***I. J. Computer Network and Information Security*, 2018, 9, 47-59**
Published Online September 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2018.09.05

# Role of Scripting Language on Unix Operating System for Risk Assessment

**Padma Lochan Pradhan**
Information Technology Dept.TGPCET, RTM Nagpur University, India
E-mail: citrprcs@rediffmail.com

Received: 17 April 2018; Accepted: 12 July 2018; Published: 08 September 2018

***Abstract***—This proposed dynamic scripting language is a vital role in the complex real-time operating system to review, analysis, protect, detect & correct on data, application, network, software and hardware as per the desired manner to achieve the highest performance of the RTOS. The dynamic language is a primarily responsible for analysis of the fine tuning, performance, fault-tolerance, throughput of the system components at the right time. This research work contributes to the design and development of an automated mechanism that objective to investigate the minimal resource utilization to the robust computing services. We have to define & deployment the Unix scripting codes for real time operating system to achieve the reliability, scalability of the real-time MIMD platform. We have to run this SPL on background process to integrate with real-time hardware, software, network to facilitate to other dependants components. This dynamic code is integrating, interfacing, communicating, message passing, replicating among the several subjects and objects over a real-time operating system.

***Index Terms***—Dynamic Scripting programming Language, Processor, Memory, Performance analysis, Access Control Mechanism (ACM), Unix File System (UFS), Procedural Programming Language (PPL), Unix File System (UFS), Preventive Detective Control (PDC), Read Write Execute (RWX).

## I. Introduction

The real time operating system is very fast and quick respondent of any systems programming. These systems are used in an environment where a large number of events (generally internal & external dependants) must be accepted and processed in a short time frame. The real time processing requires quick transaction and characterized by supplying immediate response. The real time programming, applications can be made in almost any language (see, C, C++, JAVA)[9],[26]. The environment (operating system, runtime and runtime libraries) must however be compliant to real time constraints. In most cases, the real-time means that there's always a deterministic time in which something happens. Deterministic timing being usually a very low time value in the microseconds/milliseconds range [2],[4,5].

The Scripting language for Unix operating system was the primary purpose behind the creation of C. Additionally, as programs scripted in C get executed with speeds equivalent to assembly language, C language has been an integral part of the development of multiple operating systems. Unix-Kernel, Microsoft Windows utilities and operating system, applications, and a large segment of the Android operating system have all been scripted in C. Steve Bourne wrote the Bourne shell which appeared in the Seventh Edition Bell Labs Research version of Unix. Many other shells have been written; this particular tutorial concentrates on the Bourne and the Bourne Again shells. Other shells include the Korn Shell (ksh), the C Shell (csh), and variables such as tcsh) [15, 16,17].

The scripting languages are high level programming languages is a set of code, instructions, commands, scripts, other symbols & syntaxes use to write the dynamic scripting application software packages for specific scientific and commercial purpose. The programming languages that the developer use to write source code to solve for a specific problem is called high-level languages (Perl, Java script, PHP, TCL, Python, Ada, BASIC, COBOL, REXX, C, C++, JAVA). These scripting languages do not create any executable, binary, library and link files & no memory will be allocated. The scripting languages are designed to be easy to readable, writable, executable, reliable (RWR), available, robust, scalable and understandable by human being [22], [24, [26]. These languages are just like common English sentences. The programmers can able to write postcode, source code using syntax, logical symbol and just like English words, grammar as per flow chart and the algorithm (physical-design-specification). For example, the control statements and reserved English words like GOTO, for loop, do while, if then else, continue and break are used in most major programming languages to construct the programmed to solve our scientific & business purpose. The logical operators and symbols ( &&, | |, ++, <, >, == and != ) are common syntaxes are available in all most all high level programming languages as available on today on concurrent, parallel and distributed environment. There are many scripting & high-level languages are similar enough that programmers can easily understand source code written in multiple languages like PERL, PHP, TCL, C, REXX, PASCAL, Ada, COBOL, C++, Java & C# for multiple purpose. Now a day, there are many programming languages are available and massively used in both commercial & scientific applications are used in parallel, distributed and concurrent operating system [9], [26], [28].

The scripting language is a high level programming language that supports scripts, programs written in a special run time environment which can interpret (rather than compile) and automate the execution of various tasks that could run one by one in a batch mode operation of system programmers. The RTOS environments that can be controlled through automated scripting including system software, application software, web pages, the shell of RTOS and embedded systems. There are many of the scripting languages help in the dynamic scheduling of the real time operating system [45, 46, 47],[ 49], [53].

The users are accessing the shell, then preventing the UFS through ACM (RWX), that graphical diagram presented here to hand shake with Shell, UFS & Kernel.

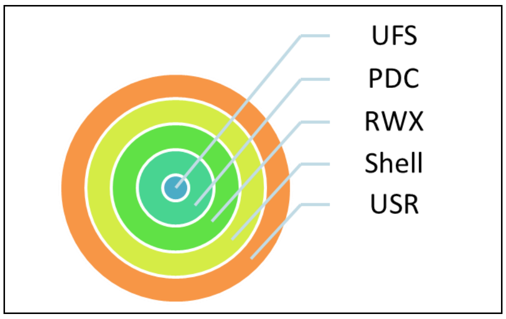

Fig.1. Prevention of OS Components

## II. Existing Scripting Languages

The scripting language is a programming language designed for integrating and communicating with other programming languages. Some of the most widely used scripting languages are JavaScript, VBScript, PHP, Perl, Python, Ruby, ASP and Tcl. Since a scripting language is normally used in conjunction with another programming language, they are often found alongside HTML, Java or C++. The activities of the shell are not restricted to command interrelation alone [9], [26], [28].

The shell has a whole set of internal commands that can be strung together as a language. We need to discuss this language in this paper. In this work, we focus on the Bourne shell-the lowest common denominator of all shells. We have reserved the discussion of the advanced features of the Koran and Bash shell for Part totally different programming constructs and have been separately treated in the proposed method [45, 46, 47] shell program runs in interpretive mode. It is not compiled into a separate executable file as a C, exe, dll, Lib, link programs . Each statement is loaded into memory when it is to be executed. The shell scripts consequently run slower than those written in high-level languages (POP & OOP). However, what makes shell programs powerful is that external UNIX commands blend easily with the shell's internal command modes. The speed is not a factor in many jobs we do, and in many cases, using the shell is an advantage – especially in system administrative tasks. The UNIX system administrator must be an accomplished shell programmer [45, 46, 47, 49].

