Volume 3, Issue 8, August 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

An Introduction and Study to Tiny Operating System (T.O.S.)

Balram Yadav
Reader, CSE Dept. MITM Ujjain India

Vishwas Karhadkar
Lecturer, CSE Dept. MITS Ujjain India

Yodesh Kakde
Lecturer, CSE Dept. MITM Ujjain India

Abstract— this paper present, the study and introduction of tiny operating system. As the name implies tiny means small in size and we all know what an operating system is? In simple terms, OS provides an interface between user of computer and computer hardware; it provides lot of services to user directly or indirectly. TinyOS [6,10] means an operating system but small in size (i.e. in terms of small lines of code and also small set of services and features).this operating system is not used wide range of services it is specifically used for a particular service or some sort of event handling for example wireless sensor networks [1]. Tiny operating system now a day is a technology which has capability to change many of the information communication and technological aspects in the upcoming era. Important properties of TinyOS [6, 10] are Small Physical Size and low power consumption, con currency operations and parallessim.Design of Tiny-OS meets these challenges well and have become the platform of choice for sensor network research area.

Keywords— Event handling, low power consumption, concurrency model, sensor network.

I. INTRODUCTION

TinyOS [10] is a free and open source software, component-based operating system and targeting to wireless sensor networks (WSNs) [1]. Tiny OS is an embedded operating system written in the nesC languages a set of cooperating tasks and processes. The emergence of compact, low-power wireless communication and Networked sensors [1] is giving rise to entirely new kinds of embedded systems that are distributed and deployed in dynamic, constantly changing and adaptive control environments. These Networked Sensors are compact devices that can be used to sense light, heat, position, movement, etc from real environments and communicate information back to traditional computers. They also need to assist each other in collection of data and conveying them back to the centralized collection point. TinyOS [6] has an event based environment that is designed for use with embedded networked sensors [1]. It is design to have some important features like to support the concurrent operations required by networked nodes with minimal intervention with hardware. It uses the Message Communication model. TinyOS was developed by a group of four Computer Science Graduate Students at the University of California, Berkeley. The development of TinyOS [10] was supported by Defence Advanced Research Project Agency (DARPA), the National Science Foundation (NSF) and Intel Corporation [1]. A critical step towards achieving the vision behind wireless sensor networks are the design of software architecture that supports many features such as Tiny threading operating environment [10], small execution model and component model [2] are the important TinyOS features.The background and application requirements that motivated the development of TinyOS. It enumerates the characteristics associated with any typical Networked Sensor application. Tiny OS is the dominant software platform used for sensor networks, enabling hundreds of research results. It is used in numerous commercial products, such as Zolertia, Cisco's smart grid systems (formerly Arch Rock), and People Power Company.

TOSSIM is the simulator to simulate entire applications of TinyOS or you can implement code of TinyOS in nesC [8] programming language.

II. Tiny OS Features [1, 6, 10]:

The main features of Tiny OS that made them very useful are

- Concurrent model
- Sequential and event-based processing
- Memory model
- Single stack of global data
- Modularity Design
- Application is based on components
- Tiny OS is a "library"

• Minimal lines of code and low power consumption

Reduce in Size (Lines of Code) and minimal power consumption and the processing time, storage and interconnect capacity of the devices. Due to these constraints on resources, the operating system and applications have to use them efficiently.

• Concurrent Operation

Devices have to communicate information and data with little processing. Hence the system must handle multiple flows

of data concurrently and also perform processing and communication parallel.

• To support Parallelism and central Controller

The Number of nodes, hardware devices and resources, their capabilities and complexity of the interconnection are much lower for these Networked Sensors when compared to conventional systems. The sensors provide a primitive interface directly to the central controller unlike conventional systems that distribute concurrent processing over multiple levels of controllers.

• Diversity in Design and Usage

These devices are application specific, hardware is specific to the application and the variations in them are likely to be large. Hence these devices require an unusual degree of Software Modularity that must be efficient and specific to the appropriate task assign to them.

• Robust Operation

The devices, nodes and other resources will be numerous and they have to work concurrently and respond fractionally, to enhance the reliability of individual devices is essential and efficient fault tolerant approaches in needed.

