IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727
PP 35-41
www.iosrjournals.org

A COMPARATIVE STUDY ON NETWORK OPERATING SYSTEMS

Athira Joshy[1], Saranya Ramadas[2]
[1](*Department of Computer Application, Sree Narayana Gurukulam College Of Engineering Kadayiruppu, India)*
[2](*Department of Computer Application, Sree Narayana Gurukulam College Of Engineering Kadayiruppu, India)*

Abstract: *Operating System gives life to a computer. Now a days networks are inevitable in human life. Different Networks demand different operating systems for their effective functioning. In this background we have made a comparative study of network operating systems. Their features, advantages and limitations are described in this paper.*
Keywords*: TinyOS, Mantis OS, SOS.*

I. INTRODUCTION

An Operating system (OS) is a collection of software modules that facilitate an effective interface between the computer user and the system resources. The computer resources comprise CPU, Main Memory, I/O Devices, Secondary Storage Devices, Files and Network etc. An Operating System is a software that manages the computer hardware and provide common services for computer programs. Operating system is acts as a platform for application programs to run.

Network Operating System is an infrastructure that ensure the reliable distribution of processes, files systems, networking components, networking protocols, and other associated components in order to produce a system which is reliable and secure, and which operates within a required specification. In the absence of an operating system a computer is merely a box with circuits. In case of an Network Operating System, a network is nothing more than a collection of computer devices connected together. A network operating system enhances the information flow and communication across different networks.

While comparing a network operating system with workstation operating system, NOS is primarily designed to provide services to remote clients. Another key difference is that network operating systems are designed and optimized to support concurrent multi-user environment. That means that with the network operating system we can have lots of users all concurrently using resources on our server.

Workstation operating systems do not provide network services. They can support multiple users, but have a limit on users who can work concurrently, designed to run on lower-end hardware. Network operating systems provide services to remote network clients. NOS is highly scalable with multi user support. They are designed to run on high-end hardware, and rely on redundancy. With redundancy the server can continue to work even if something on it fails which makes the NOS robust. Server OS usually provides and requires high security implementations.

II. STUDY OF DIFFERENT OPERATING SYSTEMS

2.1 TINY OS

2.1.1 PAPER 1: Smart Irrigation System for Outdoor Environment using Tiny OS[1]

TinyOS is a free and open source operating system for Wireless Sensor Network. In this paper the application of Tiny OS in Smart Irrigation for Outdoor Environment is illustrated.

TinyOs is a component based operating system designed only for embedded system application. TinyOS is introduced in collaboration as between the University of California and Intel Research. It is an embedded operating system in the nesC programming language. TinyOS support various controller families and radio boards. TinyOs is specially used for nesC programming language. The language learning is very easy

because some of the existing code. Because TinyOS has millions of users this code is very efficient and very easy to bug. TinyOS was developed primarily for use with networks of small sized sensor devices.

2.1.2 PAPER 2:Comparison of Operating Systems TinyOS and Contiki[2]

The paper compares TinyOS and Contiki, widely used and known sensor node operating systems. Their comparison is done based on their different requirements.

TinyOS was developed at the University of California in Berkeley and is now maintained as an open source project by a community of several thousand developers and users lead by the TinyOS Alliance. Event driven programming model is used in TinyOS and concurrency is implemented through non-preemptive tasks. TinyOS programs are component based and are written in the NesC language. It is optimized for limited memory in sensors.

- *Programming Model*

As mentioned above, TinyOS user applications and the operating system are composed of three types of components. Commands, Events and Tasks. Commands are requests to a component to do something, for example to start a computation. Events usually signify the completion of such a request. Tasks are not executed immediately, when a task is posted, the currently running program will continue its execution and the posted task will later be executed by the scheduler.

