PRACTICAL-6

AIM

Study and Introduction of Tiny OS in IOT development.

THEORY

- **TinyOS** is an embedded, component-based operating system and platform for low-power wireless devices, such as those used in wireless sensor networks (WSNs), smartdust, ubiquitous computing, personal area networks, building automation, and smart meters.

- It is written in the programming language nesC, as a set of cooperating tasks and processes. It began as a collaboration between the University of California, Berkeley, Intel Research, and Crossbow Technology, was released as free and open-source software under a BSD license, and has since grown into an international consortium, the TinyOS Alliance.

- TinyOS has been used in space, being implemented in ESTCube-1.

- TinyOS is an open-source, BSD-based operating system which uses the nesC programming language to control and manage wireless sensor networks (WSN). The sensor devices (called motes) in such networks are characterized by low power, limited memory and very small form factor.

Why Is TinyOS Useful for Wireless Sensor Networks?

- Low-power sensors, due to their limitations in scope, require efficient utilization of resources. TinyOS is essentially built on a “components-based architecture” to reduce code size to around 400 to 500 bytes and an “events-based design” which eliminates the need for even a command shell.

- The components-based architecture uses “nesC,” which is a C programming language designed for networking embedded systems. Each code snippet consists of simple functions placed within components and complex functions integrating all the components together.

- TinyOS also uses an “events-based design” whose objective is to put the CPU to rest when there are no pending tasks. An event can be something such as the triggering of an alert when the temperature of a thermostat rises or falls above a certain value. As soon as the event is over, the sensor motes can go to sleep.

- The need for a design like TinyOS is mandatory in applications such as smart transit and smart factories. Because of thousands of sensors, it is important to have a very small memory footprint to reduce power requirements.

- Where Is TinyOS Being Used?

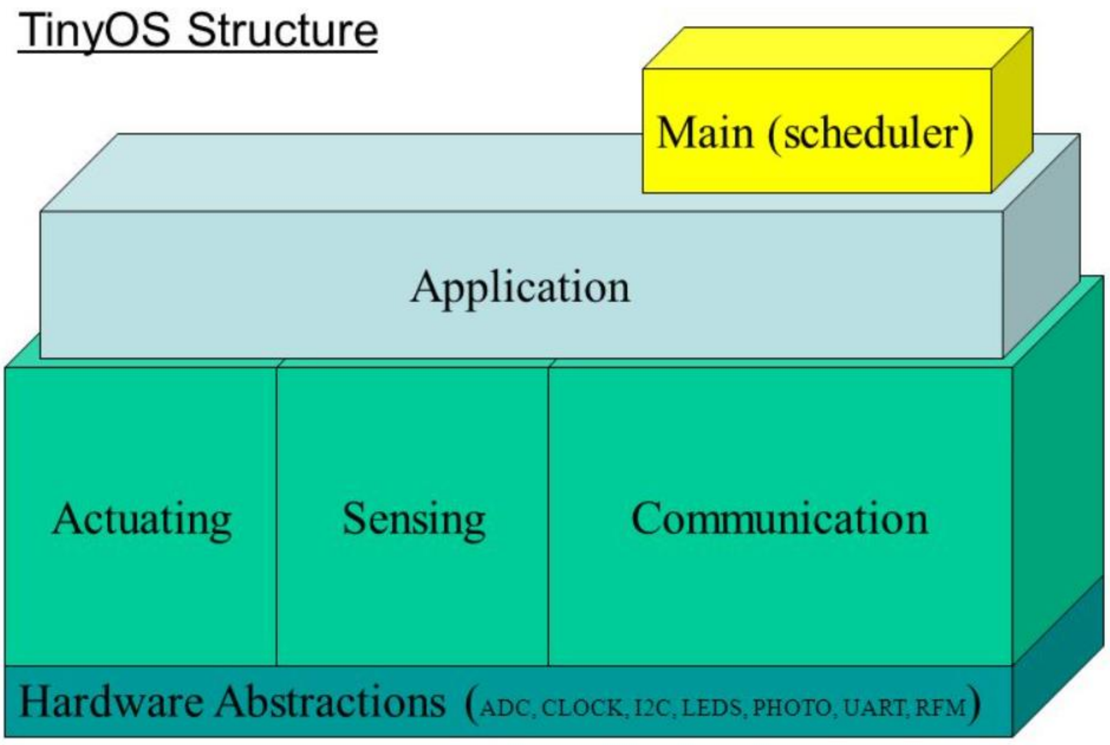
- At the moment, TinyOS has over 35,000 downloads. Its main applications lie in all kinds of devices which utilize wireless sensor networks.

- **Environmental monitoring:**
  - Since each TinyOS system can be embedded in a small sensor, they are useful in monitoring air pollution, forest fires, and natural disaster prevention

- **Smart vehicles:**
  - Smart vehicles are autonomous and can be understood as a network of sensors. These sensors communicate through low-power wireless area networks (LPWAN) which makes TinyOS a perfect fit.

- **Smart cities:**
  - TinyOS is a viable solution for the low-power sensor requirements of smart cities' utilities, power grids, Internet infrastructure and other applications.

- **Machine condition monitoring:**
  - Machine-to-machine (M2M) applications have many sensor interfaces. It is impossible to assign a complete computing environment to each sensor. TinyOS can perform security, power management and debugging of the sensors.



