Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 7, No. 3, Sep 2019, pp. 577~585
ISSN: 2089-3272, DOI: 10.11591/ijeei.v7i3.987

A Review on Reliability, Security and Memory Management of Numerous Operating Systems

Mohammad Marufuzzaman[1], Shafin Al Karim[2], Md Saifur Rahman[3], Noor Mohammad Zahid[4], Lariyah Mohd Sidek[5]
[1,5]Sustainable Technology and Environment Group, Institute of Energy Infrastructure, Universiti Tenaga Nasional, 43000, Kajang, Selangor,Malaysia
[2-4]American International University, Dhaka, Bangladesh.

Article Info

Article history:

Received Jan 25, 2019
Revised May 1, 2019
Accepted Jun 26, 2019

Keyword:

IoT
Scalabilitye
Energy efficiency
Operating system

ABSTRACT

With the improvement of technology and the growing needs of computer systems, it is needed to ensure that operating systems are able to provide the required functionalities. To provide these functionality operating systems are designed to maintain some design factors such as scalability, security, reliability, performance, memory management, energy efficiency. However, none of these factors can be achieved directly without facing any challenges. This research studied several design issues that are connected to each other in terms of providing an effective result. Therefore, this review article tried to reveal the major issues, which are independently more complex to solve at once. Finally, this research provides a guideline to overcome the challenges for future researchers by studying many research articles based on these design issues.

Corresponding Author:

Mohammad Marufuzzaman,
Sustainable Technology and Environment Group, Institute of Energy Infrastructure,
Universiti Tenaga Nasional,
43000, Kajang, Selangor,Malaysia

Email: marufsust@gmail.com

1. INTRODUCTION

The operating system (OS) is the brain of the computer that controls everything happening between the consumer and the hardware. In the modern era, the OS is required in any smart environment [1-3]. By definition, the operating system is called the communication bridge between the hardware and the user of a computer [4-5]. The operating system is formed on several programming instructions that are delivered to the hardware. The core programming instruction of the operating system is called the kernel. The computer hardware refers to central processing units, circuit board, monitor, keyboard, mouse and other disk drivers. There are several operating systems such as desktop operating systems, mobile operating systems, server-based operating systems and embedded operating systems. Few of the popular operating system is Windows OS, Linux/UNIX OS, Mac OS, Windows Server OS and Linux Server OS.

Several types of research are ongoing regarding different types of operating system to improve the design processes [6-7]. A significant amount of design factors needs to be considered while creating an operating system. Few of the design factors are openness, scalability, security, failure handling, concurrency, transparency, quality of service, reliability, performance, memory management, energy efficiency [8-10]. Among these factors, openness refers to the extension of the system where if new resources are introduced the system can cope up with it. Scalability is similar to openness; however, it mainly focuses on accommodating the growing amount of works or processes in the system rather than handling new resources. Performance and reliability are inversely proportional in terms of maintaining a good design of an operating system. If a system performs, well it can be assumed that several reliable factors are compromised in terms of achieving better

performance. However, security is one of the major factors in operating system designing that can never be compromised. With the growing amount of workloads and data transferring between devices, it has to be ensured that the data are secured from outside interruption. This security needs to be ensured by the operating system. Additionally, with the improvement of the internet of things (IoT), several operating systems have been developed. One of the major concerns of these operating systems is memory management and energy efficiency.

Scalability is one of the major design factors as the improvement of technology and devices, multicore systems are becoming more popular and effective in terms of processing a good amount of data. However, with the growing amount of multicore systems it must have to be ensured that the current operating systems are able to accommodate this growth.

Apart from scalability, security and reliability of an operating system are the two major design factors. An insecure system is always obscure to people. On the other hand, authorization and authentication are the two most desired aspects of an operating system. Similarly, one of the main aims of making a distributed system lies on the reliability issues. However, some features like security, reliability are some of the important factors of any operating systems, which will be focused in this review paper.

IoT enabled devices are becoming popular in the modern era as the concept of the smart city, smart home, the smart planet becomes a demand of time. In IoT, there are three types of communication: man-to-man, man-to-device and device-to-device communication [11]. For establishing the device-to-device communication, operating systems are needed, which provides instructions for the devices on communication method with each other. Recently different types of operating systems are designed to run the IoT devices. The operating systems for these devices need to be light-weighted due to the low-end devices. In addition, memory management and energy efficiency need to be focused while designing an operating system for these devices. Memory management provides an abstraction of programming taking care of things such as caching, memory allocation, memory de-allocation, virtual memory, logical-physical address mapping and memory protection at the back-end. Static and dynamic are the two types of memory management. Static memory allocation is simple and fixed, whereas dynamic memory allocation is run-time memory allocation. Moreover, as the devices are small, energy efficient also plays a major role while designing an operating system for Youth as It devices consume energy during sensing, data processing and data transfer.Therefore, a comprehensive review is required.