We have already discovered the basic features of the shell - both as an interpreter and as a scripting language. But the shell is more capabilities than just an interpreter. It is also a process, an environment which makes it available to programs. It is necessary that you understand the environmental changes that take place when the shell execute a program, especially a shell script. We should also know how to change these environmental parameters (see %, $ , # ) [15, 16, 17].

The advanced knowledge of shell programming is needed by the system administrator who has to constantly devise scripts that monitor and correct system functioning in a systematic ways. The detailed knowledge of the shell's subtle features is also necessary if we aspire to be an ethical script writer. The following discussions mostly assume the Bourne shell, but the special features of the Korn and Bash shells are also examined in commands mode. [45, 46, 47], [49], [53].

The most of the scripting languages are often string-oriented, since this provides a uniform representation for many different things. A type less language makes it much easier to hook together components. There are no a priori restrictions on how things can be used, and all components and values are represented in a uniform fashion. These any component or value can be used in any situation; components designed for one purpose can be used for totally different purposes never foreseen by the designer. For example, in the Unix shells, all filter programs read a stream of bytes from an input and write a string of bytes to an output; any two programs can be connected together by attaching the output of one program to the input of the other(&& and | | ). The following shell command stacks three filters together to count the number of lines in the selection that contain the word scripting as follows: select | grep scripting | wc 3, who –Hart | grep top, ps-aef | grep vmstat, ls-ailtr, dmesg. [22], [26], [27], [35].

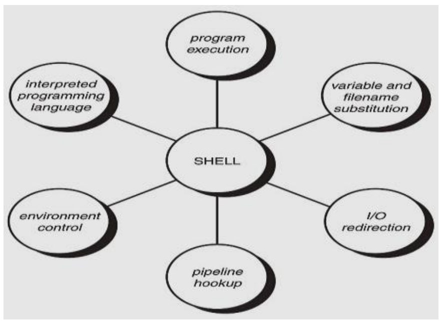

Fig.2. Shell Activities Diagram

## III. Literature Survey

There is a thorough review of the literature was conducted with the primary objective of determining the risk assessment on the Unix operating system for complex IT infrastructure for the large electronic workplace. This research has been conducted on real time Unix file systems applying hardening, re-configuration, access control mechanism, scripting, programming and system programming as per the demand of complex Business, Technology & Resources.

In early 1950 the Main Frame Computers were non-interactive, instead using batch processing. In IBM MF machines Job Control Language (JCL) languages are used to control batch processing.

The first interactive SHELL were developed in the 1960 to enable remote operation of the first time sharing systems, and these used this shell script, which controlled running computer programs within a computer program, the shell. The JCL, REXX & EXEX language are generally credited with inventing command substitution, the ability to embedded commands in scripts that when interpreted insert a character string in the script under IBL CP/CMS, MVS and VMS in 1966 [45, 46, 47], [49], [53].

In early 1950 the Main Frame Computers were non-interactive, instead using batch processing. In IBM MF machines Job Control Language (JCL) languages are used to control batch processing.

The first interactive SHELL were developed in the 1960 to enable remote operation of the first time sharing systems, and these used this shell script, which controlled running computer programs within a computer program, the shell. The JCL, REXX & EXEX language are generally credited with inventing command substitution, the ability to embedded commands in scripts that when interpreted insert a character string in the script under IBL CP/CMS, MVS and VMS in 1966 [49], [51, 52, 53].

Table 1. History of Shell

| Shell Name | Developed by | Where | Remark |
|---|---|---|---|
| BASH ( Bourne-Again SHell ) | Brian Fox and Chet Ramey | Free Software Foundation | Most common shell in Linux. It's Freeware shell. |
| CSH (C SHell) | Bill Joy | University of California (For BSD) | The C shell's syntax and usage are very similar to the C programming language. |
| KSH (Korn SHell) | David Korn | AT & T Bell Labs | -- |
| TCSH | See the man page. Type $ man tcsh | -- | TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH). |

There is a major class of scripting languages has grown out of the automation of job control, which relates to starting and controlling the behavior of system programs. (In this sense, one might think of shells as being descendants of IBM's JCL, which was used for exactly this purpose.) Many of these languages' interpreters double as command line interpreters such as the UNIX SHELL ( Bash, Tcl shell, K shell C shell ) and the MS-DOS command.com ( dir, edit, copy, delete ) just like of English-like commands to build scripts [45, 46, 47], [49], [53].

In order to understand the differences between scripting languages and system programming languages, it is important to understand how system programming languages evolved. The system programming languages were introduced as an alternative to assembly languages. In assembly languages, virtually every aspect of the machine is reflected, affected, corrected in the program. Each statement represents a single machine instruction [2,], [4, 5].

Another key difference between scripting languages and system programming languages is that scripting languages are usually interpreted whereas system programming languages are usually compiled. Interpreted languages provide rapid turnaround during development by eliminating compile times. Interpreters also make applications more flexible by allowing users to program the applications at run-time [49],[ 51], [53].

The scripting languages are less efficient than system programming languages, in part because they use interpreters instead of compilers, but also because their basic components are chosen for power and ease of use rather than an efficient mapping onto the underlying hardware. For example, scripting languages often use variable-length strings in situations where a system programming language would use a binary value that fits in a single machine word, and scripting languages often use hash tables where system programming languages use indexed arrays [45, 46, 47].

The file system /etc/services is protected HTTP port, FTP, Telnet, SMTP and NMTP services This is a service port and /etc/host file protecting the host. We are emphasizing on verification and validation of UFS. We are taken care of protecting; detecting and correcting the Unix file system by applying access control mechanism. We are more conscious about scripting programming on Unix File System server site.

Researcher Ching-Hsien (2014) focused and discussed on Real-Time Multicore architectures and Wireless sensor networks, but we are emphasized on Real Time programming, dynamic scripting language, system programming and as per requirement of business, resources, and technology all the time and every time. We are emphasizing on the business, resources, and technology, which are strongly integrated with BCP & DRP.

According to authors Tiago and Antonio (2014) suggested the Meta programmed C++ for hardware, software integration & communication, but not emphasized on dynamic scripting language, system programming, and system management as per requirement of business, resources, and technology all the time and every time.

Researcher Nassima and Jean (2013) suggested on web application security (HTTP, XSS, Java) for front-end tool, but not risk management (system programming & scripting languages). We are emphasizing on the Unix backend (UFS) security and risk assessment for all the time and every time over a cloud computing, distributed system & pervasive and ambiqutus computing.

Researchers Jan & Maria et al. (2014) emphasized on XML language privacy & authentication, but not a Unix file system, OS scripting and system programming.