III. Introduction to Language \ OS Co design

The **nesC (network embedded systems C) [8],** a programming language for networked embedded systems that represent a new design for application developers.

The implementation of Tiny OS and applications are all written in nesC **(network embedded systems C)[8]**, a new language which is based on components [2]. The nesC language is primarily intended for embedded systems such as sensor networks. nesC has a C-like (C Language) syntax, but supports the Tiny OS concurrency model, as well as other features needed to implement TinyOS.

Tiny OS applications are written in nesC, a language related with C and C language is optimized for the memory management and other features for sensor networks. Associated libraries and tools, such as the nesC compiler are written in C language. Tiny OS is based on nesC, Applications are based on components and code is encapsulated in components defined by the interfaces. Interfaces are bidirectional; they do not only define the commands that have to be implemented by the lower level that implements the interface. The nesC language allowed TinyOS to achieve near-optimal resource efficiency (minimization) and a surprisingly low bug rate (prevention). A severe challenge with TinyOS is that it has only one stack implementation for Task and all nodes accessing this stack in concurrent manner, this can problem us in real implementation. We all know the Race condition of OS of Process synchronization.

Race Condition: A race condition occurs when two or more threads or processes can access shared data and they try to change it at the same time. Because the thread or process scheduling algorithm can swap between threads at any time, we can't know the sequence in which the threads or processes will attempt to access the data (which is shared). Therefore, the result of the change in data is dependent on the thread scheduling algorithm, i.e. both threads are "racing" to access/change the data.

Example: RACE Condition

```
If (x==3) // checking the condition
{
y=x*4; // accessing the value
            // problem is that if another thread
            changed x in  between y will not
            be 12
}
```

nesC's language features allowed to write robust code that used very few hardware resources. Furthermore, nesC gave us the flexibility to discover novel programming abstractions that are not possible in C and greatly improve system development, such as static virtualization. Components are a significant improvement over basic C code. They provided clean, reusable interfaces, data privacy, and enabled many tools for checking and verifying

Components Specification: A nesC application code consists of one or more components coupled with each other to form an execute environment. A component uses interfaces; these interfaces are the only point of access to the component and are bi-directional in nature. An interface declares a set of functions (of C) called commands (nesC) that the interface provider must implement and another set of functions called events that the interface user must implement. For a component to use or call the commands in an interface first of all it hah to implement the events of that particular interface. A single component may use or provide multiple interfaces and multiple instances of the same interface.

Implementation[8]:

There are two types of components in nesC: **modules** and **configurations**.

Modules: implementing one or more interfaces.

Configurations are used to couple other components together, connected interfaces used by components.Every nesC application is described by a top-level configuration that coupled together the components inside.

IV. Programming Model [8]: Components, Commands, Events and Tasks

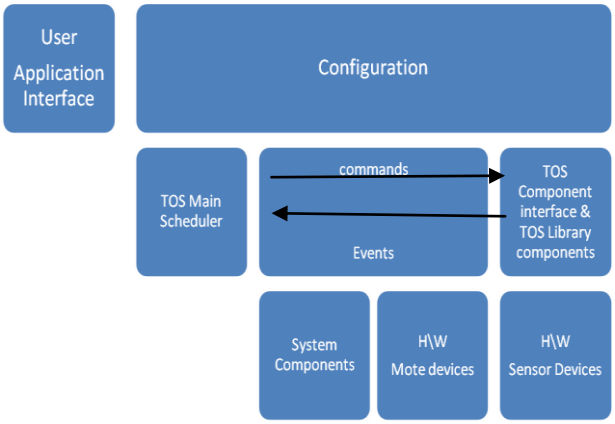

Fig1: TOS working model

As already mentioned, Tiny OS applications used by user and the operating system itself are composed of components. Components enter three types of elements: Commands, Events and Tasks. All three are basically normal C functions but they differ significantly in terms of who can call them and when they get called. **Commands** often are requests to a component to do something, for example to query a sensor or to start a computation. **Events** are mostly used to signal the completion of such a request. Components use an interface to implement all of the specified events and can use all of the commands provided by that Interface. An interface specifies which commands the user of the interface can call and which events the provider can signal. The connections between components are called configurations.