In this research, overcoming the design issues in different operating systems will be discussed. Four major design areas of OS are covered. The scalability, reliability, security and memory management issues play a major role in designing the OS. Thus these issues are reviewed which will guide the future engineers on designing OS.

2. REVIEW ON SCALABILITY ISSUES

Scalability refers to the handling of a growing amount of work and processes or mechanisms to accommodate that growth. Currently, researchers are focusing more on scalability issues while designing operating systems. This study has been focused on the solution of this scalability issue by reviewing different papers on scalability, which was proposed between the years 1986 to 2009.

Acetta et al. designed a new kernel and environment for UNIX multiprocessors named as Multi-User (or Multiprocessor Universal) Communication Kernel (MACH) [12]. This research was conducted to improve the features of the UNIX 4.3 BSD release. The improvements include support for multiprocessors such as furnishing tightly coupled and loosely coupled multiprocessors. In addition, to run multiple threads within a single task in this process was divided into threads and tasks. It was also provided an advanced virtual memory, which designed to make the virtual address spaces sparser and considerably extended. Moreover, it was also provided the facility of capability-based inter-process communication and limited basic system support facilities such as internal kernel debuggers, which is similar to Android Debug Bridge (ADB) and remote procedural language support [12]. In addition to the large and extended virtual addresses, MACH was also provided memory-mapped files, read-write memory sharing between tasks and copy-on-write virtual copy operation. However, to provide all the new features and facilities into existing UNIX 4.3 BSD release, MACH, was integrated portion of servers into the microkernel, which was increased the size of the microkernel being too large. However, the large microkernel size was a big problem on MACH microkernel and other microkernels, which encouraged other researchers to show microkernel as inefficient and inflexible. Therefore, Härtig et al. proved that these redundancies were not coming from the basic microkernel concepts [13]. Instead, it was developed by overloading the kernel in addition to improper implementation. It was provided a suggestion for implementing a minimal primitive on a microkernel, where performance was not the determining criteria rather than some functionality. Nevertheless, this research proved that it would individually measure the performance factor without increasing the size of the kernel. Moreover, this study presented the design techniques followed by increasing the performance factors of the previously designed microkernels. Furthermore, this paper discussed the method with the principles of independence and principles

of integrity, which in turn guaranteed in microkernel designs. It provided an efficient way of managing memory and paging technique. It proved that context switching overhead in the kernel could be improved by 6-10 times by using appropriate microkernel design. However, despite having all these advantages this research was only limited to finding out the inefficiencies of previous microkernel designs. The research has not conferred the method of measuring the finding or results of the above-mentioned techniques. Additionally, this research was not discussing the spatial non-uniformity nature of massively multicore processors, which is an essential parameter for scalable systems.

To solve the scalability and reliability issues of shared memory multiprocessor operating systems a system called Hive was introduced by Chapin et al [14]. The Hive was mainly designed for shared memory multiprocessors. The Hive described the independent kernels as cells and it was structured all the kernels as internal distributed systems. These mechanisms solved the reliability issues, as any fault in the system will only damage one cell at a time rather than damaging the whole system. This research was focused on solving the issues of fault containment and memory sharing among the cells, where fault containment referred to figure out the hardware and software errors into the cells at the time it happened. Additionally, memory sharing among cells would make application performance better in Hive compared to the other multiprocessor operating systems. The Hive has introduced the concept of FLASH firewall that was used to prohibit wild writes in the memory pages. However, despite having all these advantages Hive required some trade-off between performance, scalability, cost and fault containment. Additionally, it was mainly focused on fault containment rather scalability. Moreover, it was only supported up to four processors, which were noticeably less in terms of the current age.

Another research on shared memory multiprocessors conducted by Gamsa et al., where a system called TORNADO was proposed [15]. With the advancement of multiprocessor hardware, costs of cache missing, sharing, and complication due to NUMA-ness had increased. This research was proposed 'TORNADO' to be an optimized operating system that was focused more on the locality of a system, whereas, the previous systems were mainly focused on the concurrency issues. This advancement provided TORNADO with the ability to map locality with the operating system kernel when an application requests for operating system services. TORNADO achieved this locality by introducing an object-oriented structure in its design, where an independent object could represent all the virtual and physical resources [16]. This research was proposed as an efficient way to improve locality and concurrency with the introduction of three key innovations. At first clustering of objects were introduced that was used to partition objects across different processors. To preserve the locality and the concurrency a protected procedural call facility was introduced. Finally, the third key was introduced to improve the locking strategy where all locking protocols are encapsulated within the objects. TORNADO was the simpler structure compared to the other multiprocessor operating systems, which provided with the opportunity to be easily optimized. TORNADO was able to provide a good scalability result for NUMA shared memory machines. However, the design structure of the TORNADO was only supported by NUMA shared memory machines. It left the single-chip multicore processors behind in terms of its design pattern. Additionally, TORNADO requires architecture to support future shared memory, hardware, which reduces its chance of being a good example of a scalable operating system.

