomplishes this by giving every device on the network a special IP address and using that address to route data packets from the source device to the destination device.

### **3.3 OSI and TCP/IP models**

#### **3.3.1 What is OSI model?**

The OSI (Open Systems Interconnection) model is a theoretical framework that describes how a communication system works and how its parts interact. The model is divided into seven levels, each of which offers a particular set of services and protocols that combine to make it possible for devices on a network to communicate with one another.

The OSI model is organized into seven layers, which are:

- Application layer
- Presentation layer
- Session layer
- Transport layer
- Network layer
- Data link layer
- Physical layer

### 3.3.2 TCP/IP model

The four layers that make up the TCP/IP protocol suite are:

- 1- Application layer: This layer oversees giving user apps network services. It consists of protocols like Telnet, HTTP, FTP, SMTP, and SMTP.
- 2-Transport layer: This layer oversees facilitating dependable end-to-end communication between programs that are hosted on various hosts. It consists of protocols like TCP and UDP.
- 3- Internet layer: This layer oversees offering the fundamental packet-switching and routing capabilities required for data transmission over an internetwork. It consists of ICMP, IP, and IGMP protocols, among others.
- 4- Network access layer: For a networked device and the network medium to communicate with one another, this layer must provide a physical and electrical interface. It consists of DSL, Ethernet, and Wi-Fi protocols.[6]

**Each layer is briefly described below:**

- 1 - Application layer: In the TCP/IP protocol family, the application layer is the uppermost layer. It oversees designing the protocols that apps use to connect with one another across the network and offers network services to user applications.
- 2 - Transport layer: The transport layer oversees facilitating dependable end-to-end communication between programs that are hosted on various hosts. It consists of protocols like TCP and UDP. TCP offers a dependable, connection-oriented service that ensures that data will be delivered in the order that it was provided. Data delivery is not guaranteed by the connectionless, unstable service offered by UDP.
- 3- Internet layer: The internet layer oversees offering the fundamental packet switching and routing capabilities required for data transmission over an internetwork. It consists of ICMP, IP, and IGMP protocols, among others. To move packets between hosts, IP offers a connectionless, best-effort service. The method for reporting faults and status details on network circumstances is provided by ICMP. The technique for controlling multicast group membership is provided by IGMP.



4- Network access layer: To provide a physical and electrical interface between a networked device and the network media, the network access layer oversees doing so. It consists of DSL, Ethernet, and Wi-Fi protocols. These protocols specify the details of the network medium's construction, including the kind of cable used, the transmission speed, and the kind of modulation that is applied to the media to encrypt data.

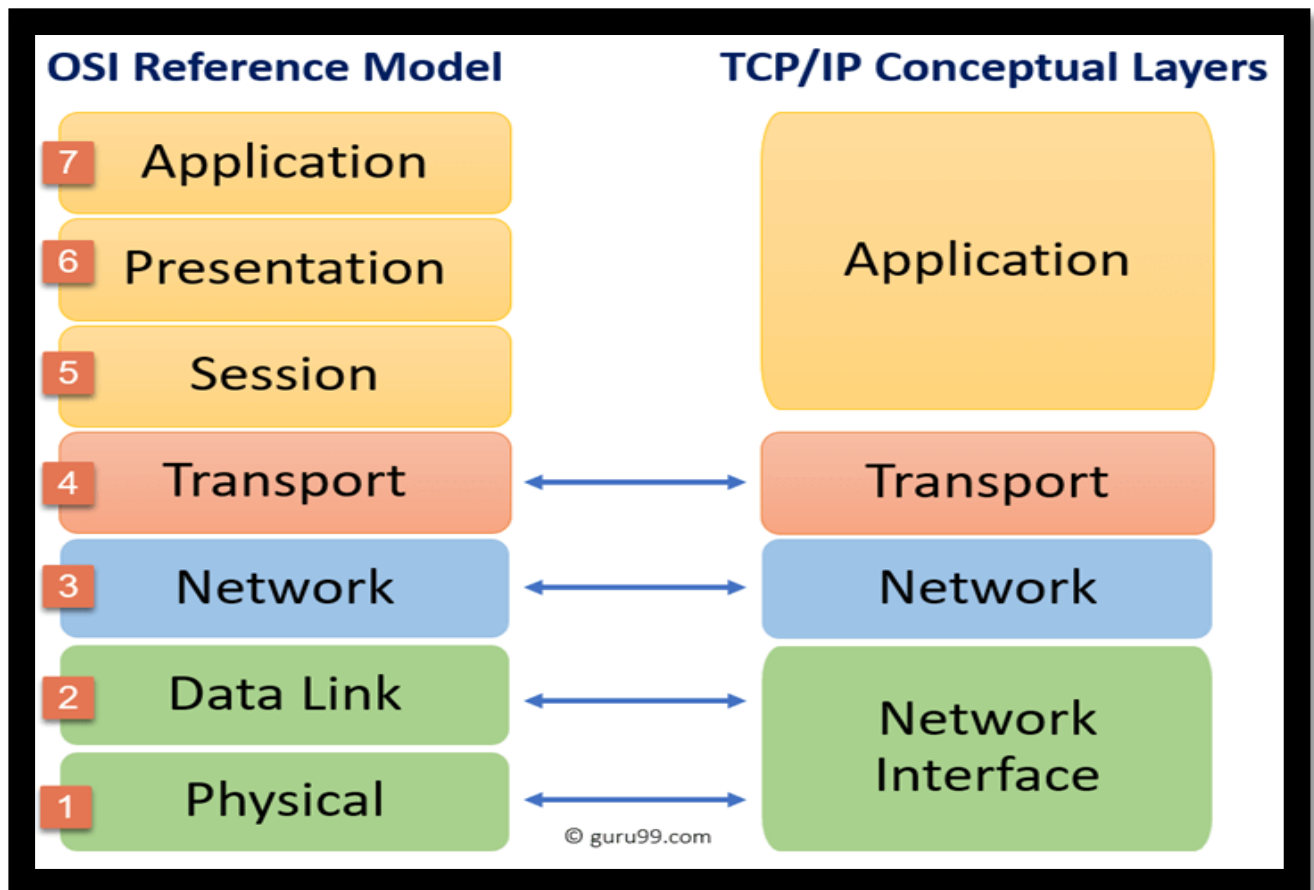


Fig 3.1 shows the OSI layers vs TCP/IP layers.

### 3.3.3 Reasons why OSI model is being used.

**The OSI model is utilised for several purposes, including:**

The OSI model has several benefits over the TCP/IP paradigm, even though it is less often used and implemented in practice. Several benefits are as follows:

1- **Modularity:** With seven different levels that clearly separate functions, the OSI paradigm is more modular than the TCP/IP architecture. This modularity facilitates network protocol design, development, and maintenance as well as potential network troubleshooting.

2- Standardization: The OSI model is a standardized model, which implies that it gives network protocol designers and implementers a common vocabulary and structure. The standardization can aid in ensuring compatibility across various network applications and devices from various suppliers.

3- Flexibility: Having well defined interconnections between layers, the OSI architecture is more versatile than the TCP/IP approach and enables more customization and specialization. In some applications or contexts where there are requirements or limits, this flexibility might be helpful.

4- Educational value: Because of its obvious layering and modular structure, the OSI model is frequently used in educational settings to teach networking principles and protocol creation. Students may better grasp the roles played by each layer and how they interact to produce network communication by using the OSI model.

Overall, the OSI model has some benefits in terms of modularity, standardization, flexibility, and educational value, even though the TCP/IP model is more frequently used and adopted in practice. In practice, network protocols and applications are often developed and implemented using the TCP/IP model, and it should be emphasized that the OSI model is not as extensively used as the TCP/IP model.

Here are some more specifics on the OSI model:

1- Layered architecture: The OSI model is built on a layered architecture, meaning that each layer consumes services offered by layers below it and offers a certain set of services to layers above it. As a result, new protocols and technologies may be implemented more easily and with modular architecture.

2- Interoperability: The OSI model was created to encourage communication between various kinds of computer networks and systems. The paradigm enables communication between devices from many manufacturers and with various underlying technologies by creating a set of common protocols and services at each tier.

3- Protocol stack: A protocol stack refers to all of the protocols that a device uses to connect with other devices on a network. One or more protocols in the stack correspond to each layer in the OSI model. The

transport, network, and data connection layers of the OSI model are represented by the TCP/IP protocol stack, which is utilised by the internet.

4- Encapsulation: At each layer of the OSI model, data must be enclosed in a protocol header to be transmitted over a network. Encapsulation is the process by which the data may be enhanced by each layer as it moves up the protocol stack.

5- Protocol implementation: Depending on the technology being utilised, the way protocols are implemented at each layer of the OSI model might differ. Even though they both function at the same layer of the OSI model, the data link layer protocols used by Ethernet networks and Wi-Fi networks are distinct from one another.

In general, the OSI model offers a common framework for comprehending how computer network's function and how various protocols and technologies combine to make it possible for devices to communicate with one another. The OSI model is still employed in some academic and research environments, even though it is not as frequently utilised in practice as it once was. It is nevertheless a crucial conceptual framework for comprehending networking topics.

**The OSI model's seven layers are:**

1- Physical layer: In charge of sending unprocessed data via a physical channel, like a wire or fiber-optic cable. It specifies the medium's physical properties, including the timing of signals and voltage levels.

2- Data link layer: This layer oversees transferring data among local network devices. To guarantee reliable and error-free data transmission, it employs several protocols.

3- Network layer: This layer oversees directing information among networks. It employs protocols like IP (Internet Protocol) to transport data packets across various networks and identify them.

4- Transport layer: This layer oversees guaranteeing reliable and effective data transmission between devices. It employs UDP (User Datagram Protocol) for quicker, less reliable transmission and protocols like TCP (Transmission Control Protocol) for reliable data delivery.

5- Session layer: This layer controls how devices on a network communicate with one another. It creates, controls, and ends sessions between devices and oversees the data synchronization between them.

6- Presentation layer: In this layer, data formatting and display are handled. It makes sure that data is delivered in a way that the receiving device can understand.

7- Application layer: This layer acts as a bridge between the network and any device-based apps. It outlines the protocols that apps employ while corresponding with one another across a network.

The OSI model offers a framework for comprehending the operations of computer networks and the duties carried out by each layer. It aids in the understanding of how various protocols and technologies interact to provide communication between devices on a network-by-network administrators and developers.[11]

### 3.3.4 OSI layers

#### 1 - Physical layer:

The first layer of the OSI (Open Systems Interconnection) architecture is called the physical layer, and it oversees sending unprocessed data through a physical media like a wire or fiber optic cable. The physical layer defines the physical characteristics of the medium, such as voltage levels, signal timing, and other physical attributes that are required to transmit data over the medium.

#### **The following tasks are handled by the physical layer:**

I) Data encoding and transmission: The physical layer oversees converting digital data that must be communicated into analogue signals that can be conveyed through the physical media. Additionally, it specifies the parameters for signal transmission over the medium, including the voltage levels and timing of the signals.

II) Transmission medium: The physical layer specifies the transmission medium's physical properties, including the kind of cable, the cable's maximum length, and the connectors that are used to join devices.

III) Physical topology: The physical layer also specifies the network's physical topology, or the manner that objects are physically affixed to one

another. Bus, star, ring, and mesh topologies are a few examples of physical topologies.

IV) Network interface: The network and physical devices, such as network interface cards (NICs) and modems, are interfaced through the physical layer. The physical layer establishes the parameters for the network interface, including the type of connection in use and the highest supported data rate.

Ethernet, Wi-Fi, Bluetooth, and RS-232 are a few protocols and technologies that function at the physical layer.

Overall, because it oversees the fundamental transmission of data over a physical media, the physical layer is a crucial part of the OSI model. The physical layer facilitates communication between devices on a network by specifying the physical properties of the medium and the rules for transferring data over it.

## **2 - Data link layer:**

The second layer of the OSI model is called the data link layer, and it oversees transferring data among local network devices. To guarantee reliable and error-free data transmission, the data connection layer employs several protocols. The Media Access Control (MAC) sublayer and the Logical Link Control (LLC) sublayer are the two sublayers that make up the data link layer.

The MAC sublayer oversees managing data packet transmission and restricting access to the physical media. It outlines the protocols for utilised the medium, including how devices share it and how collisions are handled. A protocol that functions at the MAC sublayer is Ethernet.

The LLC sublayer oversees providing flow management and error checking, as well as determining the protocol being utilised for data transfer.

The LLC sublayer may be utilised with a range of various network types and is not tied to any one form of physical medium. HDLC (High-Level Data Link Control) and IEEE 802.2 are two examples of protocols that function at the LLC sublayer.[11]

**The following duties are within the purview of the data connection layer:**

I) Framing: To enable network transmission, the data link layer splits data packets into smaller frames. Each frame is given a header and a trailer that includes details about the data, including the source and destination addresses.

II) Access control: The data connection layer is in charge of regulating the transmission of data packets and restricting access to the physical medium. It outlines the protocols for utilising the medium, including how devices share it and how collisions are handled.