Researcher Stephane and Raphae (2014) proposed and worked on the real risk assessment on attack tree, but not over an Unix OS Risk analysis[53] We are emphasizing on the system management on Unix based platform. We are focused and taken care of UFS ACM, scripting and system programming on Unix operating system all the time and every time for web services, ubiquitous, pervasive and self-autonomy system.

According to Martin (2014) developed the application level (SXX) cross side scripting for privacy & security policy, but not on server site programming, scripting on the RTOS. We are more emphasized on RTOS (UFS, scripting & programming) as per requirement of business.

According to Shujun et al. (2014) suggested the threat detection, analysis on the application but not on the Unix operating system. We are taken care of prevention, detection, and correction of Unix file system through access control mechanism (ACM) as per business requirement. The prevention detection correction can be done through (chmod 111 File System) [ Please refers to section 7.3; Action Plan I ).

Authors Marimuthu and Saravanan (2014) are interested in user and application level authentication but not on system security management (privacy & authentication). We are taken care of authentication and privacy, protection, detection and correction of Unix file system through access control mechanism as per users, application and business requirement. We take care of on UFS ACM, trust, authentication, safety, integrity & identity management of the Unix operating system and data management for all the time on web services.

Researchers Diogo et al. (2014) proposed and focused on security issues on cloud computing [29] but not on the Unix operating system. We are emphasizing and taken care of security, issue and risk identification, analysis and assessment of Unix file system (UFS) through access control mechanism as per users, application and business requirement.

The risk of assets is identified in terms of non-identity, integrity, non-repudiation, high availability, authentication, accountability, scalability, and reliability. The critical of each and every risk as rated accuracy of potential impact likelihood of occurrences [7,8].

Therefore, the risk management consists of sets of the possible threat that may be hardware, software and human error, failure, defect uncertainty, unordered, unsafe and the probability of them occurring at any time. The exposure of an asset to a particular threat generally referred as the vulnerability of assets [41], [49,50,51,51,53,54].

The risk is happening, in the course of system operation, maintenance and services. As a result of internal strategies, system processes, policy, procedures, and information used by the organization. The risk analysis is the study of potential threats, vulnerabilities and impacts in order to identify and assess the extent and potential severity of the risks to which the organization and its assets are exposed. The risk assessment is closely associated with risk analysis [41], [49,50,51,51,53,54].

The risk management (identifying, analysis and mitigation) is a process for optimizing the risks to acceptable levels (HML) by the application of various control strategies of prevention, detection, correction, verification and validation. The risk analysis, assessment, and managements are the integrated process for business, technology, and resources for all the time and every time. Assessing the risks and needs of business, resources, and technology have become standard practice in much more IT organization. The understanding of the concepts of risk and need is essential factors for important of decisions involving business, resources, and technology. The risk is created due to the dynamic decision over a business and technology. The risk is propagated over vendor, order, and customers across the all related sub-systems (49, 53, 54].

## *3.1. Data Collection*

These scripting programming language is a high level computer language, the programmers use to develop applications, commands, scripts, and other set of instructions for a computer to execute for specific purposes. We have to find out the several different programming and scripting languages currently listed in our database collection and survey for the specific requirements as per customer's on multi-computer environment for multiple-purpose is defined on the table 2. There are several scripting programming languages, list out in category wise as of today [45], [47], [49], [51, 52].

Fortunately, the performance of a scripting language is not usually a major issue. The applications of scripting languages are generally smaller than applications for system programming languages, and the performance of a scripting application tends to be dominated by the performance of the components, which are typically implemented in a system programming language [45], [47], [53].

Table 2. Data Collection

| SN | Heterogeneous OS | DESRIPTION | TYPES OF LANGUAGES | PURPOSE |
|---|---|---|---|---|
| 01 | UNIX, Win, NT, Linux | APPLICATIONS AND PROGRAMS DEVELOPMENT | C, C++, JAVA, C#, PERL | Commercial & Scientific |
| 02 | UNIX, Win, NT, Linux | ARTIFICIAL INTELLIGENCE DEVELOPMENT | LOGIC (Prolog ), LISP, Haskell, ML | Mathematical & Scientific |
| 03 | UNIX, Win, NT, Linux | DATABASE DEVELOPMENT | Dbase, Fox pro, SQL, My SQL, Sybase, Oracle, Ingress. | Business & Commercial, ERP |
| 04 | UNIX, Win, NT, Linux | GAME DEVELOPMENT | C, C++, JAVA, RUBY, Ant | Entertainment |
| 05 | UNIX, Win, NT, Linux | DEVICE DRIVER (HARDWARE ) | C, C++, JAVA, C#, PERL | Mathematical, Scientific & Benchmarking, |
| 06 | UNIX, Win, NT, Linux | INTERFACE DEVELOPMENT | C, C++, JAVA, C#, PERL | Mathematical, Scientific & Benchmarking, |
| 07 | UNIX, Win, NT, Linux | INTENET, INTRANET & WEB DEVELOPMENT | DHTML, HTML, JAVA SCRIPT, PERL, AWK, SED, Ant, PHP, VBScript. Python. | Business & Commercial, ERP &Web Engineering |
| 08 | UNIX, Win, NT, Linux | SCRIPTING DEVELOPMENT | HTML, XML, VBScriptPerl, PHP, Python, Ruby, Ant, Perl | Business, Telecom. Commercial, ERP, Web & Mobile Mathematical & Scientific |
| 09 | UNIX, XP, Win7-8 | AI, WEB, DLL, Build in UNIX script | Macro,Unix sh, PERL, PYTHON, RUBY, ANT, JAVE, C, HASKEL, ML, LISP | Business, Telecom. Commercial, ERP, Web & Mobile Mathematical & Scientific |

Table 3. Present Scripting Languages

| The Past (1960-2000) Less Demand | The Present (1990-Till Date) High Demand |
|---|---|
| • REXX, JCL, EXEC (1960)<br>• Unix shells: sh, ksh, bash (1971)<br>• Perl (Larry Wall, 1987)<br>• Python (Guido van Rossum, 1990)<br>• Ruby (Yukihiro "Matz" Matsumoto, 1995)<br>• PHP (Rasmus Lerdorf, 1995)<br>• JavaScript (Brendan Eich, 1995) | • Ruby, Perl, Python, PHP etc., are all open source.<br>• Rely on volunteers<br>– Write documentation<br>– Write test cases(RTOS)<br>– Maintain the systems(HW/SW)<br>– Port to new platforms<br>– Fix bugs<br>– Implement libraries(exe, lib, dll, link)<br>– Implement new features |

In the Unix world, the most important application providing such a language was the system shell. In the beginning, the system engineer used the built-in-shell language, but it was so limited that they soon enough started to use more power, general purpose languages write such scripts (Perl, Python, Unix shell etc.). It leads to some people starting to call these scripting languages, even if it is an improper name for them, basically confusing interpreted language called from the shell (interpreted to mean here that human being start a file with a shebang line and do not bother to compile or anything beyond typing the source code) and scripting language [15, 16, 17].