- *Execution Model*

The execution model of TinyOS is split-phase which means request and response of an operation after completion is decoupled. Hardware and software modules follow split-phase execution model, which is represented in the programming model: Both are components that can handle commands and at a later time signal events after the completion of the operation. Concurrency in TinyOS is achieved with tasks. Tasks are basically functions that can be posted by other tasks or in- terrupt handlers. They don't get executed immediately but instead will later be executed by the scheduler. The TinyOS scheduler executes one task after another and so tasks can never preempt each other. Each task runs un- til completion and the next task is started after that. This makes it possible that all tasks can use the same stack, which saves memory because not every task needs its own desig- nated stack like in a multi-threaded approach.

TinyOS was built with the goal of minimal resource consumption since wireless sensor nodes are generally very constrained in regards to processing speed, program memory, RAM and power consumption. The core of the TinyOStool chain is the NesC compiler. Current implementations of the NesC compiler take all NesC files, including the TinyOS operating system, that belong to a program and generate a single C file.

2.1.3 PAPER 3: Operating Systems for Wireless Sensor Networks: A Survey[3]

This paper presents a survey on the current state-of-the-art in Wireless Sensor Network (WSN) Operating Systems (OSs).

TinyOS is an open source, flexible, component based, and application-specific operating system designed for sensor networks. TinyOS can support concurrent programs with very low memory requirements. The OS has a footprint that fits in 400 bytes. The TinyOS component library includes network protocols, distributed services, sensor drivers, and data acquisition tools. The following subsections survey the TinyOS design in more detail.

- *Architecture*

TinyOS falls under the monolithic architecture class. TinyOS uses the component model and, according to the requirements of an application, different components are glued together with the Scheduler to compose a static image that runs on the mote. A component is an independent computational entity that exposes one or more interfaces. Components have three computational abstractions: commands, events, and tasks. Mechanisms for inter-component communication are commands and events. Tasks are used to express intra-component concurrency. A command is a request to perform some service, while the event signals the completion of the service. TinyOS provides a single shared stack and there is no separation between kernel space and user space.

- *Communication Protocol Support*

Earlier versions of TinyOS provide two multi-hop protocols: dissemination and TYMO. The dissemination protocol reliably delivers data to every node in the network. This protocol enables administrators to reconfigure queries and to reprogram a network. The dissemination protocol provides two interfaces: *DisseminationValue*and *DisseminationUpdate*. A producer calls *DisseminationUpdate*. The command *DisseminationUpdate.change()* should be called each time the producers wants to disseminate a new value. On the other hand, the *DisseminationValue*interface is provided for the consumer. The event *DisseminationValue.changed()* is signaled each time the dissemination value is changed. TYMO is the implementation of the DYMO protocol, a routing protocol for mobile *ad hoc* networks. In TYMO, packet formats have changed and it has been implemented on top of the active messaging stack.

Lin et al. have presented DIP, a new dissemination protocol for sensor networks. DIP is a data discovery and dissemination protocol that scales to hundreds of values. TinyOS version 2.1.1 now also provides support for 6lowpan , an IPv6 networking layer within a TinyOS network.

At the MAC layer, TinyOS provides an implementation of the following protocols: a single hop TDMA protocol, a TDMA/CSMA hybrid protocol which implements Z-MAC's slot stealing optimization, B-MAC, and an optional implementation of an IEEE 802.15.4 complaint MAC.

- *Support for Real-Time Applications*

TinyOS does not provide any explicit support for real-time applications. As we already discussed in the scheduling section above, tasks in TinyOS observe run to completion semantics in a FIFO manner, hence in its original form, TinyOS is not a good choice for sensor networks that are being deployed to monitor real-time phenomena. An effort has been made to implement an Earliest Deadline First (EDF) process scheduling algorithm and it has been made available in newer versions of TinyOS. However, it has been shown that the EDF algorithm cannot produce a feasible schedule when tasks content for resources. In the nutshell, TinyOS is not a strong choice for real-time applications.

TinyOS does not provide any specific MAC, network, or transport layers protocol implementations that support Quality of Service requirements of real-time multimedia streams. At the MAC layer, TinyOS supports TDMA, which can be fine-tuned depending upon the requirements of an application to support multimedia traffic streams.