III) Detection and repair of faults: The data connection layer has capabilities for identifying and fixing transmission mistakes. Checksums, cyclic redundancy checks (CRCs), and parity bits are a few examples of error detection and repair algorithms.

Ethernet, Wi-Fi, and Bluetooth are a few examples of protocols that work at the data connection layer. A crucial part of the OSI model is the data link layer, which oversees making sure data is transferred successfully and error-free via the local network.

### **3 - Network layer**

The third layer of the OSI (Open Systems Interconnection) paradigm is the network layer, which oversees directing data between networks. Data packets are identified and routed over numerous networks using protocols like IP (Internet Protocol) at the network layer.

**The following activities are under the purview of the network layer:**

I) Addressing: Each device on the network is given a special logical address by the network layer, which is used to identify the device and direct data packets towards it. Logical addresses used by the network layer include IP addresses.

II) Routing: Data packets must be routed across networks using the network layer. Based on variables including network congestion, speed, and dependability, it employs routing protocols to decide the optimum route for data packets to travel via the network.

III) Fragmentation and reassembly: The network layer has the ability to split up a large data packet into smaller ones that may be sent over the network.

To reconstitute the original data packet, it also reassembles the packets at the target device.

IV) Quality of service (QOS): Data packets may be given higher priority by the network layer depending on their significance or the kind of service they require due to quality of service (QoS). As a result, time-sensitive applications like audio and video run better and network resources may be used more effectively.

The Internet Protocol (IP), ICMP (Internet Control Message Protocol), and routing protocols like OSPF (Open Shortest Path First) and BGP (Border Gateway Protocol) are a few examples of protocols that work at the network layer.

Overall, because it oversees directing data packets between networks and ensuring that they are sent effectively and dependably, the network layer is a crucial part of the OSI model. The network layer makes it possible for devices on various networks to communicate with one another by giving logical addresses to devices and utilising routing algorithms to choose the optimal route for data packets to travel.

#### **4 - Transport layer**

The transport layer, the fourth OSI (Open Systems Interconnection) layer, oversees making sure that data is transported between devices effectively and reliably. The TCP (transfer Control Protocol) and UDP (User Datagram Protocol) protocols are used by the transport layer to offer reliable data transfer and quicker, less reliable transmission, respectively.

**The following activities are within the purview of the transport layer:**

I) Segmentation and reassembly: The transport layer could split up big data packets into smaller ones that can be sent over the network. To reconstitute the original data packet, it also reassembles the segments at the target device.

II) Management of connections: The transport layer controls the creation, upkeep, and disconnection of connections between devices. Creating a communication session, managing flow, and recovering from errors are all included in this.

III) Reliability: The transport layer offers techniques to guarantee error-free and dependable data transmission. An example of a protocol that offers

dependable data transfer is TCP, which employs techniques including flow control, retransmissions, and acknowledgments.

IV) Multiplexing and demultiplexing: The transport layer can manage several concurrent communication sessions between devices. To identify the communication session and make sure that data is transferred to the right program on the target device, it employs port numbers.

TCP (Transmission Control Protocol), UDP (User Datagram Protocol), and SCTP (Stream Control Transmission Protocol) are a few examples of protocols that work at the transport layer.

Overall, the transport layer is a crucial part of the OSI model since it oversees making sure that data is transported between devices effectively and reliably. The transport layer supports communication between devices on a network by offering techniques for segmentation and reassembly, connection management, dependability, multiplexing, and demultiplexing.

## 5 - Session layer

The session layer is the fifth layer of the OSI (Open Systems Interconnection) model, and it manages the communication sessions between devices on a network. The session layer establishes, manages, and terminates sessions between devices and manages the synchronization of data between devices.

**The session layer is responsible for the following functions:**

I) Establishing a communication session between devices on a network is the responsibility of the session layer. This involves deciding whether a simplex, half-duplex, or full-duplex session should be formed.

II) Session management: The session layer controls flow, recovers from errors, and synchronizes data between devices during a communication session between devices. To make sure that data is transferred in the proper order, it oversees controlling the sequence of messages that are exchanged between devices.

III) Session termination: A communication session between devices on a network must be terminated by the session layer. As part of this procedure, any resources allotted for the session must be released, and any data communicated during the session must have been received and processed.



IV) Security: The session layer oversees making sure that device-to-device communication is safe. This involves securing data transmission against mistakes or manipulation and encrypting data to prevent unauthorized access.

SSH (Secure Shell), which offers secure communication sessions between devices, and NetBIOS (Network Basic Input/Output System), which controls communication sessions between devices on a LAN, are two examples of protocols that function at the session layer.

Overall, because it controls how devices on a network communicate with one another, the session layer is a crucial part of the OSI architecture. The session layer supports communication between devices on a network by initiating, maintaining, and ending sessions between devices as well as making sure that data is sent safely and without mistakes.

## **6 - Presentation layer**

The OSI (Open Systems Interconnection) model's sixth layer, the presentation layer, oversees formatting and presenting data. The presentation layer makes ensuring that data is presented in a way that the receiving device can understand. To guarantee that data is transferred safely, the presentation layer also handles data encryption and decryption.

### **The following tasks fall under the purview of the presentation layer:**

- I) Data conversion: The presentation layer could convert data between different file formats. For instance, it may convert picture files from one format to another or text from one character encoding to another.
- II) Data compression: To lessen the quantity of data that must be communicated over the network, the presentation layer might compress data. This can increase network efficiency and decrease the quantity of data storage needed.
- III) Data encryption and decryption: To make sure that data is delivered safely across the network, the presentation layer might encrypt it. Data that has been encrypted by a different device can also be decrypted by it.
- IV) Data formatting: Data can be formatted by the presentation layer before being presented to the user. It can format data as text, graphics, or multimedia material, for instance.

Examples of protocols that work at the presentation layer include MIME (Multipurpose Internet Mail Extensions), which enables email messages to contain multimedia content like images and videos, and SSL/TLS (Secure Sockets Layer/Transport Layer Security), which offers data encryption and decryption for secure transmission over the internet.

## **7- Application layer**

The OSI (Open Systems Interconnection) model's seventh and topmost tier, the application layer, serves as the interface between the network and programs operating on devices. The protocols that programs employ to interact with one another across the network are specified at the application layer.

**The following duties fall under the purview of the application layer:**

I) Application protocols: A variety of protocols are included in the application layer that specify how apps communicate with one another via a network. HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), and DNS (Domain Name System) are a few examples of application layer protocols.

II) Data exchange: The application layer oversees transferring data between programs that are executed on various devices. This covers data requests, answers, and any other information that must be shared across programs.

III) User interface: The user interface for programs operating on devices is provided by the application layer. This covers command-line interfaces (CLIs), graphical user interfaces (GUIs), and other interfaces that let users communicate with software.

IV) Network services: Applications running on devices may receive network services from the application layer. Email, online surfing, and file transfer are a few examples of network services the application layer offers.

The HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), and FTP (File Transfer Protocol) are some examples of protocols that work at the application layer. HTTP is used for web surfing, SMTP is used for email, and FTP is used for file transfers.

Overall, because it serves as the interface between the network and programs operating on devices, the application layer is a crucial part of the OSI model. The application layer enables communication between devices

on a network and permits users to access the resources and services they require by defining the protocols used by applications to connect with one another across the network and offering network services to applications.[14]

### 3.4 LwIP

An open-source TCP/IP stack called lwIP (lightweight IP) is made for embedded systems and real-time applications. It is intended to be compact, effective, and very portable, making it ideal for usage in situations with limited resources like microcontrollers and embedded systems.

lwIP was developed by Adam Dunkels. In the late 1990s, he created the stack while completing his PhD thesis at the Swedish Institute of Computer Science (SICS). Dunkels was inspired by the need for a compact and effective TCP/IP stack that could function on resource-constrained embedded computers.

In 2001, Dunkels published the initial iteration of lwIP, and since then, the embedded systems industry has embraced it with open arms. Over the years, Dunkels has continued to develop and maintain lwIP, which is now maintained by a developer community on GitHub.

Dunkels has created other well-known software in addition to lwIP, such as the Contiki operating system for low-power wireless networks and the uIP TCP/IP stack for compact embedded devices.

LwIP is primarily used to give real-time and embedded systems networking capabilities. This involves delivering dependable and effective network communication as well as allowing devices to connect with one another across Ethernet networks or other communication interfaces.

#### **The key components of lwIP are:**

- I) IP forwarding over multiple network interfaces is supported by lwIP, enabling devices to transport IP packets between several networks. Applications like IoT (Internet of Things) applications that call for communication between devices on various networks might benefit from this functionality.
- II) ICMP for network upkeep and troubleshooting: lwIP contains support for ICMP, a protocol used for network upkeep and troubleshooting. ICMP messages can be used to evaluate network performance, identify network issues, and determine if hosts are reachable.

III) IGMP for multicast traffic management: IGMP, a protocol for multicast traffic management, is supported by lwIP. Devices can join and exit multicast groups at any time thanks to IGMP messages, which control multicast group membership.

IV) Experimental UDP-lite extensions with UDP: lwIP offers support for the UDP protocol, which is used to transmit and receive datagrams. UDP-lite, a subset of UDP that supports partial checksums and selective retransmission, is also experimentally supported by lwIP.

V) Congestion management, RTT estimate, and rapid recovery/quick retransmit: TCP, a protocol for dependable, connection-oriented communication, is supported by lwIP. Congestion management, Round Trip Time (RTT) estimate, and rapid recovery/quick retransmit are three features of lwIP's TCP implementation that serve to ensure dependable and effective network communication.

VI) Raw/native API for improved performance: The IP layer may be accessed directly using the raw/native API included in lwIP. Applications needing high speed or in-depth control over the network stack should use this API.

VII) Optional Berkeley-like socket API: To be compatible with current networking applications, lwIP additionally contains an optional Berkeley-like socket API. This API makes it simpler to create lwIP-using apps by providing a higher-level interface for network communication.

VIII) Support for the DNS protocol, which is used for domain name resolving, is provided by lwIP. Devices may connect to remote hosts on the Internet more easily by using DNS to transform domain names into IP addresses.

IX) SNMP for network administration: SNMP, a protocol used for network management, is supported by lwIP. Remote device monitoring and management is made possible through SNMP, which makes it simpler to identify and resolve network issues.

X) The DHCP protocol for dynamic host setup is supported by lwIP. This protocol is used for configuring dynamic hosts. DHCP makes it simpler for devices to join to networks by enabling them to automatically get IP addresses, network configurations, and other configuration data.

XI) AUTOIP for IPv4: lwIP also supports the Automatic Private IP Addressing (Automatic Private IP Addressing) protocol, which is used to allocate IP addresses automatically when a DHCP server is not present. On a local network, AUTOIP enables communication between devices without the need for manual configuration.

XII) PPP for point-to-point communication: PPP, a protocol for point-to-point communication over serial lines, is supported by lwIP. It is simpler to link devices together thanks to PPP, which offers a standardized method of delivering IP packets over a range of physical media.

XIII) ARP for Ethernet: The address resolution protocol (ARP), which converts IP addresses to Ethernet addresses, is supported by lwIP. ARP enables Ethernet frames to be used for local network communication between devices.

LwIP is an all-around capable and adaptable TCP/IP stack that may be utilised in a variety of embedded devices and real-time applications. For developers that need dependable and effective network connectivity in their applications, its tiny memory footprint, fast performance, and extensive protocol support make it a desirable option.

**A few of lwIP's important characteristics are as follows:**

I) Compact memory footprint: lwIP is intended to be extremely memory-efficient, making it ideal for use in embedded devices and other situations with limited resources.

II) High performance: lwIP can process data packets rapidly and effectively despite its tiny size, making it a good choice for real-time applications that need both high performance and low latency.

III) Full TCP/IP stack: The lwIP software comes with a full TCP/IP stack that includes support for the ICMP, IP, TCP, UDP, and DHCP protocols.

IV) Portability lwIP may be readily customized to run on a variety of hardware platforms and operating systems, making it a very portable programmed.

V) Open source: lwIP is distributed under the BSD license, which permits unrestricted usage and software modification.

VI) Support for different network interfaces: lwIP offers support for Ethernet, PPP, SLIP, and Wi-Fi, among other network interfaces. This enables programmers to select the ideal network interface for a given application.

VII) IPv6 support: The most recent iteration of the Internet Protocol, IPv6, is supported by lwIP. Developers are now able to benefit from the newest networking capabilities and technologies.

VIII) Integration with other software components: To offer a full software stack for embedded systems, lwIP may be combined with other software components like RTOSs (Real-Time Operating Systems) and device drivers.

IX) Configurable: lwIP is very adaptable and may be set up to cater to the special requirements of a given application. This includes setting the memory pool's size, turning off unnecessary functionality, and adjusting the TCP/IP stack's specifications.

X) Excellent documentation and community support: lwIP offers excellent documentation and a sizable developer community that supports the project, contributes to it, and shares expertise.

Numerous embedded systems and real-time applications, such as industrial automation, automotive systems, and networked sensors, use lwIP extensively. Developers looking for a compact and effective TCP/IP stack for their applications will find its excellent performance, portability, and small size appealing.