Table 4. Existing Risk Analysis & Audit on Unix RTOS

| SN | How to do ? Scripts | What to do? Description | Risk Analysis (Problem Analysis) What happen & When ?? |
|---|---|---|---|
| 01 | /var/adm/message | System mesg ( event mgmt) | Date & time stamp SECONDARY RISK ASSESSMENT |
| 02 | /var/adm/syslog | syslog system logs | Detective control, Accountability & Authentication |
| 03 | /var/adm/sulog | super user log | Detective control, Accountability & Authentication |
| 04 | /var/adm/loginlog | user login log | Detective control, Accountability & Authentication |
| 05 | etc/ssh/sshd_config | AES, CKM Key mgmt. | Run the scripts: Preventive control |
| 06 | df | Disk fragmentation | Reports file system usage statistics |
| 07 | du | Disk utilization | Summarizes disk usage in a specified directory hierarchy |
| 08 | ls - ialtr | Review the file system with all attributes (10 fields) ACM | File long listing with inode [ACM] PRIMARY RISK ANALYSIS |
| 09 | mount, umount | Mount the UFS on OS | Attaches, or detaches, a file system (super user only) |
| 10 | fsck | Verify the UFS | PRIMARY RISK ANALYSIS |
| 11 | mkfs | Create the new UFS | |
| 12 | quota | Review disk quota/uses | PRIMARY RISK ANALYSIS |
| 13 | lsof | List of open file statistics | List of open file, PRIMARY RISK ANALYSIS |
| 14 | Ps -aeuf | List of processor statistics | Review the all processor |
| 15 | who –a | current user login on the system | Identified the specific user |
| 16 | Last login | last login on the system | Accountability & Authentication |
| 17 | /etc/.profile | USER PROFILE INCLUDING SHELL | Profile file PRIMARY RISK ASSESSMENT |
| 18 | Ls -ialtr | Long listing file system | Verify the FS ACM |
| 19 | ps -auef | Processor detail | FS and task currently running |
| 20 | iostat | Input output statistics | Performance of input & output |
| 22 | pmstat | Processor memory statistics | Processor statistics |

## IV. Problem Statements

- ❖ Skill manpower and programming cost have been always higher than HLP.
- ❖ Automation, Integration & control of many more program & subprogram.
- ❖ Integration, Security, Integrity, Scalability, Interoperability's & high availability are under top management decision.
- ❖ Modification, Updation & Change management are part of the security issue (Change Management: ITIL CMDB).
- ❖ The scripting, codification, updation, modification, change when requirement changes as per past & present Risk Analysis of RTOS.
- ❖ Problem in change management & incident management.
- ❖ Inconsistency in large based program.
- ❖ Increasing the uncertainty, unsetup, unordered & unsafe of HW-SW.

## V. Reseacrch Questions: Why we need SPL?

- We have to optimize HW, SW, Network & Application
- To optimize manpower, cost, space & time
- To maximize the performance of CPU & Memory (Fine tuning, benchmarking).
- To make arrangement for automatic memory management(Automatic garbage collection: AGC)
- Day by day increasing our business (large volume of data, Information, data warehousing & data mining).
- Day by days increasing the millions of users (Web & Mobile computing).
- Increasing the uncertainty, unsetup, unordered, unsafe.
- Day by days increasing the hardware & software capabilities (N-th bits processor & number of CPU, Memory).

## VI. Objective

- Our proposed SPL Programming will be great helpful for (B2B, P2P, G2G,C2C) E-commerce, E-governess, science to science, product to product, business to business & society to society on anywhere & any time.
- Compatibility of past & present requirement of business continuity planning & disaster recovery planning ( BCP/DRP).
- High availability, reliabilities for internal & external system audit (CMDB & ITIL).
- Planning, organised, control, analysis & optimize the system balance among RTOS, Network, Application & Various types of devices, sub-systems, resources & users need (High Fault tolerance).
- Improve the security, reliability, high availability, scalability & interoperability of the RTOS.
- Improve the analysis mechanism of Fine Tuning, Fault Tolerance, and Reliability & High Availability.
- Automation, control, integration of HW, SW application Packaging.

## VII. Proposed Dynamic Scripting Programming Language on RTOS

### *7.1. Define*

We have to define, design, develop and deployment (D^4) this proposes scripting programming languages for web based services and fix up the majors automated system configuration to maintain residual risk. Meanwhile, we have to maintain the system control & balanced by applying the automated method, model, mechanism (M^3) & tools at the operating system level to optimize the risk and maximize the decision management as per business requirement and availability of resource & technology. Our SPL should be designed in such way, that the file system, shell and kernel automatically protected, detected & corrected in around the clock for millions of users. The scope and features of the scripting programming languages are as follows Refers to Figure 2 ].

- ➢ Interpreted (no compilation)
- ➢ Dynamically
- ➢ High-level model of underlying machine
- ➢ Garbage collected
- ➢ Prevented, detected & corrected
- ➢ Organising, co-ordinating, controlling & analysis (integration)
- ➢ Don't have to declare variables
- ➢ Both Batch and Interactive use.
- ➢ Economy of Expression.
- ➢ Lack of declarations; simple scoping rules
- ➢ Flexible, dynamic typing.
- ➢ Easy access to other programs (openness).
- ➢ Sophisticated pattern matching.
- ➢ High-level data types.
- ➢ To support rapid development and interactive use, scripting languages require very little boilerplate codification.

We have to implement dynamic SPL to optimize the system attacks and down time by implementing script based system programming mechanism & automated CPU, Memory management, meanwhile improving the throughput of the UFS, Memory, Processor & Kernel system. Finally, we have to maximize the performance & minimize the cost of the operating system. Our objective is that fix-up the risk at the lowest level with minimal scripting cost and time.

### *7.2. Design*

Proposed Real Time Unix Machine Scalable Metrics The following dynamic, scalable matrix data is helping to our purpose for fine tuning, performance, benchmarking and throughput. We have to design the machine size specification, as per business requirement problem size). The machine size should be greater than the problem size. Meanwhile, we have to update these data dynamically as per implementation of UNIX SHELL SCRIPTING as follows: We have to apply varity of scripting & programming on the following scecifications for betterment of our business & technology.