- *Additional Features*

In this section, we discuss some additional features provided by TinyOS.

- File System

TinyOS provides a single level file system. The rationale behind providing a single level file system is the assumption that only a single application runs on the node at any given point in time. As node memory is scarce, having a single level file system is therefore sufficient.

- Database Support

The purpose of sensor nodes is to sense, perform computations, store and transmit data, therefore TinyOS provides database support in the form of TinyDB. Further details on TinyDB can be found in .

- Security Support

Communication security in wireless broadcast medium is always required. TinyOS provides its communication security solution in the form TinySec .

- Simulation Support

TinyOS provides simulation support in the form of TOSSIM . The simulation code is written in NesC and consequently can also be deployed to actual motes.

- Language Support

TinyOS supports application development in the NesC programming language. NesC is a dialect of the C language.

- Supported Platforms

TinyOS supports the following sensing platforms: Mica , Mica2 , Micaz , Telos , Tmote and a few others.

- Documentation Support

TinyOS is a well-documented OS and extensive documentation can be found on the TinyOS home page at http://www.tinyos.net.

2.1.4 PAPER 4: An Operating System for Tiny Embedded Networked Sensors[4]

This paper is viewed as An Operating System for Tiny Embedded Networked Sensors. This paper throws light on Tiny OS Design which is described here.

TinyOS is an event based operating environment that is designed for use with embedded networked sensors .It is designed to support the concurrency intensive operations required by networked sensors with minimal hardware requirements. It uses the Active Message Communication model for building non-blocking applications and higher Networking capabilities like Multihop ad hoc routing. TinyOS was developed by a group of four Computer Science Graduate Students at the University of California, Berkeley. The development of TinyOS was supported by Defense Advanced Research Project Agency (DARPA), the National Science Foundation (NSF) and Intel Corporation.

The emergence of compact, low-power wireless communication and Networked sensors 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, chemical presence 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.

These Embedded Sensors are characterized to be agile, self-organizing, critically resource constrained and communication centric. Their application space is huge including all monitoring applications in a context aware situation, situation monitoring of life science experiments, disaster management and others. There are two design issues associated with this scenario: these devices are Concurrency Intensive where several flows of data must be handled simultaneously and the system must provide Efficient Modularity, which means that hardware specific and application specific components must coalesce together with little processing and storage overhead.

There are bursts of activity where data and events stream in from the sensors and the Network and periods of passive listening to significant events. During the bursts a mix of real time actions and long-scale processing/computation must be performed and in the remaining majority of time, the device shuts to a very low power state and monitors for changes in the system state.

- *TinyOS Design*

TinyOS uses an Event model so that high levels of concurrency can be handled in a very small amount of space unlike the stack based threaded approach that uses too much stack space and also has a high context switch time. Since Power is a precious resource, CPU resources must be utilized efficiently. The event-based approach handles tasks associated with events rapidly without allowing blocking or polling. Unused CPU cycles are spent in sleep state as opposed to actively looking for events. TinyOS was developed in C.

‰Components, Commands, Events and Tasks

TinyOS is divided into a collection of Software Components. A TinyOS application consists of a scheduler and a graph of components describing their interaction. A Component has four parts: a set of Command Handlers, a set of Event Handlers, an encapsulated fixed-size frame and a bundle of simple tasks. Each component declares the commands it uses and events it signals. The fixed sized frames are statically allocated which helps to know the memory requirements of a component at compile time. The frame is an internal storage space that contains the state of the component and is used by the events, commands and tasks.