K42 was an operating system implemented by the members of several universities, which was mainly focused on scalability issues. It was run on different ranges of multiprocessors and it supported different scales of application [17]. K42 was also an adaptable OS, which gave the ability to manage system resources effectively when the running application changes its need for the system resources. This OS was also extensible, meaning, which could support the addition of different components without interrupting the OS services. K42 was supported open source customization, meaning that anyone could contribute to improving its features. However, by adding this open source feature K42 was compromising the scalability issue they were offering. Moreover, K42 had proposed to work for up to 24 multicore processors, which was inefficient when compared to the latest era of working with more than hundreds of multicore core processors.

DISCO was a system to solve the scalability problem of OS's based on the idea of 1970s popular virtual machine monitoring concept, which was introduced by Bugnion et al. [18]. This approach was proposed to solve the scalability problem without giving much effort in the implementation process. This research was not offered any additional design criteria into the existing operating systems rather solved the scalability problem, which introduced the system software to work between the operating system and the hardware. In addition, this procedure ensured all the functionalities to the reliability offered by the hardware. The system software was the track of multiple virtual machines that were running on multiple processor systems. Level of scalability of different operating systems was leveraged on DISCO. It was also enabled, the file sharing between multiple OS running on virtual machines. However, in this process DISCO only utilized the multiprocessor system as a cluster where the qualitative combination of a cluster was limited to compare with a single system image.

An extended version of the DISCO called Cellular DISCO was proposed by Govil et al. [19]. In this research, an improved hardware partitioning and scalability issues were proposed in DISCO. Based on a DISCO design scheme, Cellular DISCO has converted the large-scale shared memory multiprocessor into virtual machine clusters. Unlike DISCO, Cellular DISCO was supported the fault containment and heterogeneity without affecting the scalability issues. At DISCO, no hardware fault-recovery system was discussed, which was discussed and maintained in Cellular DISCO. Compared to DISCO, Cellular DISCO was more improvised; However, Cellular DISCO was used a rigid cell boundary to maintain the fault containment, which would limit the scalability compared to the recently proposed operating systems. Moreover, similar to DISCO, Cellular DISCO has also utilized the multiprocessor system as a cluster, where the qualitative combination of a cluster was limited while it came to a comparison with a single system image [20].

The advancement of technology and the increased use of the multicore system was a significant effect on the scalability issue. With the increased use of multicore systems, it needed to be ensured that the present systems should have the capabilities of incorporating with the multicore systems. Considering scalability as the prime design concern factored operating system (FOS) was introduced by Wentzlaff and Agarwal [8]. FOS is such a system that target multicore system where scalability was set as the major design concern compared to the other design issues. Additionally, time-sharing was replaced by space sharing in FOS design. This paper illustrated on the FOS and designing it in such a way, that will be efficient enough to handle the challenges of future multicore systems. Additionally, this research was also paid attention to the improved scalability of current single core operating systems.

According to the review of all those researches, it is being noticed that unlike other operating systems suggested to solve the scalability issues, FOS was significantly improved. Comparing to MACH, FOS has introduced a microkernel, which was noticeably smaller. Additionally, FOS was looking for parallelizing within a server rather than parallelizing over different servers. This issue needed to ensure a single high-level functionality of a server. In terms of scalability between the number of cores K42 and TORNADO was able to provide the supports up to 24 cores, whereas FOS was designed to support up to 1000 plus cores. HIVE was designed to provide scalability in multiprocessor systems. However, HIVE was mainly focused on fault containment rather than the scalability issue. FOS was not only focused on scalability claim rather it focused on fault containment. DISCO and Cellular DISCO was able to provide scalability up to a certain level. Conversely, FOS was solely proposed to solve the scalability issue and it was able to compete with these two systems where multiprocessor systems were used as a cluster that was restricting the qualitative interface of a cluster.

3. REVIEW ON SECURITY AND RELIABILITY ISSUES

Now a day, people can recognize few numbers of the operating system, which are successful among numerous invented operating systems as most of the operating systems are failing to fulfill the most significant design criteria such as security. If the system is secured, then it can be used otherwise it is almost similar to an unworthy machine. In this modern era, security becomes the most important feature for any operating systems. Therefore, this section mainly focused on the most significant issue like security, which is carried out by researchers so far.

Between April 1971 and April 1973, Levine et al. an operating system was developed within the Project SUE at the University of Toronto on security design aspects. According to the SUE design, the access control scheme of the design was nearly related to the design issues, which were carried out by the researches in terms of authorization and protection. The main purpose of the SUE Project was to design and implement an OS that was reliable & secured. During the design process of the OS, the objectives of reliability and security issues were compatible. However, the design of the SUE system was finished, but it was not actually documented & implemented yet. Along with this, in the SUE system design, variable-length capabilities were rejected because of much complexity to several system components [21].