The following preservation metrics satisfying one program with varity of component or vice versa.

Table 5. Preservations Metrics

| B | X | X | X | X | X | X | X | Business | Unix sh | HA | VLIW | MIMD |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | M | M | M | M | M | M | M | Kernel | M | HA | VLIW | MIMD |
| SPL | Unix sh | Unix sh | Unix sh | Unix sh | Unix sh | Unix sh | Unix Sh | SPL | Unix sh | HA | VLIW | MIMD |
| E | 16 | 32 | 64 | 128 | 256 | 512 | E=2^n | Encryption | Unix sh | HA | VLIW | MIMD |
| UFS | UFS | UFS | UFS | UFS | UFS | UFS | UFS | UFS | UFS | UFS | UFS | MIMD |
| P | 64 | 128 | 256 | 512 | 1024 | 2048 | P=2^n | Processor | UFS | UFS | UFS | MIMD |
| M | 4 | 8 | 16 | 32 | 64 | 128 | M=2^n | Memory(MB) | Unix sh | HA | VLIW | MIMD |
| C | L | L | M | H | H | H | | Control | Unix sh | HA | VLIW | MIMD |

Note: Whereas X: Volume of the business( unknown), M: Kernel Value ( unknown ),VLIW: Very large instructions word, MIMD: Multiple Instruction on Multiple Data, HA: High Availability, C: Control, RM: Risk Mitigation. When control is high, then risk is law as per Fuzz's law applied in to benchmarking.

## *7.3. Development (Mechanism)*

Proposed Automation Scripting Language on Unix Machine

The scripting languages are subsets of sub program, commands, command variable & Unix buildin commands. We can package this sets of commands into the shell programming. This shell programming used in command mode of various shell like k Shell, Born Shell, Bash etc. as per availability of resources on the machine. These simulation of the integrated shell programming languages are developed as per business requirement of the large scale real time Unix operating system.

# Save the program program-name.sh (source code file) & make it as chmod +x Source code file name.

# Run the script # ./program-name.sh or # sh program-name.sh

# Output will be display interactively one by one in batch mode as follows:

ACTION I Prevention & Correction (Action & Reaction Applied To the Newton's Third Law)

```
#ACTION I. PREVENTIVE ACCESS CONTROL
MECHANISM  (INPUT )
#!/bin/sh
#menu.sh: Uses case to offer 8-item menu
#
echo " MENU- RISK ANALYSIS ON UNIX RTOS:UFS ACM
Verification & Validation\n
1.List out Current UFS(Attributes)\n2.List out Open File
system\n3.Long listing of UFS status
4.Modification of UFS-ACM(UGO)\n5.Process Status of USER
\n6.USER status\n7.ID of user on the current system\n8.list of attributes
of UFS(ACM)\n9. Quit to UNIX\nEnter your option:\c"
read choice
case "$choice" in
1)  ls -ailtr > fl ;;
2)  lsof > lsof ;;
3)  ls -iltra > long  ;;
4)  chmod 111 menu*.* ;; [UFS ACM PDC]
5)  ps -aef | grep pl > ps ;;
6)  users > ulist  ;;
7)  who -b > lastboot ;;
8)  ls -iltra > ufslist ;;
9) exit ;;
*) echo "Invalid option "# ;; not really required for the last option
esac
```

Prevention & Correction Action plan for IOT ( Access control Mechanism )

```
echo '#!/bin/bash' > unix-script.sh
$ echo 'echo Hello Script' >> unix-script.sh
$ chmod 777 unix-script.sh
$ chmod 711 unix-script.sh
$ chmod 555 unix-script.sh
$ chmod 511 unix-script.sh
$ chmod 111 unix-script.sh
$ chmod 000 unix-script.sh
$ ./unix-script.sh
Hello Script
$
To make it execuatable file as follow:
$ chmod a+rx unix-script.sh
This script apply to the Table 4 & 5.
```

Detection and Analysis of current Access control mechanism

**# ls –iltra Long listing of UFS**
**Table 6. # ls –iltra | grep “pl” | sort +4n | more (Long listing of UFS)**

| ACM | **Inode Subject Link U G Date Stamp UFS(Object)** | Risk |
|---|---|---|
| 777 | 134208 -rwxrwxrwx 1 pl pl 727 2014-11-08 16:02 menu1.sh<br>141049 -rwxrwxrwx 1 pl pl 461 2014-11-08 16:17 menu4.sh<br>141050 -rwxr-xr-x 1 pl pl 547 2014-11-08 16:37 menu5.sh<br>140886 -rwxrwxrwx 1 pl pl 505 2014-11-09 16:52 menu | H |
| 701 | 123901 -rwx-----x 1 pl pl 727 2014-11-08 16:02 menu1.sh<br>765234 -rwx-----x 1 pl pl 461 2014-11-08 16:17 menu4.sh<br>875431 -rwx-----x 1 pl pl 547 2014-11-08 16:37 menu5.sh<br>213456 -rwx-----x 1 pl pl 505 2014-11-09 16:52 menu.sh | M |
| 777 | 213456 –rwxrwxrwx 1 pl pl 727 2014-11-08 16:02 menu1.sh<br>654123 -rwxrwxrwx 1 pl pl 461 2014-11-08 16:17 menu4.sh<br>908761 -rwxrwxrwx l pl pl 547 2014-11-08 16:37 menu5.sh<br>123456 -rwxrwxrwx 1 pl pl 505 2014-11-09 16:52 menu.sh | H |
| 555 | 213456 -r-xr-xr-x 1 pl pl 727 2014-11-08 16:02 menu1.sh<br>213452 -r-xr-xr-x 1 pl pl 461 2014-11-08 16:17 menu4.sh<br>456123 -r-xr-xr-x 1 pl pl 547 2014-11-08 16:37 menu5.sh<br>234561 -r-xr-xr-x 1 pl pl 505 2014-11-09 16:52 menu.sh | M |

Action on Access control Mechanism #chmod 111 menu*.* >> output file (INPUT )

| | | |
|---|---|---|
| 111 | ---x--x--x 1 pl pl 727 2014-11-08 16:02 menu1.sh<br>---x--x--x 1 pl pl 461 2014-11-08 16:17 menu4.sh<br>---x--x--x 1 pl pl 547 2014-11-08 16:37 menu5.sh<br>---x--x--x 1 pl pl 505 2014-11-09 16:52 menu.s | Low Risk |
| 000 | ---------- 1 pl pl 727 2014-11-08 16:02 menu1.sh<br>---------- 1 pl pl 461 2014-11-08 16:17 menu4.sh<br>---------- 1 pl pl 547 2014-11-08 16:37 menu5.sh<br>---------- 1 pl pl 505 2014-11-09 16:52 menu.sh | Access Deny |