A **configuration [5]** defines for each interface of a component which other component uses the provided interfaces and which component provides the used interfaces. These connections are called wirings. Configurations are itself a component, which means that they can also provide and use interfaces. This makes it possible to build Tiny OS applications in a hierarchical manner where components on a higher level are made up of several components of a lower. NesC is related very much with c. It supports all of the concepts of Tiny OS, like components, interfaces, commands, events, tasks, and configurations etc nto a language like c. The biggest difference between C and nesC is how the function to be executed is selected. In C the function to be executed at a function call is selected by its name at run time. In nesC when a command or event should be executed the programmer explicitly selects with the wiring in configurations which component's implementation [5] of the command or event should be used. But"normal" C functions can still be used in NesC and to differentiate between them and commands, events and tasks we used some reserve keywords for invoking each of the non-C function types(components, interfaces, commands, events, tasks, configurations etc.). We can also use C libraries and C preprocessor directives to implement some thing.

A Component [5] has four parts: a set of Command Handlers (to handle commands interpretation), a set of Event Handlers (for event handling), a fixed-size frame and a queue of simple tasks. Each component declares the commands it uses and events it signals. Each Component is described by its interface and its internal implementation. An interface contains commands and events. These declarations are used to compose the components and this composition creates layers of components that are application specific. The higher-level components [5] issue commands to lower-level components while the lower ones signal events to the higher-level components. Hence we can think of the component to have an upper interface, which names the commands it implements and the events it signals a lower interface which names the commands it uses and events it handles. Commands are requests [5] made to lower level components. A command contains request parameters and encapsulates it into its frame and conditionally posts a task for a execution at some specified later time. It also provides feedback to its caller (from a higher level component) by returning status of success or failure. Event Handlers are invoked to deal with Hardware events. The bottom most level components have handlers routine connected directly to hardware. Event Handler deposit information in its frame, post tasks, signal higher-level events or call lower level commands. Tasks perform the work and are atomic (all or none mechanism) with respect to other tasks. They run to completion and can call lower commands, signal higher-level events and schedule other tasks within the same Component. The run-to completion property helps to allocate a single stack to the currently executing task and this conserves space. Tasks also allow concurrency within each component as they execute asynchronously. They must never block to avoid delaying progress in other components. Hence we can look at these tasks as blocks of computation.

The Task scheduler [2] is a simple FIFO scheduler [2] that has a bounded size (limited) scheduling data Structure. It is power sensitive and puts the processor to sleep when there is no task in or when queue is empty, but leaves the peripherals in operating mode to wake up the system in case of any new hardware event comes.

V. Conclusions

The objective of this paper is to study TonyOS , it's features , services and model. This paper also introduces about the nesC programming language , used for implementation of TinyOS.it is just an attempt to introduce the basics of TinyOS, features, some implementation details.nesC language provides the features that are necessity of TinyOS requirement. A clear view is provided to differentiate the components, interface, events and task. This research topic is very vast and currently came into the picture. It is open for research and there are many fields related to this topic which demands more efficient solution like memory management, task scheduling, event handling and interrupt handling etc.

References

[1] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. "System architecture directions for networked sensors". In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, November 2000.

[2] Sharan Raman," TinyOS – An Operating System for Tiny Embedded Networked Sensors" Paper Presentation for Advanced Operating Systems Course, Spring 2002

[3] Jitendra Joshi,Rama Bhardwaj,Priyanka Sharma,Chetan Singh,Manjari Kumari "Tiny OS –software structural design and implementation for wireless sensor networks" Internation Journal of Advanced research in computer science and software engineering ,volume 3,Issue 1, January 2013

[4] Levis Philip "Experiences from a Decade of Tiny OS Development"

[5] Book : Levis Philip "Tiny OS Programming"

[6] Arvind Easwaran " Tutorial : Tiny OS"

[7] Eric Trumpler,Richard Han"A Systematic Framework for Evolving TinyOS"

[8] David Gay,Philip Levis,Robert ven Behren,Matt welsh,Eric Brewer,David culler "The nesC Language : A holistic approach to networked embedded systems"

[9] Matthias Niederhausen "Feature model of Tiny OS Makerules "November 2, 2007.

[10] Rev. A "Tiny OS getting started guide " October 2003 Document 7430-0022-03

.