Overall, lwIP is a well-established and popular TCP/IP stack for embedded devices that supports multiple network interfaces, has a full implementation of the TCP/IP protocol suite, is highly customizable, and has solid documentation and community support. Developers looking for a compact and effective networking solution for their embedded systems and real-time applications will find its high performance, portability, and small size appealing.

**Embedded systems and real-time applications are only a few examples of the applications that make use of LWIP. Here are a few examples of programs that make use of lwIP:**

I) Industrial automation: lwIP is frequently used in systems for industrial automation, such as distributed control systems (DCSs), programmable logic

controllers (PLCs), and human-machine interfaces (HMIs). These systems need to communicate reliably and effectively via Ethernet networks, hence lwIP is a good fit for their networking requirements.

II) Vehicle systems: lwIP is utilised in a range of vehicle systems, including advanced driver assistance systems (ADAS), telematics, and entertainment systems. These systems demand high dependability and low-latency communication, hence lwIP is an excellent option for their networking requirements.

III) Medical devices: Patient monitoring systems, medical imaging equipment, and diagnostic systems are just a few examples of the devices that employ lwIP. Due to the high dependability and low latency requirements of these systems, lwIP is an excellent alternative for their networking requirements.

IV) Consumer electronics: lwIP is a technology utilised in many consumer electronics items, such as streaming gadgets, smart TVs, and home automation systems. These goods need networking protocols that are stable and effective, therefore lwIP is an excellent option for their communication requirements.

Overall, lwIP is widely employed in several embedded systems and real-time applications where lightweight, effective, and dependable networking are needed. It is a well-liked option for developers that want a TCP/IP stack for their applications because of its adaptability, flexibility, and simplicity of integration.

**To provide dependable communication in embedded systems, lwIP offers several mechanisms:**

I) TCP protocol: The TCP protocol, which enables dependable, connection-oriented communication between devices, is fully implemented in lwIP. Data is safely transferred over the network via processes like sequence numbers, acknowledgments, and retransmissions, which are all used by TCP.

II) Congestion control: Controlling network congestion is another feature of lwIP that helps prevent packet loss and poor network performance. Network congestion may be detected and avoided using these strategies. These methods include quick retransmission, congestion avoidance, and slow start.

III) Error detection and correction: lwIP contains techniques for error detection and repair, such as checksums and cyclic redundancy checks (CRCs), in transmitted data. These controls aid in ensuring accurate and error-free data transmission.

IV) Flow control: The lwIP protocol contains tools for controlling how data is sent between devices, such as window-based flow control and selective acknowledgments. These controls aid in preventing data overload on devices and ensuring that data is transferred at the proper rate.

Overall, using these mechanisms, lwIP ensures reliable communication in embedded systems. By providing a full implementation of the TCP protocol, managing network congestion and flow control, detecting, and correcting errors, and setting appropriate retransmission timeouts, lwIP enables embedded systems to transmit data reliably over the network.[7]

## 3.5 UART

### 3.5.1 What is UART?

Universal Asynchronous Receiver Transmitter is referred to as UART. A UART is a microprocessor that has been programmed to manage the interface between a computer and any associated serial devices. It may alternatively be described as a programmable baud-rate generator that can generate rates of up to 10 Mbps (divided by 8), or 5 Mbps for ordinary speed (divided by 16). When two devices need to exchange serial data, UART is utilised. UART is primarily used as a tool-to-tool or device-to-device hardware communication protocol by embedded systems, microcontrollers, and computers.

It is asynchronous if a clock signal is not needed. The receiving and transmitting devices must have an identical baud rate because the latter does not have a clock. It is a physical circuit used in microcontrollers or standalone ICs to transmit and receive serial data; it is not a communication protocol. The serial communication protocol is UART. On a device, the UART is frequently referred to as the serial port. The UART uses just two wires for its sending and receiving ends, excluding the ground, of the various communication protocols. The transmitter is identified by the label TX, while the receiver is identified by the label RX. Each device's transmitter and receiver lines serve the primary function of sending and receiving serial data



meant for serial communication. A unit of measurement for transmission rate is the baud rate (Bd). The speed of communication between the transmitter and receiver across the data channel is determined by this parameter. The start and stop bits, which are not data, are included in the baud rate, thus valuable information is conveyed from the

transmitter to the receiver at a somewhat slower pace. 9600 baud is the most often used UART baud rate.

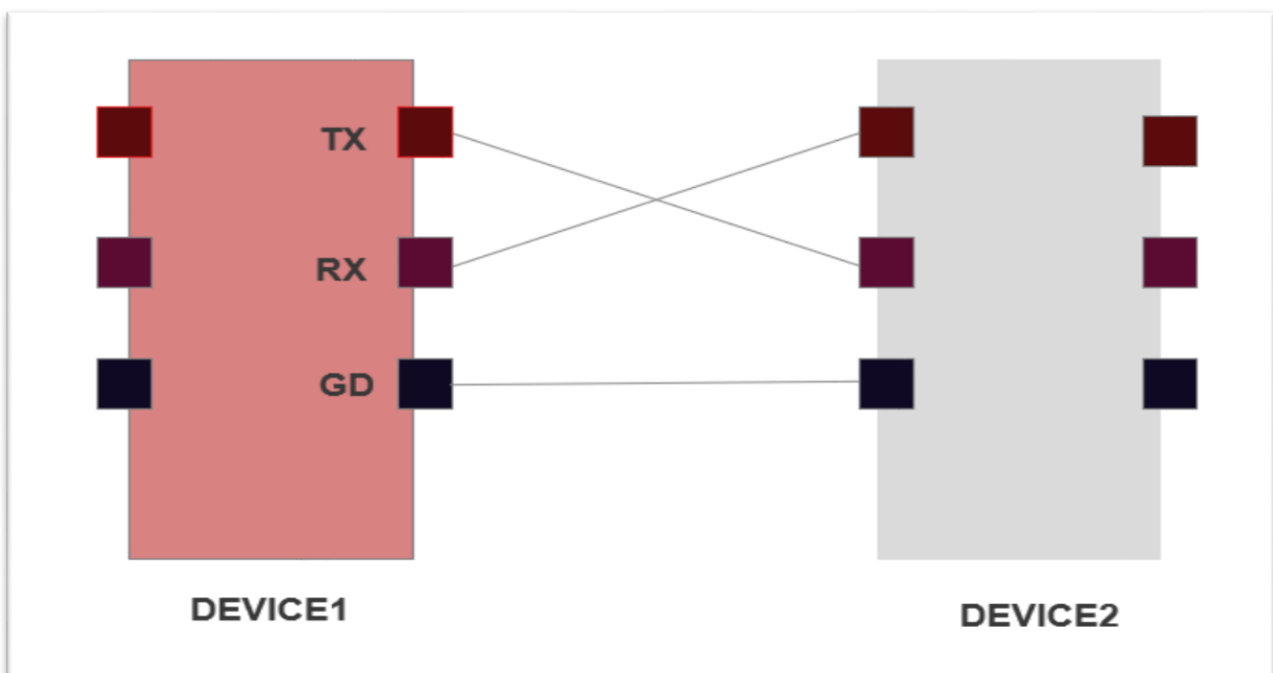


Fig 3.2 simple UART transmission

### 3.5.2 Data transmission

A packet is the manner of transmission used by UART.

Serial packet production and hardware line control are both included in the component that joins the transmitter and receiver.

A start bit, data frame, optional parity bit, and stop bits make up each packet.

#### Start bit:

The start bit marks the beginning of the packet and starts the serial data flow. The transmitting UART pulls the transmission line from high to low for one (1) clock cycle to initiate data transfer. The receiving UART starts reading the bits in the data frame at the frequency of the baud rate as soon as it notices the low voltage.



Fig 3.3 UART packet

**Data frame:**

The data being sent is contained in the data frame. If parity bit is employed, it can range from five (5) to eight (8) bits in length. The data frame can be nine bits long if no parity bit is utilised.

**Parity bits:**

The evenness or oddness of a number is described by parity. To make sure that the receiving data hasn't changed during transmission, the parity bit is employed. Electromagnetic radiation, incompatible baud rates, and long-distance data transmissions can all alter bits.

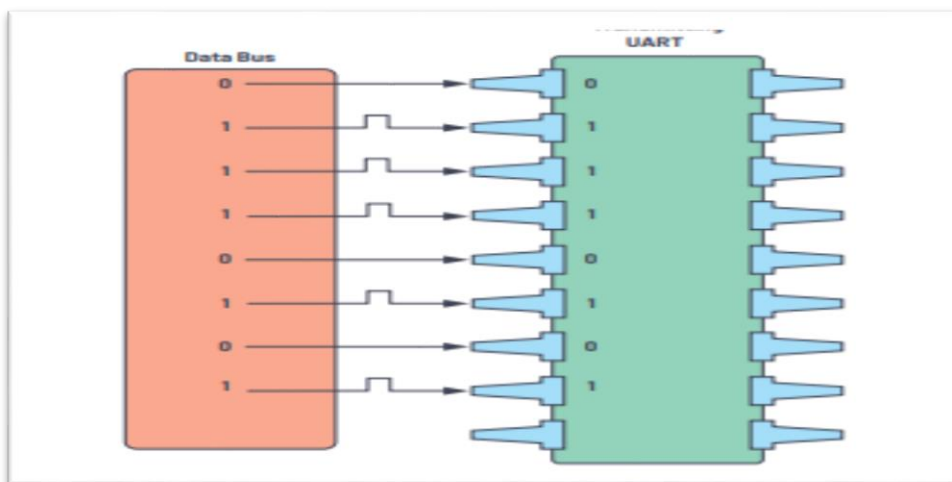


Fig 3.4

**Stop Bits:**

The data packet ends with the stop bit, which might be one or two bits long.

The transmitting UART raises the voltage of the data transmission line from low to high.

**3.5.3 How does transmission work in UART?**

First: Data from the data bus is received by the sending UART.

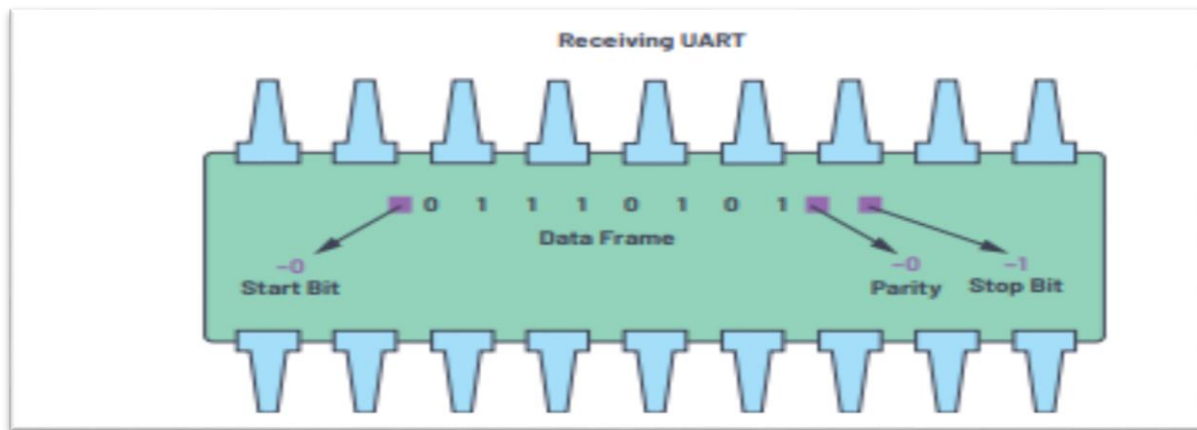


Fig 3.5

Second: The transmitting UART increases the data frame's start, parity, and stop bits.

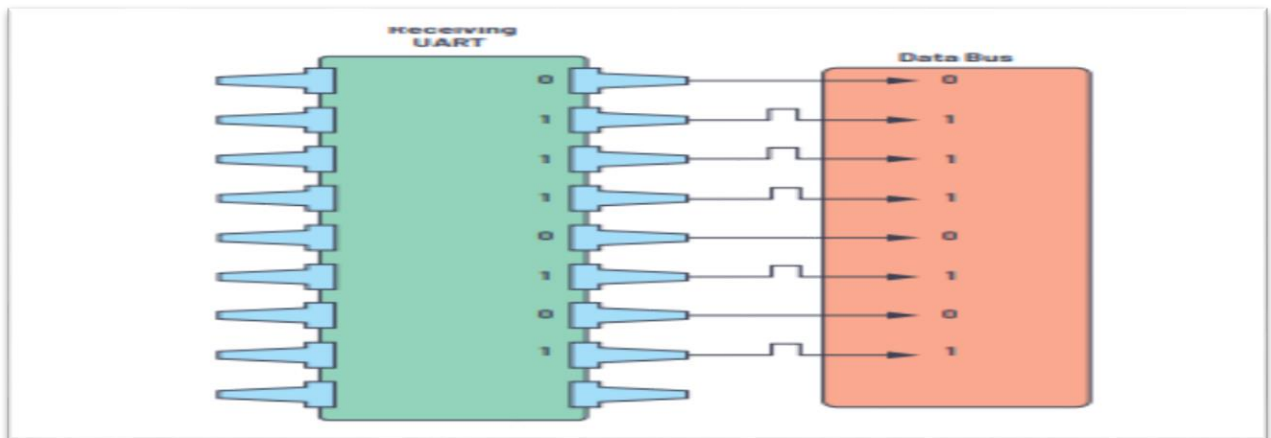


Fig 3.6

Third: The data packet is sent serially from the start bit to the stop bit by the sending UART to the receiving UART.