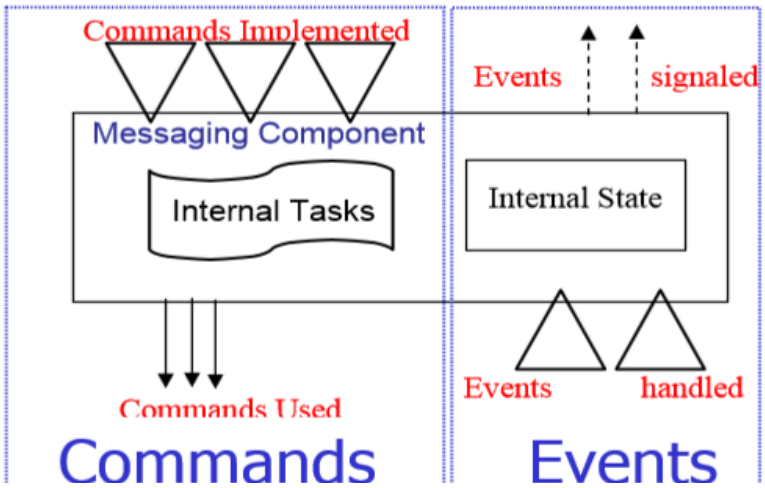

Figure 1: Structure of a Component in TinyOS

2.2 MANTIS OS

2.2.1 PAPER 1:Mantis Operating System Summary[5]

Mantis is a multimodal system for wireless sensors. It is written in c. MOS is implemented in RAM footprint that consumes only 500 byte of memory. It has a power efficient scheduler which puts the microcontroller into sleep when the sleep function is called. The features of MOS includes flexibility, platform support and ability to test on pc's PDA's and other micro sensor platform. It also support remote login and dynamic reprogramming. The other advanced features include multimodal prototyping, dynamic reprogramming. The things which are to be improved in MOS are its low power consumption reliability in code updated over network optimizing the size and security and authenticity of updates.

2.2.2 PAPER 2: MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms[6]

One of the attracting feature of MANTIS OS is its energy- efficiency. MOS is a lightweight operating system for Multimodal Networks. deployment and management of sensor networks is made easy through the tools available in Mantis OS. It provides a new multithreaded cross-platform embedded operating system for wireless sensor networks. A sensor networks does many complex tasks such as compression, aggregation and signal processing, pre-emptive multithreading in the MANTIS sensor OS (MOS) enables micro sensor nodes to natively interleave complex tasks with time-sensitive tasks, thereby mitigating the bounded buffer producer-consumer problem. Most of the complex tasks that comes along with a wireless sensor network is well managed and maintained through MOS.

MANTIS is a convenient platform for WSN applications. MANTIS provides a comprehensive set of System APIs for I/O and system interaction. The design of the MANTIS network stack is focused on efficient use of limited memory, flexibility, and convenience. The stack is implemented as one or more user-level threads. MANTIS adopts the traditional logical/physical partitioning with respect to device driver design for the hardware. The energy efficiency in Mantis OS is implemented by a sleep function. Its power-efficient scheduler recognizes when all threads are sleeping and then sleeps the microcontroller for a duration deduced from each thread's sleep time.

The MANTIS, the multithreaded OS widens the path of sensor systems to support increasingly complex tasks and keeping up the resource constraints of energy and memory of sensor networks. The advantage of time shared multithreading is that a single segment of application code cannot block the execution of other tasks. This is important in sensor systems, since blocking certain time-critical tasks from executing, such as network packet processing, can result in overflow of network buffers when tasks are sufficiently long-lived and a sensor node's RAM buffers are sufficiently small.

Sensor networks impose additional unique demands on the design of operating systems beyond resource constraints. Sensor net- working application developers need to be able to prototype and test applications prior to distribution and physical deployment in the field. Also, during deployment, in-situ sensor nodes need to be capable of being both dynamically reprogrammed and remotely debugged. In the next sections, MANTIS identifies and implements each of these three key advanced features for expert users of general-purpose sensor systems.

2.3 SOS - Dynamic operating system for sensor networks

2.3.1 PAPER 1: SOS - Dynamic operating system for sensor networks [7]

This paper describes about SOS which is the Dynamic operating system for sensor networks. The information gathered on SOS is described below.

SOS is a Dynamic operating system for sensor networks. It has Kernel and dynamically loadable modules and ported to Mica2, MicaZ, XYZ and Telos. Also it provide convenient, yet compact, kernel interface. In SOS safety features achieved through run-time checks. It include Type safe linkage and Memory overflow checks. Performance of SOS is no worse than TinyOS for real world applications.