#ls –iltra >> output file ( Review & Analysis of the Action & Reaction ) (INPUT)

```
INODE   UFS-ACM linkUSR_GRP    DT & Time        UFS
141044 -rwxrwxrwx  1 pl pl   260 2014-11-08 15:19 CPU.sh
141049 -r--r--r--  1 pl pl   461 2014-11-08 16:17 menu4.sh
141050 -r--r--r--  1 pl pl   547 2014-11-08 16:37 menu5.sh
141052 -rwxrwxrwx  1 pl pl   750 2014-11-09 08:12 authentication.sh
140886 -r--r--r--  1 pl pl   505 2014-11-09 16:52 menu.sh
141063 -rwxrwxrwx  1 pl pl   839 2014-11-10 19:48 perftest.sh
141051 -rwxrwxrwx  1 pl pl   830 2014-11-10 19:56 performance.sh
141060 -rwxrwxrwx  1 pl pl   215 2014-11-11 07:38 CPU1.sh
                        OUTPUT
271323 drwxrwxrwx  3 pl pl  4096 2014-11-11 12:26 kamal
141142 -rwxrwxrwx  1 pl pl   775 2014-11-27 10:04 ACM-
SCRIPT271114
271331 -rwxrwxrwx  1 pl pl 15533 2014-12-03 12:26 test3
399032 drwxrwxrwx  3 pl pl  4096 2014-12-04 19:59 Program
134208 -r--r--r--  1 pl pl   725 2014-12-15 12:04 menu1.sh
271321 -rwxrwxrwx  1 pl pl   775 2014-12-15 12:21 ACM.sh
269357 -rwxrwxrwx  1 pl pl   766 2014-12-15 12:29 ACM
271396 -rw-r--r--  1 pl pl  1246 2014-12-15 21:36 long
271354 drwxr-xr-x  2 pl pl  4096 2014-12-16 10:19 IJRCTS
141036 drwxr-xr-x 33 pl pl  4096 2014-12-16 11:22 ..
271399 -rw-r--r--  1 pl pl     0 2014-12-16 11:23 ufslist
269614 drwxr-xr-x  5 pl pl  4096 2014-12-16 11:23 .
```

ACTION II. Prevention & Correction (INPUT )

```
# ACTION 2 RISK ASSESSMENT ON UNIX  RTOS:
Authentication Verification of USR & Mechine.
#!/bin/sh
#menu.sh: Uses case to offer 8-item menu
#
echo " MENU- RISK ASSESSMENT ON UNIX RTOS:
Authentication Verification\n
1. Listout Current USERS(who) \n2. List out last
users(last)\n3. USERS Status(users)
4. List of open file(lsof) of the RTOS \n5.
Process Status of USER(ps -auef|grep pl) \n6.
Process Status of RTS(ps -aufe)\n7. ID\n8. list
of attributes in UFS(ACM)\n9. Quit to
UNIX\nEnter your option: \c"
read choice
case "$choice" in
     1)  who -Hart | more ;;
     2)  last | more ;;
     3)  users ;;
     4)  lsof |more  ;;
     5)  ps -aef | grep pl | more ;;
     6)  ps -aufe  ;;
     7)  id ;;
     8)  ls -iltra | more ;;
     9) exit ;;
      *) echo "Invalid option "      # ;; not
really required for the last option
esac
```

REACTION
#Who –Hart >> output file ( Authorize user on the system ) (INPUT )

```
pl@pl-HP-15-Notebook-PC:~$ who –Hart ( Users Authentication )
NAME     LINE      TIME           IDLE        PID COMMENT
EXIT
          system boot  2014-11-21 06:50
          run-level 2  2014-11-21 06:50
LOGIN    tty4      2014-11-21 06:50          834 id=4
                OUTPUT
LOGIN    tty5      2014-11-21 06:50          839 id=5
LOGIN    tty2      2014-11-21 06:50          850 id=2
```

#last >> **output file** (last users on the system) ( INPUT )

```
pl      pts/0       :0.0            Tue Dec 16 11:22   still logged in
pl      tty7        :0              Tue Dec 16 11:22   still logged in
reboot  system boot  2.6.35-22-generi Tue Dec 16 11:22 - 11:26
(00:03)
pl      pts/0       :0.0            Tue Dec 16 10:14 - 10:20  (00:05)

pl      tty7        :0              Tue Dec 16 10:14 - down   (00:05)
reboot  system boot  2.6.35-22-generi Tue Dec 16 10:14 - 10:20
(00:05)    OUTPUT
pl      pts/0       :0.0            Mon Dec 15 21:19 - 21:39  (00:20)
pl      pts/0       :0.0            Mon Dec 15 20:43 - 21:09  (00:26)
pl      tty7        :0              Mon Dec 15 20:41 - down   (00:58)
reboot  system boot  2.6.35-22-generi Mon Dec 15 20:41 - 21:39
(00:58)
pl      tty7        :0              Mon Dec 15 17:32 - down   (00:00)
```

ACTION III. Prevention & Correction

```
# ACTION 3 RTOS Performance & Fault Analysis (INPUT )
#!/bin/sh
#menu.sh: Uses case to offer 8-item menu
#
echo " MENU- RISK ASSESSMENT ON REAL TIME UNIX
OPERATING SYSTEMS: PERFORMANCE, BENCHMARKING &
FAULT TOLERANCE VERIFICATION-ANALYSIS\n
1. Listout CPU Status \n2. listout Memory status\n3. Free Memory(free)
4. Uptimes of the RTOS \n5. Load Factor of UNIX RTS \n6. Process
Status of RTS\n7. Process Tree\n8. Inter Process Comm\n9. Disp Log
Msg of Components\n10. Quit to UNIX\nEnter your option: \c"
read choice
case "$choice" in
    1)  top > toptxt ;;
    2)  vmstat -a > vmtxt ;;
    3)  free -mt > freetxt ;;
    4)  uptime > uptimetxt ;;
    5)  w > wtxt;;
    6)  ps -aufe > pstxt ;;
    7) pstree | more > pstreetxt ;;
    8) ipcs > ipcstxt ;;
    9) dmesg > dmlog;;
    10) exit ;;
    *) echo "Invalid option "      # ;; not really required for the last option
esac
```