Fig 3.7

Fourth: To obtain the real data, the receiving UART discards the start, parity, and stop bits from the data frame.

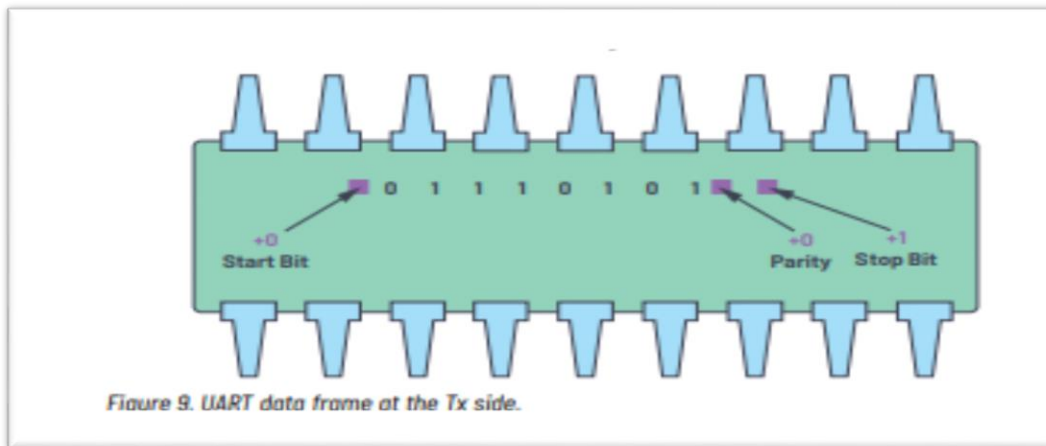


Fig 3.8

Fifth: On the receiving end, the receiving UART transforms the serial data back into parallel and sends it to the data bus. If there is a parity bit, the receiving UART verifies that the data coming from the transmitting UART has been corrected.[2]

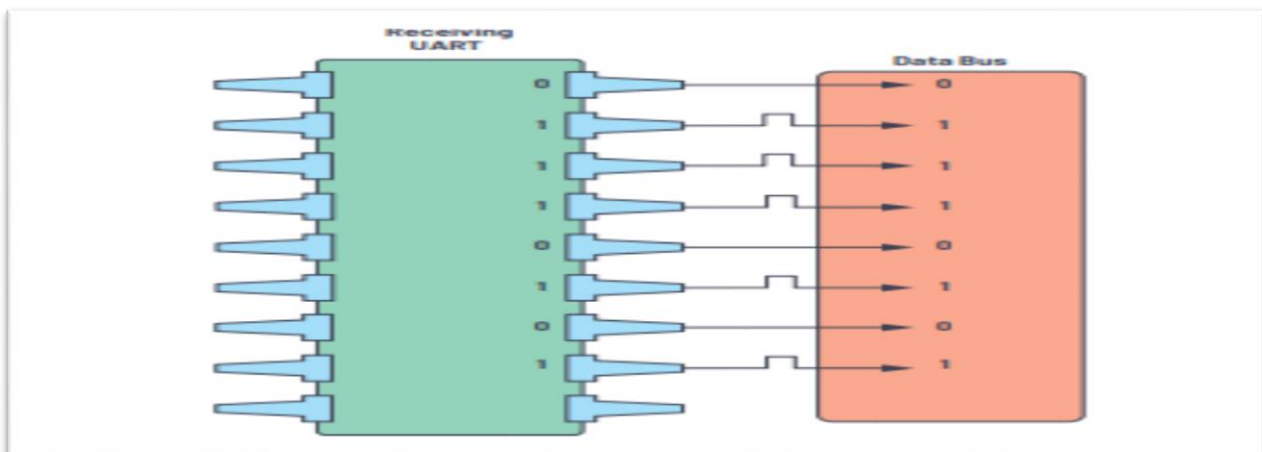


Fig 3.9

# **Chapter 4:**

# **Desktop Application**

#### 4.1 Introduction:

After we knew about our program's purpose and its connection it's important to know how the end user (client) will be able to deal with it and here comes the rule of the most reliable component of our program from the sight of end users which is Graphical User Interface (GUI) as it's hard or impossible for normal clients to deal with CMD or programming languages so we provide an interface for them to make the best use of our program and also make it easier for them to control access to their accounts and easily give orders to the program and in the seek to make all these goals come true we found that we can help users manage their tasks and projects more efficiently by providing GUI.

**Java, C++, and Python are just a few of the programming languages that may be used to create desktop applications. The needs of the application and the level of experience of the development team will determine the programming language to be used. There are several important factors that developers must keep in mind when creating desktop applications, including:**

- 1- Desktop applications must include a graphical user interface (GUI) that enables user interaction with the application. The user interface (UI) must be simple, responsive, and intuitive. When designing the UI, it's critical to take the target audience's needs into account.
- 2- Performance: Desktop applications must be quick and responsive, especially if they are used for labor-intensive tasks like 3D modelling or video editing.
- 3- Designers are required to provide seamless user experience by streamlining the application's code and reducing resource consumption.
- 4- Security: Security risks like viruses and hacking can affect desktop apps. To safeguard the program and its users, developers must add security mechanisms such data encryption and authentication.
- 5- Cross-platform compatibility: Desktop applications may be created with the intention of running on more than one operating system, such as Windows or macOS.
- 6- Apps that are cross-platform can function on a variety of operating systems, broadening their audience, and improving their usability.

## 4.2 Languages used to build the application.

### 4.2.1 Python:

#### 4.2.1.1 Brief history about Python:

The idea for the programming language Python emerged in the late 1980s, and Guido van Rossum at CWI in the Netherlands began work on its implementation in December 1989 as a replacement for ABC that could handle exceptions and communicate with the Amoeba operating system. Van Rossum is the primary creator of Python, and the Python community has given him the nickname Benevolent Dictator for Life (BDFL), which reflects his continued influence over Python's trajectory. (Van Rossum, however, resigned from his position as leader on July 12, 2018). Python was given its name in honor of Monty Python's Flying Circus on the BBC.

In addition to support for Unicode and a cycle-detecting garbage collector in addition to reference counting for memory management, Python 2.0 was introduced on October 16, 2000. The development process itself saw the most significant change, moving towards a more open and community-supported procedure.

After extensive testing, Python 3.0, a significant, backwards-incompatible version, was made public on December 3, 2008. Additionally, many of its key features have been backported to Python 2.6 and 2.7, which are no longer supported but are still backwards compatible.

#### Version 1:

Version 1.0 of Python was released in January 1994. The functional programming tools `lambda`, `map`, `filter`, and `reduce` were the main new features added to this edition. Van Rossum said that "a Lisp hacker who missed them and submitted working patches, Python acquired `lambda`, `reduce()`, `filter()`, and `map()`."

Python 1.2 was the most recent update made while Van Rossum was employed at CWI. At the Corporation for National Research Initiatives (CNRI) in Reston, Virginia, Van Rossum continued working on Python in 1995 and produced several versions from there.

Version 1.4 of Python brought Python a few new features. The Modula-3-inspired keyword arguments (which are also comparable to the keyword

arguments in Common Lisp) and integrated support for complex numbers stand out among these. A simple method of data concealment through name mangling is also there, although it is readily circumvented.

### **Version 2:**

List comprehensions, a feature taken from the functional programming languages SETL and Haskell, were added to Python 2.0 in October 2000. Except for Python's preference for alphabetic keywords over punctuation characters, Haskell's syntax for this construct is remarkably identical to Python's. Additionally, a garbage collector that can gather reference cycles was added to Python 2.0.

Python 2.0 and Python 2.1 were very similar to one another. The Python Software Foundation License was given to it. The Python Software Foundation (PSF), a non-profit organization founded in 2001 and modelled after the Apache Software Foundation, owns all code, documentation, and specifications contributed after the alpha release of Python 2.1. The language definition was modified as part of the release to enable nested scopes, just as other statically scoped languages. (Until Python 2.2, the functionality was optional and by default turned off.)

The integration of Python's types (types written in C) and classes (types written in Python) into one hierarchy was a significant advance in Python 2.2, which was published in December 2001. Python's object model became exclusively and consistently object oriented because of this single unification. [ Generators that drew inspiration from Icon were further included.

The with statement, which allows Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom, was added to Python 2.5 in September 2006. It encloses a code block within a context manager (for instance, acquiring a lock before the block of code is run and releasing the lock afterward, or opening a file and then closing it).

A "warnings" mode in Python 2.6 emphasized the usage of features that were deleted in Python 3.0 while also including certain features from that version. Python 2.6 was published concurrently with Python 3.0. Like Python 3.1, which was published on June 26, 2009, Python 2.7 was concurrent with it and had features from it. Python 2.7 was the final release in the 2.x series when the parallel 2.x and 3.x releases stopped.



Although Python 2.7 would continue to be supported through 2020, users were urged to upgrade to Python 3 as soon as possible. On January 1, 2020, the code for the Python 2.7 development branch was also frozen. On April 20, 2020, a final release known as 2.7.18 was made available. It contained solutions for crucial problems and release blocks. Python 2's lifespan came to an end as a result.

### **Version 3:**

On December 3, 2008, Python 3.0 (sometimes known as "Python 3000" or "Py3K") was made available. The improvements necessary could not be made while maintaining complete backwards compatibility with the 2.x series, hence a new major version number was required. It was created to fix fundamental design faults in the language. "Reduce feature duplication by removing old ways of doing things" was the Python 3 guiding philosophy.

Like earlier versions, Python 3.0 was created with this idea in mind. Python 3.0 placed a strong emphasis on reducing superfluous constructs and modules, in keeping with the Python adage that "There should be one—and preferably only one—obvious way to do it". This was because Python has amassed new and redundant methods to do the same task.

Python 3.0 was still a multi-paradigm language, nevertheless. The object-oriented, structured, and functional programming paradigms are still available to programmers, among others, but Python 3.0 was designed to make the details clearer than Python 2.x did.[8]

#### **4.2.1.2 How did we use Python to build our Desktop Application**

In our app we used python because it was the most suitable language for our requirements. Also, the simplicity of Python makes it a great choice for developing desktop applications. Python is a popular language among developers of all skill levels because of how easy it is to understand and use. Python's syntax is simple to understand and read, which makes writing and maintaining code simple. Additionally, a variety of libraries and frameworks are available in Python that can be used to streamline the development process.

Python is a cross-platform language, meaning programs developed in it can run on Windows, macOS, and Linux, among other operating systems.

Without having to worry about platform-specific issues, it is now simpler to create desktop applications that can be used by a larger audience.

**But like everything else python has some challenges like:**

- I) It may need more time than compiled languages like c++ or java what can be a problem for high performance apps like video games.
- II) It uses third party library or framework to create user interface and because of the several options this may add complexity to the development process and may require more learning to keep up and make best use of the tools.

And speaking about python frameworks here are some examples of them: -

- I) PyQt: PyQt is a group of Python bindings for the Qt software development kit. Qt is a multi-platform UI framework that offers many different tools for creating desktop programs. PyQt offers a full set of bindings for Qt, making it simple to build intricate user interfaces and incorporate them with other Qt-based software.
- II) wxPython: The wxWidgets C++ library has a set of Python bindings called wxPython. A cross-platform GUI toolkit called wxWidgets offers a variety of widgets and building blocks for creating desktop programs. The full set of wxPython bindings for wxWidgets makes it simple to build intricate user interfaces and combine them with other wxWidgets-based programs.
- III) Kivy: Based on OpenGL, Kivy is a cross-platform UI toolkit. It offers a variety of widgets, tools, and support for touch-based user interfaces for desktop programs. Kivy is designed to be simple to use and works with many different operating systems, including Windows, macOS, and Linux.
- IV) Tkinter: A common Python library for building graphical user interfaces (GUIs) is Tkinter. It is a straightforward framework that gives programmers the means to build desktop apps with a huge selection of widgets.

## 4.3 Libraries used in the Desktop application.

### 4.3.1 Tkinter

Now that we know about different frameworks, we chose to use Tkinter as it satisfies all our requirements also, Tkinter is a flexible and easy-to-use framework for creating GUI applications with Python. It offers a large range of customization choices and widgets to developers, and its cross-platform interoperability makes it a popular option for building straightforward apps that work across multiple operating systems. Tkinter is a fantastic option for novices and developers who wish to build straightforward desktop programs fast and efficiently, even though it might not be appropriate for all projects.