Another working prototype was implemented for Linux entitled as REMUS (Reference Monitor for UNIX Systems) was developed by Bernaschi et al [22]. One of the main features of this system was to detect and block any attacks. As a result, the intruder got direct access to the system. REMUS implemented a protected operating system that was less heavyweight extensions of the kernel. Moreover, it was supported to detect immediate possible attacks. Moreover, while the intrusion was in progress, it generally allowed an attack, which was under safe analysis. Nevertheless, this research was mainly focused on how to detect intrusions once attacked instead of protecting the system from being attacked. In addition, it introduced the first major compatibility problems and after that, it was able to mark as non-executable. However, it was possible to solve this problem, but there was still a possibility of attacking, which was mentioned in this research [22].

Generally, UNIX system security lays between good and bad compared to other systems in some specific aspects. One common issue was if any system that provided similar opportunities which were like a UNIX system, might have some problems. In this aspect, the UNIX system was mainly designed to perform

as user-friendly [21]. Several issues were discussed like the insecure nature of passwords, special privileges of administrators, data encryption, protection of files, etc. It is being observed that the UNIX system has the most popular for its design just because people were relaxed with writing, installation and using programs. As it was very popular, so it was very natural that it should be more concerned about its security. Regarding this, software changes were made to increase the security of a system in a high approach. However, there was a common feature for this OS named password aging. Password aging was to change the user passwords every so often if it was activated by the system administrator. The password aging was a critical problem and still, it is unsolved.

On the other hand, another operating system like UNIX was distributed auditing subsystem [22]. It protected the audit trail from unauthorized access. It helped to record the events that were relevant to the maintenance of the security system. For example, the use of authentication and identification mechanisms, in a user's address space the introduction of objects and the deletion of those objects. In this system, it was also concerned about other security relevant events. However, in a highly auditing subsystem, this design was unable to solve the problem. The designers thought that for high security, a general mechanism will be provided so that a subsystem will be able to use it properly and there was no need of making any changes in the design pattern of the subsystem. However, the designers were not that much conscious about the security issue of the system to feel people comfortable.

Again, in terms of security, to attack the computer there exists some malware, which was used by the operating system's privileged operations. In general, some protective security services like vulnerability assessment, antivirus technologies, patch detection, host firewall, network intrusion detection are run as native applications in OS. Although it was in protection stage, still sometimes it failed to find out whether there was an attack or not. To recover these issues, one of the solutions was to protect a computer along with its OS by installing a set of security applications such as personal firewalls, antivirus, etc. It helped some activities such as monitoring an emulated hard drive, monitoring devices and kept some important rules by its performance in terms of integrity verifications [10].

Linux operating system, the security issue was the main concern for the designers. However, there were many security patches available for a different Linux distribution, but there existed a huge malware to exploit the security holes. A suggested system, which was mostly working on the user level and give access to a certain process for a limited period so that even if the process was already compromised in the least damage to the operating system. It mainly worked on user level that makes it the portability to work on different Linux distributions. Normal Linux system interface architecture directly converts a process into a system named as POSIX API. However, in this system, the process was managed by another proposed module called secured daemon (second), which was used to authorize each process that can convert into a system call. This gives the advantage of managing all the processes whether it was malicious or regular. The traditional Linux system only uses identity-based authorization, which may give access to the system resources to a malicious process. Nevertheless, this proposed system added another layer of security using secured daemon module over the identity base authorization, which gives it the access to handle each process by assigning a certain time, so if a process was compromised it could cause the least damage to the operating system. Yet, this system requires a program's resource requirement description file in order to run it, so that it can manipulate the process. A program resource requirement files were manipulated or masked which will cause the proposed system to act normally. Besides, this system was proposed for the single threaded operation to handle the concurrent process request [23].

In this modern civilization, people always search for the best invention to use the best features with the best service and reliability [24]. People are more concerned about its reliability rather than other features. Reliability is one a complement of an OS. LynxOS operating system was designed for the real-time application that was complex and to perform this quick response is needed. This operating system supports round-robin scheduling policies [25]. However, Linux's performance was better compared to LynxOS. On the other hand, to provide a fast interactive response, Windows NT is preemptive. Alike LynxOS, this operating system also follows a round robin scheduling. There has a special class in this OS named Real-Time Priority Class that has high-priority. This OS is better than Solaris and able to optimize code. However, Windows NT was not designed for the real-time operating system. In addition, this OS did not mention any way to skip hardware interruptions & operating system interruptions [26]. Another OS named Solaris is designed to work on uniprocessors. Moreover, this OS also works on symmetric multiprocessors [27]. Here it is to be mentioned that Solaris was the multithreaded implementation of POSIX. However, while executing the scheduling class, to prevent the kernel threads-Solaris provides a real-time scheduling class as a solution, which contains worst-case guarantees [28].