REACTION

```
#uptime ( Uptime & Response Time of the system ) (INPUT )
11:28:01 up 5 min,  2 users,  load average: 0.16, 0.27, 0.16
--------------------------------------------------------------------------------
#free –m
          total      used      free     shared   buffers     cached
Mem:      3996044    325952    3670092        0    38636     123156
              output
-/+ buffers/cache:    164160    3831884
Swap:     1084412         0    1084412
```

```
#vmstat ( Virtual memory ) (INPUT )
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu---
-
 r b w swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 0 0 0    0 3670176  38644 123172    0    0   248     9  112  210  3  1 94  2
output
```

ipcs Inter process communication system

```
#ipcs (INPUT )
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 0          pl         600        393216     2          dest
0x00000000 32769      pl         600        393216     2          dest
OUTPUT
0x00000000 65538      pl         600        393216     2          dest
0x00000000 98307      pl         600        393216     2          dest
0x00000000 131076     pl         600        393216     2          dest


------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

top & uptime scripts display the Unix server activities report

```
#top  >> output file  ( INPUT )
top  (CUP Utilization, no of users, processes activities
07:12:33 up 22 min,  2 users,  load average: 0.36, 0.40, 0.27
Mem:   3996044k total,   363144k used,  3632900k free,    38668k
buffers
Swap:  1084412k total,        0k used,  1084412k free,   153244k cached
               OUTPUT

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+
COMMAND        OUTPUT
 1540 pl        20   0 91516  13m  10m S    2  0.3   0:10.29 gnome-
terminal
 1070 root      20   0  108m  28m 8116 S    2  0.7   0:19.90 Xorg
 1586 pl        20   0  2624 1116  840 R    1  0.0   0:00.71 top
```