**Tkinter also provide some advantages that any developer may need in his work like:**

- Tkinter is a cross-platform, therefore programs created using it may operate on Windows, macOS, and Linux, among other operating systems. Tkinter offers a uniform appearance and feel on several platforms.
- It's popular among novices who want to learn how to use Python to build desktop apps since it's simple to understand. The framework offers a large range of documentation and examples and has a straightforward syntax.
- It is quite adaptable. Widgets' look and behavior may be modified by developers using a variety of techniques and attributes. Additionally, Tkinter supports event handling, enabling programmers to react to user actions like button clicks and mouse movements.[3]

### 4.3.2 Custom TKinter

One of the challenges that stopped Tkinter framework from being used is its look as now we need to use modern look widgets and to solve this problem one developer (Tom Schimansky) came up with a new algorithm that modified the look of the widgets and added some new functionalities to our desktop app.

Next to our main framework (Tkinter) there are some libraries that we used to produce our app and each one of them had a great role so let's talk about them:

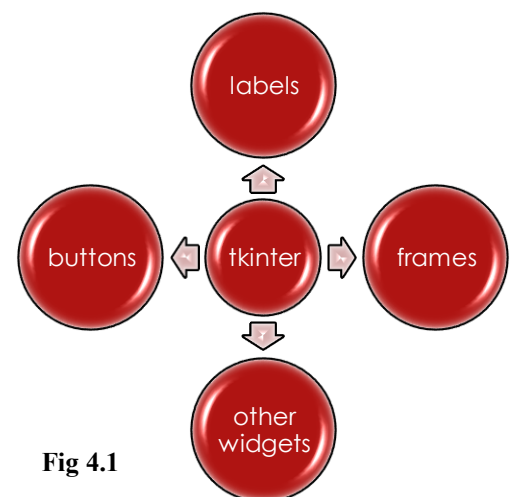


Fig 4.1

Libraries of  
TKINTER

### 4.3.3 Python image library:

A library that offers a variety of tools and methods for image processing to developers. PIL is a well-known library that is frequently used in many different applications, including web development, academic research, and digital art.

And some of its advantages are: -

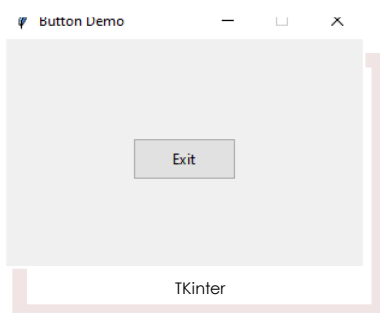
I) Its usability. The library features a straightforward and user-friendly API that enables developers to begin working on image processing tasks with ease. Additionally, PIL offers a wealth of examples and documentation to aid developers in understanding how to use the library efficiently.

The library gives programmers the resources they need to handle photos fast and effectively and is speed optimized. Developers may utilize several processors to accelerate image processing workloads thanks to PIL's support for multi-core processing.

### 4.3.4 Time library:

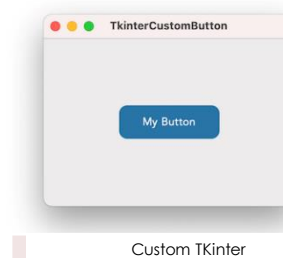
A common Python package that gives programmers the means to interact with time-related actions and functions. Python's time module may be used to calculate how long code takes to execute, add delays, and carry out other time-related operations.

**Some of its features are:**



TKinter

**Fig 4.2 Using tkinter**



Custom  
TKinter

**Fig 4.3 Using  
Custom  
TKinter**



Fig 4.4 Sign in Page

- The `time ()` method, which gives the current time (as of January 1, 1970, 00:00:00 UTC) in seconds since the epoch. By computing the difference between the start and finish timings of a code block, the `time ()` method may be used to calculate the length of time it takes for code to execute.

- Other functions for working with time values are also available in the time library, including `gmtime ()` and `local time ()`, both of which return the current time in a structured format. The `sleep ()` method, which pauses a program's execution for a given number of seconds, may be combined with the time library to generate time delays.

- Using the `strftime ()` function, time values can be formatted in a simple way for users. Developers can provide a format string for the time value using the `strftime ()` method, which includes choices for the date, time, and time zone.

#### 4.4 Desktop application design:

After deciding both programming language and framework we are going to use in our app it's time to discuss the components of our app: -

**First:**

**We shall start with our sign in page that has a main purpose which give users access to the app and its content and to design it there are some features or let's say rules that we should stick to:**

- Make sure to keep it simple and easy to use and navigate.
- Use sentences that directly describe what you need the user to do and avoid using complex terms.

- Provide clear feedback to the user like error messages in case of using wrong password or signing in with non-reserved information.
- Keep the security of your information by any action you find suitable like showing password in the shape of stars (\*).
- And this may be a common feature between all pages which is making sure all the questions that come in the user's mind will be easily answered from the design.

Now we shall talk about the main page of our app that contains all the tools our user will need to communicate with the operating system and may give the accessibility to modify it or give commands what will reduce a lot of time.

So instead of talking about it as one thing let's talk about each frame alone and clarify its components to show its importance: -

**1. First, we shall speak about the left frame which includes buttons that can be described as the user's tools.**

- TASKS button which declares the tasks that are currently being worked on.
- KNOW button in case of new user this button will open a window that contains all the information about the tasks we have.
- HELP button which will provide all communication information of the programmer in case there is a problem hard to solve with the user.
- LOGOUT button like the name says it's used to close the account to exit the program or switch accounts.

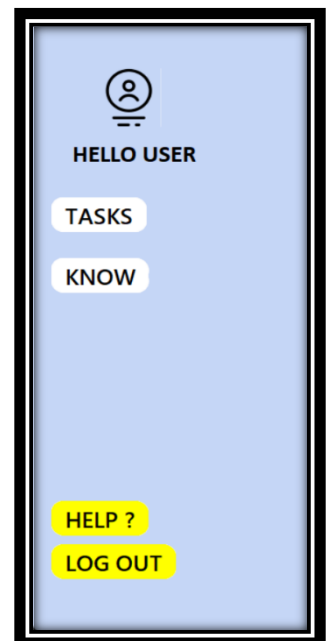


Fig 4.5

**2. Second is the up frame which only includes two objects:**

I) A text widget is used to save the notes the user takes in a word file out of the program using a combination of get and create functions.

II) A clock of real time that we used some functions from time library we talked about before to implement and show as it upgrades its value every 1 second.

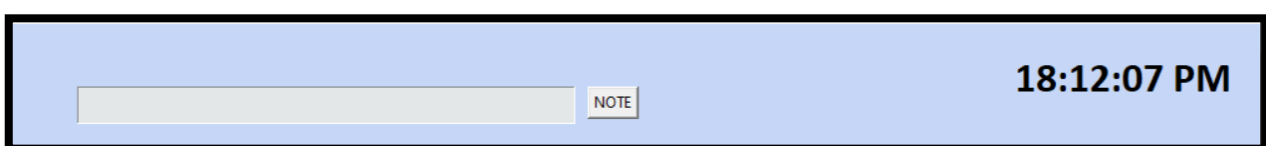


Fig 4.6 Showing Time and text widget.



**Fig 4.7 Showing the main page of the desktop.**

Finally, the main frame includes the tools of the user or in other words they are the tasks that our program provides, and it can vary from one usage to another depending on the goal that the user wants to reach.

In this example we showed the tasks that any car can provide and it can be modified to suit any program you need depending on the environment surrounding the OS we have and the only task that can be said to be common between them all is the update widget as it provides a very helpful task that cooperates with the HELP button we talked about before as after you communicate with the programmer you may need to write code on the CMD window and using this update widget you shall pen a text widget that can be considered a CMD and will send all the commands that the user types to the OS.

## **4.5 APIs (Application Programming Interface)**

### **4.5.1 What is an API?**

Application Programming Interface is referred to as API. It is a collection of standards, tools, and protocols for creating software applications that enables interaction across various software platforms.

APIs provide a mechanism for developers to access and utilise the functionality of another program or service by defining how various software components should communicate with one another. It is possible to link disparate systems and create more complicated applications by using APIs to access data, functionality, or services offered by another program or system.

APIs come in a variety of shapes and sizes, including web APIs (which communicate over HTTP or HTTPS protocols), operating system APIs (which let software components interface with the underlying operating system), and database APIs (which give access to databases).

To give users access to online services like social networking platforms, payment processors, or weather information, APIs are frequently utilised in web development. These services make it simple for developers to include their functionality into their own apps by offering a well-defined API.

A common use case or feature for an API is to provide a standardized mechanism for developers to communicate with the underlying system or service. The system's capabilities may be accessed, actions may be carried out, or data may be retrieved.

Both internal (such as within a firm) and external (such as third-party developers) users can utilise APIs. Private APIs are only used internally, whereas public APIs are accessible to other developers.

Several protocols, including HTTP, HTTPS, SOAP, and REST, can be used to access APIs. With HTTP as the communication protocol, REST (Representational State Transfer) is a well-liked architectural style for creating web APIs. It offers a few limitations for creating scalable and maintainable APIs.

Access to a variety of services and capabilities may be made available through the usage of APIs. The processing of payments (using Stripe, for instance), social media sites (using Twitter, Facebook), weather information (using OpenWeather), and search engines (using Google), are typical instances.

APIs may be made money from in several ways, such as by charging for access, giving usage-based pricing tiers, or offering a free tier with constrained capability.

For creating complicated applications and connecting many systems and services, APIs may be an effective tool. Developers must make sure they are correctly authenticating and safeguarding API calls, resolving failures and exceptions, and dealing with modifications or upgrades to the API over time. As a result, they can potentially pose security and dependability risks.



Overall, APIs are essential to modern software development because they give programmers a method to combine the functionality of many systems and services to create more sophisticated and potent applications. In order for developers to access and utilise the functionality offered by other systems and services; APIs are a crucial part of contemporary software development. Developers may create more sophisticated and potent apps while spending less time and effort creating and maintaining their software by using APIs.

#### **4.5.2 Why APIs are useful?**

**There are many reasons to utilise APIs in software development, however the following are some of the more popular ones:**

I) Through APIs to access third-party functionality Third-party services and systems, such as social networking platforms, payment processors, weather data, and mapping services, all provide functionality that may be accessed through APIs. Developers may incorporate these services into their own apps and provide their consumers with more functionality and features by leveraging APIs.

II) Creating scalable and adaptable applications: APIs may be utilised to decouple various application components, enabling their independent development and scalability. This can facilitate the development of more intricate and scalable applications as well as the ability to modify or upgrade the program over time.

III) Enabling cross-platform integration: APIs may be used to facilitate integration across several platforms or systems, enabling the sharing of data and functionality among numerous apps and gadgets. An API could be used, for instance, to facilitate integration between a web application and a desktop program or between a web application and a mobile application.

IV) Providing a standardized interface: By standardizing how various systems or services communicate with one another, APIs facilitate the development of integrations and guarantee the interoperability of various parts. APIs can also make it simpler for developers to comprehend and utilise the functionality offered by other systems by creating a uniform interface.

V) Software monetization: APIs may be used to generate income by charging a fee for access, giving usage-based pricing tiers, or offering a free

tier with constrained capabilities. Companies may be able to monetize their software and create new income streams in this way.

All things considered, APIs are an effective tool for creating more intricate, versatile, and scalable applications as well as for connecting various systems and services. Developers may create more robust apps more quickly and simply utilising APIs, while also giving their consumers greater functionality and features.

**Here are a few other well-known APIs that you might be familiar with:**

- Google Maps API: Gives users access to the Google Maps service, enabling embedding of maps, geocoding of addresses, and spatial querying by developers.
- Facebook API: Offers access to the Facebook social media network, enabling programmers to use the Facebook Graph API and obtain user information, upload material, and interact with it.
- The Stripe API gives developers access to the Stripe payment processing service, enabling them to manage subscriptions, take payments, and carry out other payment-related operations.
- Twilio API: This tool grants developers' access to the Twilio communication platform, enabling them to conduct phone conversations, send and receive SMS and MMS messages, and carry out other communication-related operations.
- Github API: Gives developers access to the Github platform for hosting code, enabling them to get information, manage repositories, and carry out other coding-related operations.
- OpenWeatherMap API: Gives developers access to weather information and forecasts so they may create weather-related software and services.
- Spotify API: Provides developers with access to the Spotify music streaming service, enabling them to obtain information about artists, albums, and songs as well as manage playing and create apps that are linked to music.

### 4.5.3 Requests

The requests library is a Python library that allows you to send HTTP/1.1 requests extremely easily. While still offering cutting-edge features for more complex use cases, it is made to be straightforward and simple to use. Many of the difficulties associated with sending HTTP requests—such as dealing with cookies, headers, and authentication—are abstracted away by the library.

**The requests library has the following salient characteristics:**

- **Simple API:** The library offers methods like `get()`, `post()`, `put()`, and `delete()` along with a straightforward and user-friendly API for sending HTTP queries.
- **Complete support for the HTTP/1.1 protocol,** including features like persistent connections, chunked encoding, and content negotiation. This library offers complete support for the HTTP/1.1 protocol.
- **Automatic connection pooling:** The library pools connections to the same host automatically, saving time by avoiding the need to create new connections for each request.
- **Customizable timeouts:** The library enables you to establish specific timeouts for requests, preventing requests from hanging indefinitely in the event that a server is sluggish to answer.
- **Support for authentication:** A variety of authentication techniques, including HTTP basic authentication, OAuth 1.0a, and OAuth 2.0, are supported by the library.
- **Built-in JSON support:** The library has built-in encoding and decoding support for JSON content, making it simple to use APIs that return JSON data.