Again, in terms of reliability, researchers showed that the architectural changes to achieve high reliability in the operating systems. At the same time, it is suggested to compare the cost of each architectural change and to consider reliability requirements against the target reliability. To improve system reliability and to enhance system performance, designers of the operating system need to adopt many techniques. However,

it may not be feasible to incorporate all these architectural changes in a single system. Moreover, the loss of data or the state maintained by the module prior to failure needed to be considered during the design phase. On the other hand, the operating system, namely the smartphone operating system (SPOS) developed by J. Wang et al was aimed to enhance the performance of regular phones [29]. An efficient OS was proposed for integrating wireless technologies with the smartphone as well as performing complex information handling using the embedded system. In this research, other issues such as power management, reliability, restriction and wireless connectivity support were discussed. The SPOS kernel design provided some features such as fast reboot, process scheduling, dynamic power management. It was supported multiprocessing using two kind process scheduling algorithm such as round robin scheduling and priority-based preemptive scheduling. Priority-based preemptive scheduling was able to schedule processes that have different priorities. However, round robin scheduling was used when most of the processes have the same priorities, which will ensure the high performance and efficient use of the system in multiprocessing. This proposed system could analyze the CPU and peripheral power requirement for each application. Thus, it could dynamically manage power consumption by adjusting core voltage and frequency. Nevertheless, this system was proposed by only keeping the performance factor, but it was unable to provide a solution for security, reliability, scalability, etc. Moreover, this system was proposed recently to comply with any latest embedded system technologies such as partial view integration in the system view [28].

In this study, several operating system design mechanisms and issues are discussed for different platforms. Eventually, the main concern of an operating system is its security and reliability. No doubt, secured and reliable OS will exist until the end. Distributed auditing subsystem maintains an audit trail from modification or destruction, which is responsible for the authentication and helpful for identification as well. In this system, the designers should be more concern about to make it perfect instead of making software to protect. There is nothing to mention about Linux, as it is one of the most aged long operating system. It is a well-known open source OS. Still, change is happening in this OS. Regarding older facts and issues, new versions are released one by one to recover them. Linux OS, Solaris, Windows NT are not that much well-known or popular OS, but this OS is good enough for some specific features. At the same time, each has some unique specialty, which does not belong to another. Solaris OS needs to have the ability to prevent its kernel thread fully. Similarly, Windows NT should have the capability to skip OS interruptions. The SPOS will ensure the high performance and efficient use of the system in multiprocessing. It also supports round robin scheduling, but it should pay some more concentration about its reliability and security. Moreover, it should have the ability to support modern system technologies as well.

4. REVIEW ON MEMORY MANAGEMENT AND ENERGY EFFICIENCY FOR IOT

The internet of things (IoT) is a system in all the devices, including vehicles, home appliances, software, sensors, and actuators to exchange data [30-31]. There are two types of IoT devices: high-end devices and low-end devices. The operating systems that are used for high-end devices are Windows, Linux, UNIX; which have a high amount of memory. However, low-end devices have a very small amount of memory and work with a low amount of energy. Therefore, while designing an OS for low-end devices the main design issues are memory management and energy efficiency [32].

Contiki was a portable operating system based on Protothread model, which was supported by event-driven and multithreading [33]. The benefit of this system was lightweight and supported multithreading. Moreover, it was provided concurrency to avoid CPU monopolization with preemptive multithreading. Multi-threading provided conditional blocking of sequential instruction blocks as well as share stack, which did not exist for context switching. Different applications for executions were used to save power after observing the event-queue size. However, there was no mechanism to power down the system. Contiki used dynamic memory allocation, but a memory protection unit (MPU) was not implemented in Contiki.

After that, a TinyOS was implemented using the NesC language. TinyOS was using static memory allocation as well as it was the OS with different components. The services of TinyOS were decomposed into a component and the software modules were attached to the hardware [34]. The application of TinyOS used to be able to customize as well as reusable. Moreover, the unused services will be excluded from the application. However, there was no users space. Events in TinyOS are responsible for hardware interrupts. The scheduling of TinyOS use was FIFO in earlier versions. The latest version uses the earliest deadline first scheduling (EDF) preemptive scheduling. The language NesC was used to provide concurrency in TinyOS. NesC made TinyOS to access the hardware in a simpler way. Whereas, the OS has no virtual memory concept; therefore TinyOS cannot allocate memory dynamically.

Lite OS was another operating system for IoT devices with the feature of dynamic memory allocation. LiteOS was a Unix-like operating system, which provided a Unix-based programming environment. The operating system consists of three subsystems: LiteFS, LiteShell and the kernel [35]. The use of LiteShell was to interact with the user could interact via its interface commands. The LiteShell executed complex commands

where executed kernel simple the commands. LiteFS file system provided support to file and directory operations with a concurrent programming model. The callback functions were used to help decrease the number of currently working threads that helped to improve the efficiency of the system. LiteOS supported the dynamic allocation of memory. The memory size of the memory can be adjusted by the user. Every time when the buffer becomes empty, the external flash was also cleared. This mechanism helped the OS to save memory. However, LiteOS was without integrated networking protocols. Similar to Lite OS, Mantis OS there was no feature of the routing protocol. The operating system was written in C and the services of the operating system followed POSIX-like behavior [36]. The threads of this operating system were assigned based on the priority. Mantis OS was supported by round robin scheduling. In terms of accessing global data, the kernel supports static RAM allocation. Moreover, the heap allocates space for the stack. The operating system used binary or counting semaphores to synchronize the threads. The semaphores handle all the events triggered by the driver of the device. When all the threads stop working or blocked, the idle thread takes the kernel to the energy-saving mode. Mantis OS supports dynamic reprogramming, which makes the kernel reprogrammable. The multilayer, as well as the multithread, refers to the operating system to be flexible. However, in terms of performance, it was lower than the single stack thread process. Besides, Mantis OS is not able to support the existing IoT protocols.