TinyOS models:

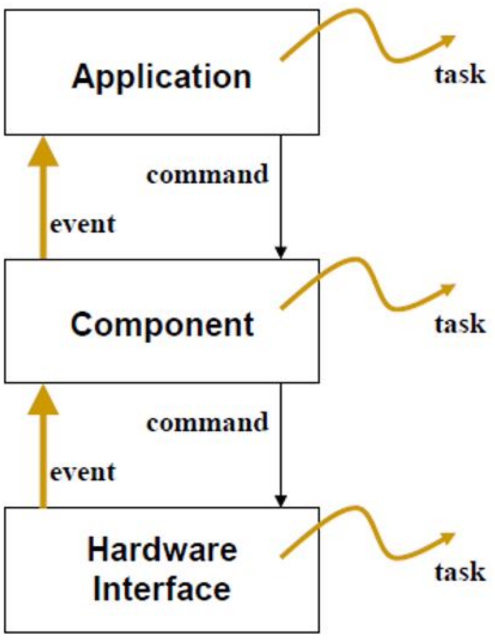
1. Data model:
    - **Static Memory Allocation**
        - No Heaps or any other dynamic structures used.
        - Memory requirements determined at compile time.
        - This increases the runtime efficiency.
    - **Global variables**
        - Allocated on per frame basis.
    - **Local Variables**
        - Saved on the stack
        - Defined in the function/method

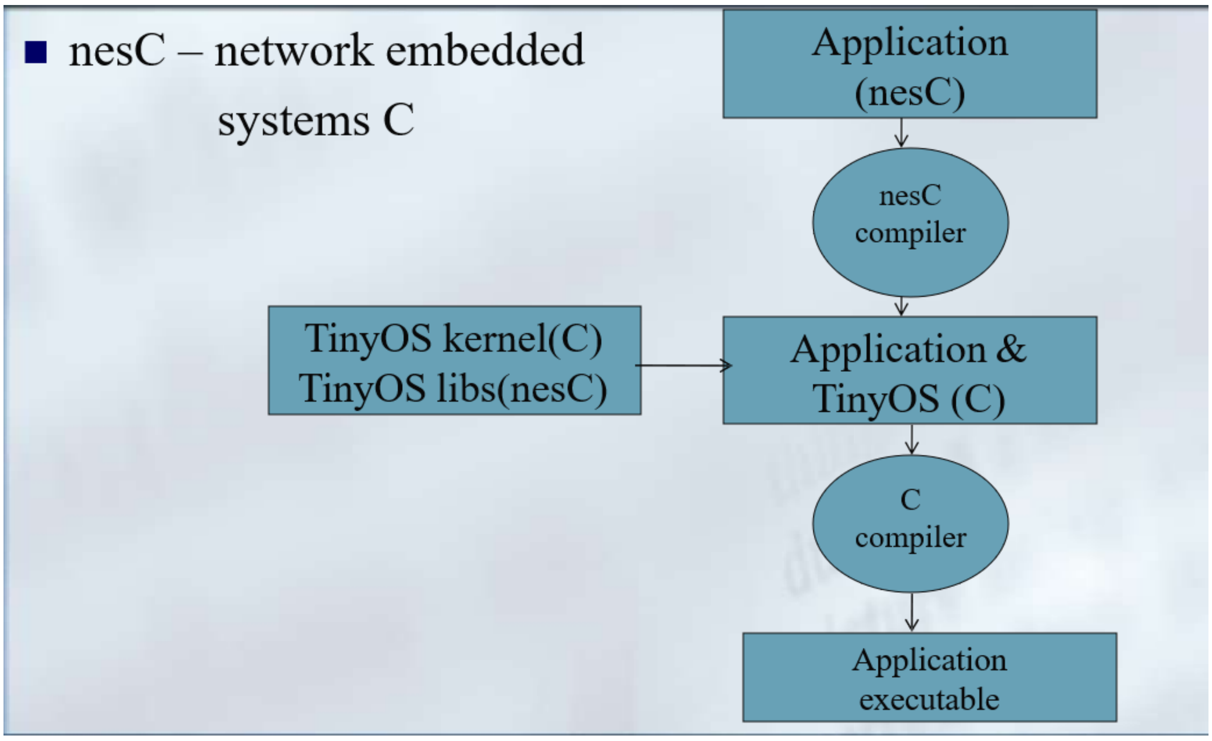
2. Thread model:
    - **Power-Aware Two-levels Scheduling**
        - Long running tasks and interrupt events
        - Sleep unless tasks in queue, wakeup on event
    - **Tasks**
        - Time-flexible, background jobs
        - Atomic with respect to other tasks
        - Can be preempted by events
    - **Events**
        - Time-critical, shorter duration
        - Last-in first-out semantic (no priority)
        - Can post tasks for deferred execution

3. Programming model:
    - **Separation construction/composition**
    - **Construction of Modules**
        - Modules implementation similar to C coding
        - Programs are built out of components
        - Each component specifies an interface
        - Interfaces are “hooks” for wiring components
    - **Composition of Configurations**
        - Components are statically wired together
        - Increases programming efficiency (code reuse) a runtime efficiency.

4. Component model:
    - Components should use and provide bidirectional interfaces.
    - Components should call and implement commands and signal and handle events.
    - Components must handle events of used interfaces and also provide interfaces that must implement commands.

TinyOS basic constructs:





■ nesC – network embedded systems C

- An extension to the C programming language, to embody the concepts and execution model
- of TinyOS.
- Filename extension .nc
- Static language

- No dynamic memory(malloc)
- No function pointers
- No heap
- Includes task FIFO scheduler.
- Designed to encourage code reuse.

CONCLUSION

In this practical, I learnt about Tiny OS.