Overall, the Python requests package provides a strong and adaptable tool for sending HTTP requests. The requests library may assist you in completing your task quickly and effectively whether you're creating a straightforward script to scrape data from a website or a sophisticated web application that must communicate with APIs.

```
import requests
url='http://www.Google.com'
#url = 'https://date.nager.at/api/v2/publicholidays/2020/US'
response = requests.get(url)
print(response) # print HTTP status code
var = response.status_code# print HTTP status code
print(var)
print("-----")
print(dir(response))
print("-----")
print(response.text)
print("-----")
```

#### Code 4.1 A simple API

This bit of Python code sends an HTTP GET request to the given URL (<http://www.google.com>) using the requests module, then prints the result.

#### The code is broken down as follows:

- The first line imports the requests library, which gives Python programmers a straightforward interface for sending HTTP requests.
- The URL is set to <http://www.google.com> in the second line.
- The third line sends an HTTP GET request to the provided URL using the `requests.get()` method, then stores the result in the response variable.
- The response's HTTP status code, which ranges from 400 to 599 and indicates whether the request was successful (status code 200) or met an issue, is printed on the fourth and fifth lines.
- The sixth line produces a line of dashes to delineate the response object's characteristics and methods from the status code.

- The response object's characteristics and methods are listed on the seventh line. They include items like the response headers, cookies, and content.
- The content of the response is shown as a string on the eighth line.

The line of code that sets the URL to `https://date.nager.at/api/v2/publicholidays/2020/US`, an illustration of an API endpoint that gives details on public holidays in the United States for the year 2020, is commented out. The code will make a call to this API endpoint if you uncomment this line and comment out the line before it that sets the URL to `http://www.google.com`.

For requesting online resources, the requests library offers several HTTP methods. The requests library supports the following HTTP methods in addition to the GET method used in the sample code you gave:

- POST: Applied to sending information to a web site. This technique is frequently used to upload files to a server or submit form input.
- PUT: A web server's method for updating an already-existing resource. On servers, this approach is frequently used to update files or data.
- DELETE: This command is used to remove resources from a web server.
- HEAD: Comparable to GET but only returns the resource's headers, not its content.
- choices: Used to find out information about a resource's communication choices.
- PATCH: A web server's resource can be partially updated using this method.

You may call the appropriate method on a requests object to utilise certain HTTP methods with the request's library, for instance:

```
import requests

url = 'http://example.com'

# Make a POST request
response = requests.post(url, data={'key': 'value'})

# Make a PUT request
response = requests.put(url, data={'key': 'value'})

# Make a DELETE request
response = requests.delete(url)

# Make a HEAD request
response = requests.head(url)

# Make an OPTIONS request
response = requests.options(url)

# Make a PATCH request
response = requests.patch(url, data={'key': 'value'})
```

Code 4.2 showing Different Types of requests

#### 4.5.4 OTP Service

OTP stands for One-Time Password, which is a temporary password that is generated for a single use and expires after a short period of time. OTPs are commonly used for authentication purposes to verify the identity of a user or to provide an additional layer of security during login or transaction processes.

An OTP is typically generated by a server or service and sent to the user's device via SMS, email, or a mobile app. The user then enters the OTP into a form or application to complete the authentication or transaction process. Since an OTP is only valid for a short period of time and can only be used once, it provides a higher level of security than a static password that can be reused or easily stolen.

OTP can be generated using various algorithms, such as time-based OTP (TOTP) or HMAC-based OTP (HOTP), which use a combination of a secret key, a timestamp, and a hash function to generate a unique OTP for each use. Services like Twilio, which is used in the code you provided, provide APIs for generating and sending OTPs to users.

**here are some additional details about OTPs:**

1. **Types of OTPs:** There are several types of OTPs, including time-based OTPs (TOTP), HMAC-based OTPs (HOTP), and challenge-response OTPs. TOTP uses a timestamp and a secret key to generate a unique password that changes every 30 seconds. HOTP uses a counter and a secret key to generate a unique password for each use. Challenge-response OTPs use a challenge generated by the server and a response generated by the user to authenticate the user.
2. **Security benefits:** OTPs provide an additional layer of security compared to traditional passwords, as they are only valid for a short period of time and can only be used once. This means that even if an attacker intercepts an OTP, they will not be able to use it to gain access to the user's account or complete a transaction.
3. **Implementation:** OTPs can be implemented using various methods, including SMS, email, mobile apps, or hardware tokens. SMS and email are the most common methods, but they are also the least secure, as they can be intercepted or spoofed. Mobile apps and hardware tokens provide a higher level of security, as they generate OTPs locally on the user's device and do not rely on communication with a remote server.
4. **Challenges:** OTPs are not without their challenges, as they can be subject to various attacks, such as phishing, man-in-the-middle, and replay attacks. In addition, the user experience of entering OTPs can be cumbersome, especially for users who have limited access to their devices or have disabilities.

Overall, OTPs provide a higher level of security compared to traditional passwords and are commonly used for authentication and transaction purposes. However, they are not without their challenges and should be implemented carefully to ensure maximum security and usability.

CODE

```

import http.server
import socketserver
import socket
from twilio.rest import Client
import os
from twilio.rest import Client

account_sid = ""
auth_token = ""

```

**Code 4.3 Showing the libraries used for the OTP Service.**

```

temp_client = Client(account_sid, auth_token)
service = temp_client.verify.v2.services.create(
    friendly_name='SquadRTOS system'
)

verify_sid = service.sid

```

This code initializes a Twilio client using the `Client` class from the `twilio.rest` module. It uses the account SID and auth token provided by the Twilio account to authenticate the client.

The code then creates a new verification service using the `temp_client.verify.v2.services.create` method. The `friendly_name` parameter is used to provide a descriptive name for the service, which can be used to identify it in the Twilio console.

Once the verification service is created, its SID is saved in the `verify_sid` variable, which is later used by the `send_otp` and `verify_otp` functions to interact with the service.

Overall, this code sets up the Twilio client and creates a verification service, which can be used to send and verify OTPs for authentication and transaction purposes.



```

def send_otp(NumberPhone):
    global verified_number
    verified_number = NumberPhone
    verification = temp_client.verify \
        .services(verify_sid) \
        .verifications \
        .create(to=verified_number, channel="sms")
    print(verification.status)

def verify_otp(OTP):
    print(verified_number)
    verification_check = temp_client.verify.v2.services(verify_sid) \
        .verification_checks \
        .create(to=verified_number, code=OTP)
    return (verification_check.status)

```

**Code 4.4 Showing the main functions for sending OTP and verifying OTP.**

These are two functions that use the Twilio client and verification service created in the previous code block to send and verify OTPs.

The `send_otp` function takes a phone number as input and sends an OTP to that number using the verification service created earlier. The `to` parameter specifies the phone number to send the OTP to, and the `channel` parameter specifies the channel to use for sending the OTP (in this case, SMS).

The function then saves the verified phone number in the `verified_number` variable for later use.

The `verify_otp` function takes an OTP as input and verifies it using the verification service created earlier. The `to` parameter specifies the phone number to verify the OTP for (which should be the same as the one used to send the OTP), and the `code` parameter specifies the OTP to verify.

The function then returns the status of the verification check (either "approved" or "pending").

Overall, these functions provide a simple interface for sending and verifying OTPs using the Twilio verification service, which can be integrated into larger projects or applications for authentication and transaction purposes.

```

class MyHandler(http.server.SimpleHTTPRequestHandler):
    def Response200(self):
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.send_header('Access-Control-Allow-Origin', '*')
        self.end_headers()
    def do_GET(self):
        print(self.path)
        API = ''
        for chr in self.path:
            if chr == '?':
                break
            else:
                API+=chr
        # print('Headers :',self.headers)
        params = self.path.split('?')
        if len(params) > 1:
            params = params[1].split('&')
            params = {p.split('=')[0]: p.split('=')[1] for p in params}
        else:
            params = {}
        if str(API) == '/send_otp':
            send_otp('+2'+params['phone'])
            self.Response200()
            self.wfile.write(b'OTP Sent')
        elif str(API) == '/check_otp':
            status = verify_otp(str(params['OTP']))
            if status == 'pending' :
                self.Response200()
                self.wfile.write(b'Wrong OTP!')
            elif status == 'approved':
                self.Response200()
                self.wfile.write(b'Accepted!')

```

Code 4.5 Showing Custom Handler for HTTP server.

This code defines a custom handler for the HTTP server that can handle incoming HTTP GET requests and route them to the appropriate API endpoint.

The MyHandler class inherits from the http.server.SimpleHTTPRequestHandler class and overrides its do\_GET method. This method is called by the HTTP server whenever an HTTP GET request is received.

The do\_GET method first prints the requested path to the console for debugging purposes. It then parses the path to extract the API endpoint and any query parameters.

If the API endpoint is /send\_otp, the method calls the send\_otp function with the phone number specified in the query parameters. It then sends an HTTP 200 response with the message "OTP Sent" using the Response200 method defined in the class.

If the API endpoint is `/check_otp`, the method calls the `verify_otp` function with the OTP specified in the query parameters. If the OTP is correct, the `verify_otp` function returns "approved" and the method sends an HTTP 200 response with the message "Accepted!". If the OTP is incorrect, the `verify_otp` function returns "pending" and the method sends an HTTP 200 response with the message "Wrong OTP!".

Overall, this code provides a simple HTTP API for sending and verifying OTPs using the `send_otp` and `verify_otp` functions defined earlier. It can be used as a basis for building more complex web applications or APIs that require OTP authentication or transaction verification.

```
PORT = 8080
```

```
with socketserver.TCPServer(("", PORT), MyHandler) as httpd:
    ip_address = socket.gethostbyname(socket.gethostname())
    print(f"Serving at http://{ip_address}:{PORT}/")
    httpd.serve_forever()
```

---

**Code 4.6 Showing initialization of HTTP server using `socketserver.TCPServer`**

This code initializes an HTTP server using the `socketserver.TCPServer` class and the custom `MyHandler` class defined earlier. The server listens on port 8080 and binds to all available network interfaces (`""`).

The `ip_address` variable is set to the IP address of the machine running the server, which is determined using the `socket.gethostbyname` method. The server is then started using the `httpd.serve_forever` method, which listens for incoming HTTP requests and passes them to the `MyHandler` class for processing.

Overall, this code provides a simple way to start an HTTP server that can handle incoming requests and route them to the appropriate API endpoint using the `MyHandler` class. The server will continue running until it is interrupted or terminated.

## Summary

This code sets up an HTTP server that can send and verify OTPs using the Twilio verification service, in response to incoming HTTP GET requests.

The code starts by importing the necessary modules, including the `http.server`, `socketserver`, `socket`, and `twilio.rest` modules. It then sets up the Twilio client and verification service, using the Account SID and Auth Token for authentication.

The `send_otp` and `verify_otp` functions are defined to send and verify OTPs using the Twilio verification service, as described earlier. These functions are used by the `MyHandler` class to handle incoming HTTP GET requests.

The `MyHandler` class inherits from the `http.server.SimpleHTTPRequestHandler` class and overrides its `do_GET` method. This method routes incoming HTTP GET requests to the appropriate API endpoint (either `/send_otp` or `/check_otp`) and calls the `send_otp` or `verify_otp` function as necessary. The `Response200` method is used to send an HTTP 200 response with the appropriate headers.

Finally, the code sets up an HTTP server using the `socketserver.TCPServer` class and the `MyHandler` class, and starts the server to listen for incoming HTTP requests.

Overall, this code provides a simple way to set up an HTTP server that can send and verify OTPs using the Twilio verification service, which can be used as a basis for building more complex web applications or APIs that require OTP authentication or transaction verification.

# Chapter 5: Security

## 5.1 Security concepts and their importance

Security is mainly protection against any potential attack, but security has a wide range of branches that goes beneath it for example: -

- Freedom to do what you want with your app can be harmed in case of lack of security.
- The presence of your data and tools is essential and can be affected with security.
- Resilience against any potential attacks.
- Secrecy of data transmitted over communication.
- And even the state of mind can be harmed in case of lack of security.