Another operating system that can directly support the IoT protocol was Nano-Rk. Nano-Rk wrote in C. The priorities of the scheduling of tasks were fixed [37]. Priority scheduling was two types. The monotonic scheduling referred to assign tasks in a static manner depending on the priorities and period of the job. The small works get higher priorities and finish first. However, harmonized scheduling was referred to save power. All the tasks were grouped together and were executed, which helped to remove the idle sequence. In Nano-Rk, the memory allocation was static. Therefore, it cannot allocate any virtual memory. Besides, Nano-Rk does not have MPU, which is a major design parameter of an operating system.

Similar to Nano-Rk, another operating system named SOS was without any MPU. The components of SOS were loosely paired. The components got together at the time of compiling. There were several versions of SOS; among them, the SOS-2.0+ versions provided more modularity. The component modules were dynamically loaded and were linked [38]. The operating system needs to be reconfigured. Moreover, SOS was a flexible operating system, which means if there were any modifications needed it could be done after deployment. This operating system was dynamically allocated memory. The RAM was divided into two portions: firstly, the core kernel used the RAM, and secondly, RAM used for dynamic memory allocation. However, the operating system was without MPU.

OpenTag was an example of a real-time operating system (RTOS). OpenTag was written in C and existed with Exokernel [39]. The major purpose of Exokernels was to communicate with the hardware directly. Small tasks were assigned at a time and the shortest jobs were completed at first. The operating system used dynamic memory allocation. The major feature of this operating system was to save power used by the sensory devices. In addition, the operating system used a virtual energy reservation system, which allowed the operating system to reserve some energy for future work and increased battery life. However, Nano- Rk supports the static routing protocol.

Very recently, tech giant company Google designed a Linux kernel-based OS named Chrome OS. The Chromium OS uses the Google Chrome web browser as its principal user interface [40]. As a result, Chrome OS primarily supports web applications. Although the operating system is mainly targeted the web application and browsers it is predictable that in future the OS will be used other applications. As the OS is mainly based on Linux Kernel, therefore, the design issues related to Linux OS is applicable for this OS too.

After reviewing all the papers proposed to solve the memory management and energy efficiency issues it is obvious that dynamic memory allocation is available in Contiki, LiteOS and Mantis OS. However, TinyOS and Nano-Rk use static Memory allocation, where TinyOS is not able to allocate virtual memory. Contiki is the operating system where the size of the memory can be adjusted depending on requirements. In SOS, the modules are dynamically loaded and linked. The kernel of Mantis OS has the function of dynamic reprogramming, which allows reprogramming the EEPROM at the device where they can be burnt to flash. However, Contiki, Nano-Rk and SOS have no MPU. In terms of energy efficiency LiteOS, every running application has called back functions that help to decrease the number of current working threads, which improve the efficiency of the system and protect memory. In Nano-Rk, harmonized scheduling is used to power as the Operating system provides priority to shorter jobs first. Contiki implements Application-specific energy conservation, which provides a sleep mode by observing event-queue size [33]. TinyOS implements Software Thread Integration (STI) in order to save battery life.

5. DISCUSSION

The current monolithic operating systems are designed based on the single chips that mainly supports up to 100 cores integrated into it. However, in this growing technology, it has to be ensured that the operating

systems are able to scale themselves up to 1000+ cores integrated into the system. FOS had promised to solve this issue. However, the FOS is still under implementation phase, which is improving day by day. Research can still be conducted in this area to make it more feasible to implement. Although SUE project was developed, yet, it could bring some significant impact on OS design, but the project is not documented yet. In the future, this project can be implemented. It is to be mentioned that the problems that are faced by this OS such as variable-length capability rejection and others will try to be skipped as much as possible. In terms of REMUS, what should make a change is to protect intrusions from being attacked whereas they are just detecting them. Now it is concerned about how to do this which can be another future task. Password aging is a longstanding issue. Although it has been modified to some extent, still there exists an operating system, which was still a problem regarding this issue. This problem solving will reach any operating system in a high position no doubt. For IoT devices, memory management and energy efficiency are the main issues. The operating systems which are discussed above, among them Contiki is better than other operating systems. Contiki supports event-driven and multithreading technologies. The Protothread model provides lightweight threading. Contiki uses dynamic memory allocation. Different functions are used to allocate, de-allocate, declare and defragment the memory. The dynamic memory allocation for the Contiki operating system allows it to adjust capabilities and change the memory requirements during run-time. Moreover, Contiki implemented application specific energy conservation. The applications provide a sleep mode after observing the size of the event queue. However, Contiki has no MPU. In the future, the feature can be implemented in this operating system to make the OS more secure, reliable as well as energy efficient.