- *SOS Application*

  - Ragobot – Mobile Sensor Node Software
  - All modules are dynamically loadable
  - Install new robot behaviours by updating navigation module
  - Future ragobot versions will support hot-swap of peripherals
  - SOS provides automatic driver updates.

SOS has contributions in areas such as Framework for binary modular re-programming which include dynamic linking, Message passing, and Dynamic Memory. Next one is Inexpensive safety mechanisms for an embedded OS which includes Type safe linking, Monitored memory allocation, Garbage collecting scheduler end error stub and Watchdog mechanism. And the last one is General purpose OS semantics on sensor nodes.

SOS is programmed entirely in C. It is Co-operatively scheduled system as well as event driven programming model but this system provides no memory protection. The designing safety features of SOS include dynamically evolving system, Goals ie, ensure system integrity and graceful recovery from failures and Design which have minimal set of run-time checks, designed for low resource utilization and does not cover all failure modes. The message safety features of SOS are high priority messaging and watchdog support.

Kernel services are available as system calls in SOS. System Jump table redirects system calls to handlers. Update kernel independent of modules. SOS provide dynamic memory allocation that is need to allocate module state at run-time. Garbage collection on failed message delivery is there.

We can summarize from this paper that SOS enables dynamic binary modular upgrades. Its design choices minimize resource utilization. SOS has run-time checks for safe code execution and it is ported to AVR, ARM and TI MSP.

Future Work

- New models for application development
  - Independent re-usable loadable binary modules
- Hierarchy of re-configuration
  - Maté VM ported to SOS - Extensible virtual machines
  - Upgrade SOS kernel using TinyOS whole image technique
- Staged checkers
  - Combination of static and run-time checks for code safety
- FLASH wear and tear management using SOS

III. COMPARISON IN A NUTSHELL

Sl No	Name of OS	Advantages	Disadvantages	Mostly used in

1	TinyOS	• Uses single stack. • Power efficiency and management. • Events/commands implemented as function call	• Absence of preemptive real time scheduling • Not flexible • Virtual memory not available	• Wireless sensors • Mote applications
2	Mantis	• Easy to learn and use • Flexible • Multimodal	• Low power management • Security of updates • Reliability in updation of code	• weather surveys biomedical research • embedded interfaces • wireless networking research • artistic works
3	Sos	• Dynamic OS • Convenient and compact kernel interface • Inexpensive safety mechanism • Event driven	• No memory protection	• Ragobot - Mobile Sensor Node Software • Building Automation • Dynamically installing new behaviour modules on ragobot • Remote operation and management of the sensor network infrastructure

IV. CONCLUSION

The network operating systems are very important in today's emerging world-whether it is wired or wireless. In this work we have compared three different operating systems which are in wide use today-Tiny OS, Mantis and SOS. All their important aspects are covered in this survey and comparative study is represented in a tabular form in nutshell. The work will be helpful to the researches working in this area. There are a few more operating systems which will definitely bring more services and performance coverage's in terms of new communication terminologies.

References

Journal Papers:

[1] P. Alagupandi / M.E. Student Electrical and Electronics Engg Department Anna University R. Ramesh / Associate Professor Electrical and Electronics Engg Department CEG, Anna University. S.Gayathri/ Teaching Fellow Electrical and Electronics Engg Department CEG, Anna University

[2] Tobias Reusing Supervisor: Christoph Söllner Seminar: sensor nodes - operation, networks & applications SS2012 Chair Network Architectures and Services, Department of Operating Systems and System Architectures Faculty of computer science, Technical University Munich

[3] Muhammed Omer Farooq and Thomas Kunz Department of Systems and Computer Engineering, Carleton University Ottawa, Canada

[4] Sharan Raman Paper Presentation for Advanced Operating Systems Course, Spring 2002

[5] Mantis Operating System Summary by Catherine Green

[6] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, Richard Han