Now we can conclude these issues in the CIA triad and every concept in this triad can be said to be complement to the other two and they can be described as: -

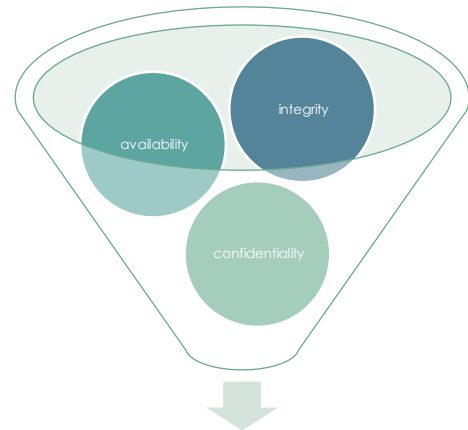


Fig 5.1 CIA triad

1. Integrity: entails ensuring that data is reliable, consistent, and accurate across its entire lifespan. Data cannot be modified while in transit, and precautions must be made to prevent unauthorized parties from changing it (for instance, in violation of confidentiality).
2. Availability: implies that information should always be easily and consistently available to authorized people. This entails keeping up with the systems, hardware, and technological infrastructure that store and show the data.
3. Confidentiality: is like privacy in most ways. Measures for maintaining confidentiality are intended to guard against unauthorized access to sensitive data. Data is frequently categorized based on the scope and nature of the harm that might result from it getting into the wrong hands.[11]

Now that we know about the three basic concepts of security, we shall discuss how we implemented each of them and we shall start with confidentiality:

The first step we took to implement confidentiality is that we used a server that generates a token that consists of 128 bit of digits that changes every time our user needs to access the services that we provide and not only

this we also use face recognition to make sure of the permissions of this user and the process goes like this:

1. The user provides his face signature using the site we provide him with.
2. The face signature is then sent to the server and checked.
3. It's checked by comparing this signature with the ones in the dictionary connected to the server.
4. If the signature is found, then the server generates the token using the algorithm we provided it with.
5. After this the server sends the token to both the user and to the web site server to give him access to transmit through pages.

```
def generate_auth_token():
    alphabet = string.ascii_letters + string.digits
    token = ''.join(secrets.choice(alphabet) for i in range(128))
    return token
```

**Code 5.1 showing "TOKEN GENERATION"**

**Then the second parameter in the CIA triad is integrity:**

Integrity can't be a problem for us as we don't provide a data base that can be infected by any attacker also our code is open source so it can be modified by any one that wants to provide any updates but also to make sure that the data of our users is safe we keep it encrypted as integrity is not only about keeping the code safe but also the user's data.

**Finally, we shall speak about the availability in our system:**

Like we said before availability is mainly concerned with providing our users with the services they need in the time they want and to make sure that this works right in our system with no traffic that can block the users or cause other problems we made sure that there's a time out period for each user as if they left the system with no orders pr requests it will automatically clear the place it occupies for other users to prevent any unwanted traffic.

## **5.2 Different ways of providing security.**

### **I) Login page in desktop app: -**

It's a basic sign in page that takes the user's username and password and compare them with the ones stored in the dictionary and do action depending on the result of this comparing process: -

1. If the user and password are found, then permissions are given to login the app.
2. If they are not found, then you receive an error message box and, in this case, you shall have the ability to create a new user if you don't have one.

## **II) Face recognition:**

face recognition is one of the main branches of deep learning that we can deal with and to implement face recognition in our project, we used a library that python provides called **Face\_recognition** and using it we went through the process of face recognition using the next steps:

1. Capture a frame using your camera from the page and send it to server to be processed.
2. Only process every other frame of video to save time.
3. Resize the frame of video to 1/4 size for faster face recognition processing.
4. Convert the image from BGR color (which OpenCV uses) to RGB color (which face recognition uses).



# Chapter 6: Webpage

## **6.1 What is a Webpage?**

A web page or web page is a document viewed using an Internet browser. Information found on a web page is usually in HTML or XHTML format. Web pages also typically contain other resources, such as style sheets, scripts, and slideshows. Users can navigate to other pages using hypertext links. A website can be accessed by typing the URL into the browser's address bar. The website may contain text, graphics and hyperlinks to other websites and files. The site provides information such as images or videos to illustrate important topics to viewers. The website may also be used to sell products or services to viewers. When you click on a link provided by a search engine, you will be taken to a web page. Website information is displayed on the web using browsers such as Internet Explorer, Mozilla Firefox, or Google Chrome. The web browser is connected via HTTP to the web server where the website content is hosted. Each web page corresponds to a different type of information that is presented to the visitor in a visual and readable way.

## **6.2 Languages used to create a Webpage.**

### **6.2.1 JavaScript**

JavaScript is a language that dates to the creation of the first online browsers. In 1994, Netscape Communications Company produced Netscape Navigator, which in the 1990s rose to become the most widely used web browser.

The company's board of directors immediately understood that browsers should allow the creation of more dynamic websites and the performance of various server-side language functions, such as input validation. First, Sun Microsystems and Netscape Communications worked together to integrate Java into Netscape Navigator. Next, they desired to incorporate an existing programming language, such as Scheme, Perl, or Python. At some point, they made the decision to develop a scripting language that would be compatible with Java and have comparable grammar.

Brendan Eich was hired by Netscape Communications in 1995 to create a scripting language for a web browser. It was prepared quickly by Eich. The official version of the new language used in the Netscape Navigator 2 beta version was dubbed LiveScript, but the first version of the language was called Mocha. JavaScript, a newly created scripting language, was given that

name in 1995 and utilised in the following Netscape Navigator 2 beta release. Many Java functionalities were adopted by LiveScript. This and, more importantly, the desire to capitalise on Java's rising popularity to give a new language a catchy name were the driving forces behind JavaScript's eventual name. Additionally, the language's server-side implementation was introduced.

JavaScript 1.0 was a hit and helped Netscape Navigator maintain its market-leading position. JavaScript 1.1 soon became available in Netscape Navigator 3.

Microsoft made the decision to include scripting technology into their browser at this time to compete with Netscape. They introduced Internet Explorer 3 in 1996 along with JScript, their own JavaScript implementation.

Since then, JavaScript has existed in two different versions. This caused compatibility problems. It was determined to standardise the language because of business demands. To "standardise the syntax and semantics of a general purpose, cross-platform, vendor-neutral scripting language," Netscape submitted JavaScript to Ecma International. The ECMA-262 standard for the language specification known as ECMAScript was issued in 1997. This standard might be used by other browser manufacturers to plan their implementation.

New specifications for JavaScript were periodically issued throughout the ensuing years as work progressed. Ecma International and Mozilla Foundation, which replaced Netscape Navigator, oversee creating JavaScript.[5]

### 6.2.2 CSS

One of the three foundational web technologies (the other two being HTML and JavaScript) is CSS. The hints are basically in the words "cascading" and "style," with cascading representing the way that one style may cascade from one to another. CSS stands for Cascading Style Sheets.

CSS allows for the usage of many styles inside a single HTML page, which is just one of its many advantages.

CSS is used to specify how HTML code will appear on a website. While CSS modifies how a web page will look, HTML (Hypertext Markup Language) is used to construct content, including written text.

Therefore, a developer may decide to have a page with tabs running down the top of the page or along the side, depending on the data they wish to present.

Another developer may decide to completely redesign an existing website or employ header and subheading styles to make the content pop off the page.

Perhaps explaining what a page would look like if it didn't utilise CSS is the greatest approach to illustrate what CSS accomplishes.

Without CSS, websites are uninspired and boring. Reading is challenging because the words glide over the entire page. However, it is precisely how web pages were before CSS.

The development of CSS is somewhat to blame for how the web appears and functions today. It is also a language that is always changing, far from being produced and therefore finished.

### **Why do we need CSS?**

First, adopting CSS guarantees the consistency of your web pages. Imagine a website with hundreds of pages. Then, picture having to input the code defining header sizes, layout, and other display data while blending it all with the text each time you wanted to create a new page. Additionally, think of being able to update just one page on a website with hundreds of others remaining the same thanks to CSS. While providing uniformity where it is required, CSS is flexible enough to let you make modifications to specific pages or parts as needed.

### **Using CSS allows a user to specify.**

- Fonts
- Color of text and links
- Use colors in the text's background.
- Where and how boxes within the content look and are placed.
- And...CSS also improves user accessibility, efficiency, flexibility and ensures browser compatibility.[9]

## Who invented CSS?

According to Wikipedia, Norwegian Hkon Wium Lie is primarily responsible for the invention of CSS. Lie set out to develop a universally accepted style sheet for the World Wide Web in 1994.

The Arena web browser served as Lie's first test site for CSS. Following the original design, Lie collaborated with Tim Berners-Lee and Robert Cailliau to produce the CSS1, CSS2, and RFC 2318 versions. In its first 10 years of existence (1994–2004), CSS, in all its standards, established itself as a web standard and had a significant impact on the appearance and use of the internet as we know it today. In 1999, CSS3 was released.

Lie is passionate about the subject of web standards. Since releasing CSS, he has urged influential technology companies like Microsoft and other browsers to adopt common web standards and he is still working on CSS's applications for online printing and screen pagination.

## What is the difference between CSS1, CSS2 and CSS3?

**CSS (1996)** allows the user to select font style and size and change the color of the text and background.

**CSS2 (1998)** has capabilities that allow the user to design page layout.

**CSS3 (1999)** allows the user to create presentations from documents and to select from a wider range of fonts including those from Google and Typecast. Uniquely, CSS3 allows the user to incorporate rounded borders and use multiple columns. CSS3 is easier to use (when compared to CSS2) because it has different modules.

## What about CSS4?

CSS is always being improved by the Working Group of W3C (w3.org), a group that discusses all technical elements of CSS's development and responds to questions from a public mailing list. It appears that the w3.org is using extensions to add additional, smaller components to the existing CSS versions rather than producing a whole new version named CSS4. However, it's important to note that many contend that CSS3 doesn't exist before we wonder where CSS4 is. The designations CSS 3 and CSS 4 are instead used to describe any improvements that were made after CSS2 (and are thus

formally referred to as CSS2.1). According to these criteria, CSS4 has occurred but it will never be known as CSS4.

Of fact, CSS's capabilities have no end, therefore they are always being improved. Regularly encouraging users to explore and experiment with CSS is the website CSS-Tricks, which among other things covers tales about the evolution of CSS. One narrative, for instance, describes the entertaining or practical things to attempt using the Universal (\*) selection. For instance, enhancing a WordPress site with border-boxes on web pages or transitions.[4][9]

### **6.2.3 HTML**

The preferred markup language for texts intended to be viewed in a web browser is HTML, or HyperText Markup Language. It frequently benefits from tools like Cascading Style Sheets (CSS) and programming languages like JavaScript.

HTML documents are downloaded from a web server or local storage by web browsers, who then turn them into multimedia web pages. HTML initially featured cues for a web page's look and semantically explains the structure of a web page.

#### **Development of HTML**

A system allowing CERN researchers to use and exchange papers was suggested and prototyped as ENQUIRE in 1980 by physicist Tim Berners-Lee, a contractor at CERN. Berners-Lee suggested an Internet-based hypertext system in a document he sent in 1989. In late 1990, Berners-Lee defined HTML and created the browser and server software. Robert Cailliau, a data systems engineer at CERN, and Berners-Lee worked together on a combined funding proposal that year, although CERN did not formally accept the initiative. An encyclopedia is the first item on Berners-Lee's list of "some of the many areas in which hypertext is used" in his own notes from 1990.

Web browsers translate and create text, pictures, and other content into audible or visual web pages using HTML, a markup language. Every piece of HTML markup has default properties that are set by the browser, but the

web page designer may also use CSS to change or improve these properties. The 1988 ISO technical report TR 9537 Techniques for using SGML, which describes the characteristics of early text formatting languages like that used by the RUNOFF command created in the early 1960s for the CTSS (Compatible Time-Sharing System) operating system, refers to many of the text elements. These formatting instructions were created by deriving them from the manual formatting commands used by typesetters. The SGML idea of generalized markup, on the other hand, is built on elements (nested annotated ranges with attributes), with independent structure and markup, as opposed to only print effects. CSS has helped HTML grow in this approach gradually.

Berners-Lee believed that HTML was an SGML application. The first proposal for an HTML specification, the "Hypertext Markup Language (HTML)" Internet Draught by Berners-Lee and Dan Connolly, which included an SGML Document type definition to define the syntax, was published in the middle of 1993 and served as the official definition of HTML by the Internet Engineering Task Force (IETF). The draught, which was noteworthy for acknowledging the NCSA Mosaic browser's unique tag for embedding in-line pictures and reflecting the IETF's policy of building standards on successful prototypes, expired after six months. In the same vein, Dave Raggett's rival Internet Draught, "HTML+ (Hypertext Markup Format)", from late 1993, advocated standardizing already-used elements like tables and fill-out forms.

Early in 1994, the IETF established an HTML Working Group following the expiration of the HTML and HTML+ draughts. This working group developed "HTML 2.0" in 1995, the first HTML specification meant to serve as a benchmark for subsequent implementations.

The IETF's efforts to further development were hampered by conflicting interests. The World Wide Web Consortium (W3C), including assistance from for-profit software providers, has been maintaining the HTML standards since 1996.[13] HTML was recognized as a global standard in 2000 (ISO/IEC 15445:2000). Late in 1999, HTML 4.01 was released, with errata appearing up until 2001. The Web Hypertext Application Technology Working Group (WHATWG) started working on HTML5 in 2004, and by the time it was finished and standardized on October 28, 2014, it was a joint deliverable with the W3C.[4]

### 6.3 Content of our page

The website contains two main pages: login page which we login using two different ways either using a phone number or using face ID, and homepage which includes services and LED page.