6. CONCLUSION

The design issue in OS is changed based on the requirements. After reviewing the major design issues in OS, it is certain that depending on the purpose and function people design the particular Operating System. Especially for tiny OS like the IoT devices the memory management and energy efficiency plays a vital role. This manuscript is reviewed all the major design issues and therefore can be used as a guideline for future OS designers. The review article particularly focused on the major design issues that need to be considered most. As OS is the vital part of any processing devices, therefore this detail review article might be helpful for choosing particular OS as well as help the future researchers while developing an OS.

REFERENCES

[1] M. Kamel, T. Stastny, K. Alexis, R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," *ROS* (pp. 3-39). Springer, Cham, 2017.
[2] M. Marufuzzaman, M. B. I. Reaz, L. F. Rahman and A. Farayez, "A Location Based Sequence Prediction Algorithm for Determining Next Activity in Smart Home," *J. of Engg. Sci. and Tech. Review*, vol. 10, no. 2, pp.161-165, 2017.
[3] M. Marufuzzaman, M. B. I Reaz, M. A. M. Ali and L. F. Rahman, "A Time Series Based Sequence Prediction Algorithm to Detect Activities of Daily Living in Smart Home," *Methods of Info. in Medicine*, vol. 54, no. 3, pp. 262-270, 2015.
[4] S. Zouaoui, L. Boussaid and A. Mtibaa, "Priority based round robin (PBRR) CPU scheduling algorithm," *Int. Journal of Electrical and Computer Engg.*, vol.9(1), pp.190-202, 2019.
[5] M. Tarahomi and M. Izadi, "A hybrid algorithm to reduce energy consumption management in cloud data centers," *International Journal of Electrical and Computer Engineering*, 2019, 9(1), p.554-561.
[6] S. Nurmaini and B. Tutuko, "Intelligent Robotics Navigation System: Problems, Methods, and Algorithm," *International Journal of Electrical & Computer Engineering*, 7(6), pp. 3711-3726, 2017.
[7] B. Banhthasit, C. Jamroen and S. Dechanupaprittha, "Optimal Generation Scheduling of Power System for Maximum Renewable Energy Harvesting and Power Losses Minimization," *Int. J. of Electrical and Computer Engg.,* 8(4), p.1954-1966, 2018.
[8] D. Wentzlaff and A. Agarwal, "Factored operating systems (FOS):the case for a scalable operating system for multicores," *ACM SIGOPS OS Review,* vol. 43(2), pp.76-85, 2009.
[9] J. Kaur and S. R. N. Reddy, "SFrame: Design & Development of Smart Framework to Port Linux Kernel on ARM based Raspberry Pi," *In IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI),* pp. 969-975, 2018.
[10] B. Armstrong, P. England, S. A. Field, J. Garms, M. Kramer, K. D. Ray, "inventors; Microsoft Corp, assignee," *Computer security management, such as in a virtual machine or hardened operating system.* United States patent US 7,409,719. 2008 Aug 5.
[11] A. Musaddiq, Y. B. Zikria, O. Hahm, H. Yu, A. K. Bashir, S. W. Kim. "A survey on resource management in IoT operating systems," *IEEE Access,* vol. 6, pp. 8459-82, 2018.
[12] M. Acetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, "A new kernel foundation for UNIX development," *In Proc. of the Summer 1986 Usenix Technical Conference and Exhibition*., pages 93–113, 1986.
[13] H. Härtig, M. Hohmuth, J. Liedtke, J. Wolter, S. Schönberg, "The performance of µ-kernel-based systems," *In ACM SIGOPS Operating Systems Review*, 31(5):66-77, 1997.