#### 6.3.1 Login Page

On the login page, the user can access the application by entering their username and password or by identifying themselves with a social media login. A login page is a web page that asks users to enter their login information (such as username and password) to access a specific website or website. A login page usually consists of a login form where users can enter their credentials and send them to the server for authentication. After the server has verified the login information, the user can access the secure area of the website or application. Login pages are typically used to protect sensitive information and prevent unauthorized access to websites or applications. They are used in many contexts including online banking and email services.

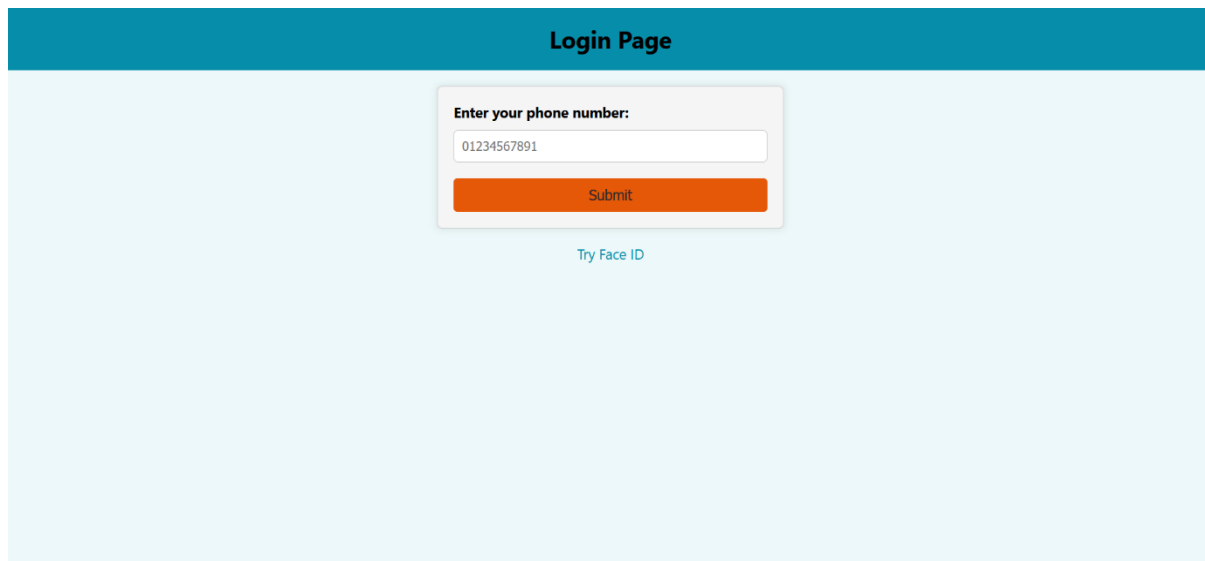
Usage scenarios related to the login page include:

A user navigates to an application and is presented with a login page to gain access to the application. There are two possible results:

- 1- Authentication is successful, and the user is directed to the application landing page.
- 2- Authentication fails and the user remains on the login page. If authentication fails, the screen should show an informational or error message about the failure. A user is automatically logged out due to inactivity. In this event, they will be returned to the login page, which will display an informational message explaining what happened. Once the user logs in again, they should be taken back to the page they were previously on before being timed out.



A user has forgotten their username and/or password. A link is available to begin the process to reset this information. Once the user clicks on one of these links, the contents of the login page are replaced with fields specific to recovering their username and/or password. There are several different ways the user could recover their password. This pattern does not dictate which methods an application should follow. Some options include: The user could provide their e-mail and be sent a temporary password or a link to reset their password. The user could answer a security question.



**Fig 6.1 showing Login Page**

This is the first page of the webpage which is a login page where the user enters their credentials to access the site. The credentials consist of typing a phone number containing 11 digits. When we log-in using the phone number, we use OTP (one-time password). One-time password (OTP) systems offer a way to access a network or service using a one-time-use only, unique password. The user can type in the phone number they associate with logging in into the site or use Face ID which is another easier way to log in.

### **6.3.2 Face ID Page**

Face ID is a biometric authentication technology developed by Apple. Face ID uses a True Depth camera system that projects and analyses over 30,000 invisible points to create a detailed 3D map of your face. The system then compares this card with cards stored on the device to authenticate the user's identity. Face ID uses a combination of floodlights and sensors to capture multiple images of your facial features. Face ID scans the contours of your

face. When you want to unlock your device, it immediately matches your mug against thousands of infrared dots and against your facial reference data to find a match.



**Fig 6.2 Face ID Page**

This is the page the user gets when they enter their Face ID. So, what the Face

ID does is identify a face through the camera lens and store it in the site's data.

So, whenever the user logs in every time, the camera will recognize them as an already active user of the site and identify them as the owner of the phone number they enter so it easier for the user to log in every time using just their face.

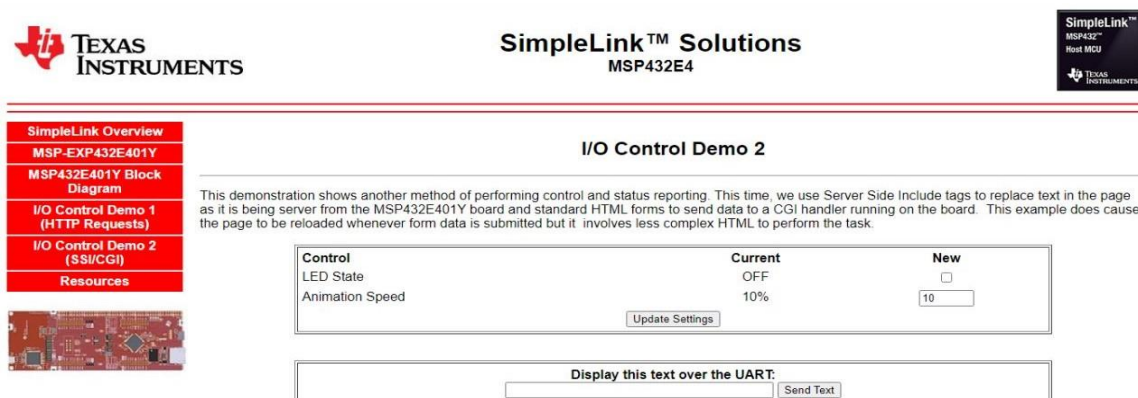
In the Face ID page, there is a “start camera” button so when the user clicks on this button, the site opens the camera and checks whether the user is someone the site recognizes or are they a first-time user. To ensure authentication, when the site checks the face of the user is the one, they recognize, they forward them to the home page. This measure is done to make sure the user and only this user has access to the home page and not anyone else can log in using their face.



**Fig 6.3 Showing Face ID Page while Camera is open.**

### 6.3.3 Home Page and its services

This is the home page the user gets when they log-in using their phone number or face ID. It has different tasks assigned in the forms of buttons that the user can click to access different pages. This page provides two services to the user that has access to it which include services like:



**Fig 6.4 Showing Home Page of our MSP432E4 Microcontroller**

### 6.3.4 LED

The user will have access to a page that has a button “Toggle LED” for toggling the led to power it on or off and it tells us whether the LED is powered on or off.

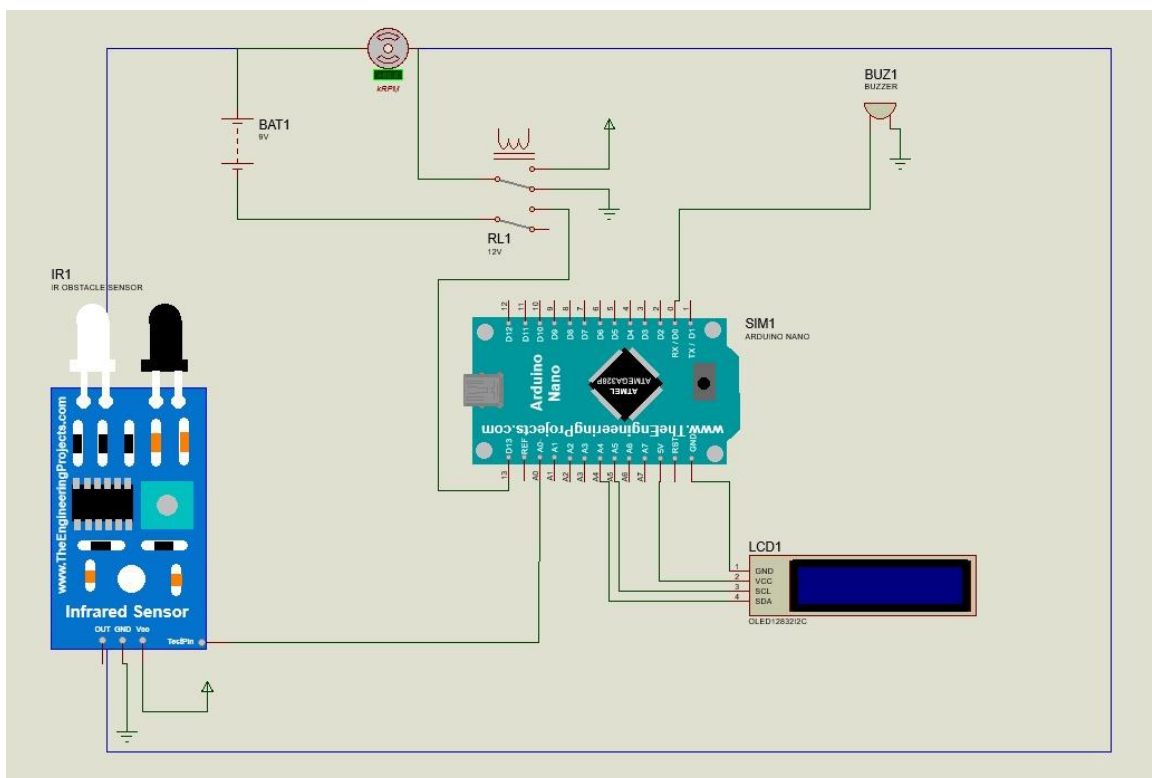


**Fig 6.5 Showing LED page that can turn LED on or off.**

# **Chapter 7: Conclusion and Future work**

The graduation project has successfully demonstrated the ability to design and implement a real-time operating system for a specific processor architecture (arm -m), and to develop a practical application that utilizes RTOS. This project has the potential to be useful in a variety of settings, such as in-home fire alert systems or can be used in an automotive application to manage and control the various subsystems such as engine control.

RTOS can also be used in other embedded systems such as medical devices, industrial automation systems, and aerospace applications.



**Picture 7.1 Showing our expectation to our first program to use with Squad RTOS.**

The development of a subsystem within our Squad RTOS has been a crucial step in enhancing the efficiency and safety of our operations. This subsystem is designed to showcase the impact of threads, particularly in situations where immediate action is

required, such as detecting a fire. When the Arduino Nano detects a fire, it instantly sends a signal to the MSP432, which triggers a series of actions to prevent the fire from escalating. The MSP432 then halts all low-priority threads and redirects the system's resources to the critical thread responsible for stopping the fire. This approach ensures that the most important task is prioritized and executed promptly, thereby minimizing the risk of damage or harm. With this subsystem, we have significantly improved our system's responsiveness and ability to handle emergencies.

**Future work:**

**Porting to other platforms:** Porting your RTOS to other platforms such as ARM Cortex-M4, Cortex-M7, or other microcontrollers can help increase its compatibility and usability.

**Real-time Debugging:** Developing real-time debugging tools such as system tracing and event logging can help diagnose and fix issues in real-time applications.

**Energy Efficiency:** Improving the energy efficiency of your RTOS can be beneficial in battery-powered devices such as IoT sensors and wearables. You can consider implementing power-saving features such as dynamic frequency scaling, sleep modes, and task suspension.

**References:**

- [1] Amos, Brian, Mistry, Rajan, and Gite, Shrikant. "Hands-On RTOS with Microcontrollers". Packt Publishing, 2020.
- [2] Axelson, Jan. "Serial Port Complete". Lakeview Research LLC, 2020.
- [3] Chaudhary, Bhaskar. "Tkinter GUI Application Development Blueprints". Packt Publishing, 2015.
- [4] Duckett, Jon. "HTML and CSS: Design and Build Websites". John Wiley & Sons, 2011.
- [5] Flanagan, David. "JavaScript: The Definitive Guide". O'Reilly Media, 1996.
- [6] Forouzan, Behrouz A. "TCP/IP Protocol Suite". McGraw-Hill Education, 2016.
  
- [7] Lin, Axel. "lwIP: The Definitive Guide". Packt Publishing, 2018.
- [8] Maruch, Aahz and Maruch, Stef. "Python For Dummies". Wiley, 2006.
- [9] Meyer, Eric A. "CSS: The Definitive Guide". O'Reilly Media, 2000.
- [10] Penmuchu, Chowdary Vencateswara. "Simple RTOS: Kernel Inside View for Beginner". BPB Publications, 2020.
- [11] Stallings, William. "Data and Computer Communications". Pearson, 2013.