[14] J. Chapin, M. Rosenblum, S. Devine, T. Lahiri, D. Teodosiu, A. Gupta, "Hive: Fault Containment for shared-memory multiprocessors," *In ACM SIGOPS OS Review*, vol. 29(5), pp. 12-25, 1995.
[15] B. Gamsa, O. Krieger, J. Appavoo and M. Stumm, "Tornado: Maximizing locality and concurrency in a shared memory multiprocessor operating system," *In OSDI*, Vol. 99, pp. 87-100, 1999.
[16] Z. Mi, *et al*., "SkyBridge: Fast and Secure Inter-Process Communication for Microkernels," *In Proceedings of the ACM Fourteenth EuroSys Conference,*p. 9, 2019.
[17] J. Appavoo, *et al*., "Experience with K42, an open-source, Linux-compatible, scalable operating-system kernel," *IBM Systems Journal,* vol. 44(2), pp.427-40, 2005.
[18] E. Bugnion, S. Devine, K. Govil, M. Rosenblum, "DISCO: Running commodity operating systems on scalable multiprocessors," *ACM Trans. on Computer Systems (TOCS),* vol. 15(4), pp.412-47, 1997.
[19] K. Govil, D. Teodosiu, Y. Huang, M. Rosenblum, "Cellular DISCO: resource management using virtual clusters on shared-memory multiprocessors," *ACM Transactions on Computer Systems (TOCS)*, vol. 18(3), pp. 229-62, 2000.
[20] Y. Li, J. Zhang, X. Chen, J. Li and J. Ding, "An accurate resource scheduling system for virtual machines based on CPU load monitoring and assessment," *Cluster Computing*, vol. 21(2), pp.1395-1410, 2018.
[21] D. L. Levine, S. Flores-Gaitan, D. C. Schmidt, "Empirical evaluation of OS end system support for real-time CORBA object request brokers," *International Society for Optics and Photonics Multimedia Computing and Networking 2000,* vol. 3969, pp.113-129, 1999.
[22] M. Bernaschi, E. Gabrielli, L. V. Mancini, "Remus: a security-enhanced operating system," *ACM Transactions on Information and System Security (TISSEC),* vol. 5(1), pp.36-61, 2002.
[23] M. S. Hecht, A. Johri, T. T. Wei, D. H. Steves, "inventors; International Business Machines Corp, assignee," *Distributed security auditing subsystem for an operating system,* United States patent US 5,032,979. 1991 Jul 16.
[24] H. Narayanan, V. Radhakrishnan, J. Poroor, "Architectural design for a secure Linux operating system," *in Proc. of the Int. Conf. on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 949-953, 2017.
[25] Lynx Real-Time Systems, *LynxOS-Hard Real-Time OS Features and Capabilities,* http://www.lynx.com/-products/ds lynxos.html, Dec. 1997.
[26] O. Gonzáles, S. Sen, S. Shirgurkar, C. Shen, K. Ramamrithham, "Using Windows NT for real-time applications: Experimental observations and recommendations," *in Proc. of the 4th IEEE Real-Time Tech. and App. Symp.,*pp.102, 1998.
[27] J. R. Eykholt, S. R. Kleiman, S. Barton, R. Faulkner, A. Shivalingiah, M. Smith, J. Voll, M. Weeks, D. Williams, "Beyond Multiprocessing: Multithreading the SunOS Kernel," *in Proc. of the Summer USENIX Conf.*, pp. 1-20, 1992.
[28] S. Khanna, M. Sebree, J. Zolnowsky, "Realtime scheduling in SunOS 5.0," *in Proc. of the USENIX Winter Conf.,* pp. 375–390, 1992.
[29] J. Wang, G. Gu, S. Xie, L. Xu, "Design of smart phone-oriented embedded real-time operating system," *in IEEE 1st Int. Multi-Symp. on Computer and Computational Sciences*, vol. 2, pp. 758-763, 2006.
[30] R. H. Putra, *et al*., IoT: smart garbage monitoring using android and real time database, *TELKOMNIKA*, 17(3), pp.1483-1491, 2019.
[31] R. Gayathri and S. K. Vasudevan," Internet of Things Based Smart Health Monitoring of Industrial Standard Motors," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 6(4), pp.361-367, 2018.
[32] F. Javed, M. K. Afzal, M. Sharif and B. S. Kim, "Internet of things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review," *IEEE Communications Surveys & Tutorials,* vol. 20(3), pp.2062-2100, 2018.
[33] A. Dunkels, B. Gronvall, T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," *29th Annual IEEE Int. Conf. on Local Computer Networks*, pp. 455-462, 2004.
[34] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, T. Schmidt, "OS for the IoT-goals, challenges, and solutions," In *Workshop Inter disciplinaire sur la Sécurité Globale (WISG2013)* 2013.
[35] Q. Cao, T. Abdelzaher, J. Stankovic, T. He, "The liteos operating system: Towards Unix-like abstractions for wireless sensor networks," *Int. Conf. on Info. Proc. in Sensor Networks*, pp. 233-244, 2008.
[36] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, R. Han, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *Mobile Networks and Applications,* vol. 10(4), pp.563-79. 2005.
[37] A. Eswaran, A. Rowe, R. Rajkumar, "Nano-rk: an energy-aware resource-centric RTOS for sensor networks*," In 26th IEEE Int. Real Time Syst. Symp*.,10 pages, 2005.
[38] C. C. Han, R. Kumar, R. Shea, E. Kohler, M. Srivastava, "A dynamic operating system for sensor nodes," *In Proc. of the 3rd Int. Conf. on Mobile Syst., App., and Services*, pp. 163-176, 2005.
[39] D. Engler, M. Kaashoek and J. O'Toole, "Exokernel: An Operating System Architecture for Application-level Resource Management," *ACM SIGOPS OS Review*, vol. 29, no. 5, pp. 251-266, 1995.
[40] Y. Jia, Z. L. Chua, H. Hu, S. Chen, P. Saxena, P., and Z. Liang, "The Web/Local Boundary Is Fuzzy: A Security Study of Chrome's Process-based Sandboxing," In *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 791-804), 2016.