```
#ps –uef| grep top >> output file  ( INPUT )
Warning: bad ps syntax, perhaps a bogus '-'? See
http://procps.sf.net/faq.html
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START
TIME COMMAND
pl        1544  0.0  0.0   6620  3276 pts/0    Ss   06:53   0:00 bash
PATH=/usr/
pl        1619  0.0  0.0   4560   996 pts/0    R+   07:17   0:00 \_ ps -uef
ORB
```

Others Sceripts
#Ls –iltra (Long listing file systems along with inode ordinary & directory UFS )
#who –b system boot 2014-12-16 11:22 Last boot of the Unix Machine

We have to test these scripts on a Unix machine on command mode on heterogeneous hardware & software for fine tuning & benchmarking purpose to evaluate the operating system performance (Processor, memory, space & time complexity). Accordingly the test result, we can able to validate the operating system components as described in below. (Refers to Table. 4-5)

### *7.4. Deployment (Implementation)*

Theoretical practice on fine tuning, performance analysis & benchmarking

We have to explore new invitations to compare various types of real time UNIX operating system parameters like processor, memory, file system, kernel on identical scripting language or virtual programming of Java or C++ verification & validation of standard application product like B2B, B2C, M2M, & P2P. The relative performance of the systems on identical tasks is more important to us than the absolute best performance that could be achieved for any individual system through system specific fine tuning and performance analysis. For comparison purpose ,we have only one or more source code available for testing of Processor, Memory & Kernel parameters, our benchmarking methodology is the black box approach available in sequential & as well as a randomize way to determine our objective. We are usually attempting to explain curious & interesting results through continuous testing and benchmarking rather than detections of Memory (space), CPU & Kernel, time code etc.

### *7.4.1. Practical Approach for Risk*

We have to verify, audit & analysis the real time operating system integrity, high availability, reliability, scalability of scripting programming languages on heterogeneous operating system platforms and we have to study the behaviours of the subsystem of the operating system like: Shell, File, Kernel, Processor, Memory & Encryption key as per business requirement & availability of technology. We can apply some review method for an internal UNIX operating system for our benchmarking purpose to mitigate the risk factor. This benchmarking method can be applied in traditional as well as Web based application which is going to be facilitated to the performance analysis, benchmarking, fault tolerance and risk management over a complex real time operating system.

### *7.4.2. Experimental Practices of Benchmarking:*

*Under SUN SOLARS UNIX*

Table 6. Brief Summary of Verification & Analysis

| SN | ACTION PLAN (SUBJECT-INPUT) | DESCRIPTIONS | RISK ANALYSIS (OBJECT-OUTPUT) |
|---|---|---|---|
| 01 | Iostat (free) | Input /output statistics | CPU & Device Utilization, HA availability, Reliability & integrity of Processor. PRIMARY RISK ASSESSMENT |
| 02 | Pmstat | Processors statistics | Global Statistics among all the processors & users : PRIMARY RISK ASSESSMENT |
| 03 | Vmstat | virtual memory statistics [MEMORY ACTIVITIES] | Statistics of all the processor runable, block, swap, free buffer, input/output block devices, CPU detail, system, user, idle, waiting stage. HA availability, Reliability & integrity of Memory. PRIMARY RISK ASSESSMENT |
| 04 | Sar, w, uptime, top | system activities | Activities report on:no of users in last 5-10-15 minutes of OS detail. PRIMARY RISK ASSESSMENT |
| 05 | ps –ef \| grep top | ACTIVITIES OF PROCESSOR | The suspious processor or orphan/dead one. [space & time complexity issue] SECONDARY RISK ASSESSMENT |
| 06 | lsof l more | FILE SYSTEM ACTIVITIES | list of open files system which is very high risk. SECONDARY RISK ASSESSMENT |
| 07 | /etc/system | KERNEL SYSTEM ACTIVITIES | Can be update the kernel PRIMARY RISK ASSESSMENT |
| 08 | Ls –alitr | Veify UFS ACM | Read, Write, Execute ACM |

### *7.4.3. Result Analysis (Event Management)*

The subjects and objects can able to mapping, integrate, communicate, synchronize and optimize through real time operating system. This SPL tool, script, commands, programming utilities and application will be more measurable, accountable, actionable for fine tuning, performance, fault tolerance, throughput, benchmarking and risk assessment of any application over a real time unix platform (HPUX, SUN SOLARIS, AIX, LINUX).

We have to further analysis & investigate the problem size and the machine size based on the proposed prevention matrix table (Refer Table. 3, 4, 5 & 6), then we can decide the programming and application ( PHP, PERL, RUBY, ANT, JAVA/C++), that can be run on the heterogeneous RTOS UNIX based operating system in a background process. We have to further analysis to detect our problem (dmesg & /var/adm/message ) the logs of scripting program application issue as well as space utilization, performance of RAM & Cache etc. ( space & time complexity). Meanwhile, we can use the various system commands and scripts for further review and analysis of the many more UNIX based real time operating system. There are some practical approach is highlighted as follows.

- ❖ How is behaving the real time UNIX server along with its sub components, when we are running on the same scripting on different processor, memory & encryption key on the same programming & application or reverse way?
- ❖ The uptime, vmstat, top, free, iostat, vmstat and pmstat commands will be given the full detail output statistics of users, file system, processor and memory on real time operating system. The primary defect & risk can be analysis of right time and right way on any real time Unix machine.
- ❖ How far are the real time operating system components (hardware, software, application, network bandwidth & related sub devices) maintaining the risk (High, Medium & Low) for real time system? The who, user, login, netstat-

ntpul scripts will be given the output statistics of network (TCP/UDP) bandwidth and related components on any platform. The primary problem, defect, error, fault & risk can be analytically on everywhere and every time in around the clock (24 x 7).

- ❖ The top management can able to adopt DSS on IT infrastructure, which is best script will be a suite for our recent business & technology like pervasive and anti-fragile technology for web and mobile application.
- ❖ The dmesg & /var/adm/message scripts will be given the output statistics of hardware and software problem of real time event management system including date and time stamp on real time unix machine.
- ❖ How is the system behaving, when millions of users accessing the same piece of data & information in around the clock ( high availability, scalability and reliability of scripting commands).
- ❖ We can only analysis & review practically based on theoretical idea. But, we have to review and justify the system behaviour of space & time complexity based on machine size and problem sizes( Refer Table 4 & ).
- ❖ On behalf of the /etc/system script, we can update and improve the kernel capability as per business requirement and that can be helped with our machine size and problem sizes analysis (Refer Table 3, 4 & 5).

In this way, we can improve the performance, benchmarking, fault tolerance and risk assessment at a time to utilizing the best scripting programming language, which is help to our resources, business, technology and society in around the globe.

### *7.4.4. Why we need Fine Tuning & Benchmarking?*

We have to make arrangement the specific fine tuning, performance analysis & fault tolerance, which is satisfying to our present as well as future BCP & DRP. The top management desired to keep the uptime machine for high availability, reliability & scalability for all the time and every time.

## VIII. ADVANTAGE

- Optimizing the programming codification & complexity.
- Improve the certainty, unification, simplification, normalization & ordered.
- Optimizing the hardware, Network, software, application resources as per business requirement.
- Optimizing the Application, USERS, CPU, Memory (SPACE), Cost & Time.
- Highly secure, readable, writable, reliable, scalable and high availability.
- Codification is faster, easier & scalable.
- Best practice of ROI, TCO, ROA, TQM for BCP/DRP & support to top management for dynamic DSS.

## IX. BENEFITS FOR UNIX SYSTEM PROGRAMMER

The Unix system programming definitely improved the high availability, scalability, security and portability of hardware, software, application and network.

The faster development through the increased number of standard interfaces with other programming languages as per ISO CMM standard.

Many more innovations are possible due to the optimized the cost & time for porting applications on heterogeneous platform.

## X. CONCLUSION

The risk analysis and optimization are the two parts of the same coin. The risk analysis, investigation and optimization are great protections on hardware, software, operating system and programming languages, when applications have been written and are working efficiently &correctly. The best design decisions and best practices in scripting application design and implementation provide a sort of pre-optimization by eliminating many of the source coding issues that might have required optimization. We can minimize the CPU & Memory (space-time) load to achieve the minimum cost and time to run our business in the right way and right time in around the globe.

Nowadays, increasingly sophisticated deployments introduce additional complexity tools in the system, on complex infrastructure and application interaction. The real time operating system and hardware selection are the best practices for fine tuning, performance analysis, benchmarking & monitoring can help to maximize application performance today and high availability, reliability, scalability, extenbility for tomorrow.

The RTOS like UNIX systems have always pioneered the high-performance networking, distributed applications and distributed storage managements that are the underlying of robust, high-performance N-tier web and mobile applications. However, the optimizations within N-tier applications themselves can make substantial contributions to the performance of those (PERL, JAVA, PHP, RUBY C++) applications in addition to making them easier to understand, more maintainable and more available, scalable & reliable for internet programming (IOT).

This analysis and optimization approach helps to the any organization the complete more effectively in local, national and international markets at any time and any place around the clock as follows:

Minimizing costs, risk and maximize profits, DSS, ROA, ROI, TQM & TCO.

The top management can find the solution of maximizing the utilization of Man, Machine, Material, Market, Money & Method (M^5) to solving multiple problems at the right time with optimal cost to utilizing overall resources more effectively in a timely manner.

## References

[1] A.K. Gupta, Management Information System. New Delhi, India: S Chand Publishing, 2012.

[2] Andrew and Richard, UNIX Network Programming. New Delhi: Person Education India, 2011.

[3] Andrew Tang, "A guide to system penetration Testing," System & Network Security, August, 2014.

[4] Adam Hoover (2010). System Programming with C & Unix, Delhi: Pearson India.

[5] Andrew, Bill; Richard (2011). *UNIX Network Programming*. New Delhi India, PHI.

[6] Adrian Waller (2014). Editorial: Special issue on Identity Protection and Management, Journal of information security and application, Vol.19.

[7] Andrew Tang (2014). A guide to penetration Testing, Network Security.

[8] Andrian Devis (2011). What Is Critical to Your Infrastructure?," Information security, Volume 8, Issue 5, Pages 18–21.

[9] Balagurusamy, E.B. (2010). Object Oriented Programming C++. New Delhi, India: Tata McGraw Hill.

[10] Balagurusamy, E.B. (2007). *Programming in Java*. New Delhi, India: Tata McGraw Hill.

[11] Byrons Gottfried.(2009). *Programming with C***.** New Delhi, India: Tata McGraw Hill

[12] Colin Ritche (2006) Operating System in Unix & Window, New Delhi India, BPB Publication.

[13] Ching-Hsien Hsu (2014). Real-time embedded software for multi-core platforms, Journal of Systems Architecture, Vol. 60, 245–246.

[14] Diogo A. B. Fernandes, "Security issues in cloud environments: a survey," International Journal of Information Security, Springer. 13:113–170, 2014.

[15] Debasis, Jana (2010). *C++ & Object Oriented Programming*. Delhi: Pearson India.

[16] EKTA WALIA (2002) Operating Concept New Delhi India, Khanna Book Publishing

[17] Franklyn & David. (2008). *Design Concept in Programming Languages*. Delhi:PHI.

[18] Finne, T. (2000). Information Systems Risk Management: Key Concepts and Business Processes. Computers & Security, 19(3), 234–242. doi:10.1016/S0167-4048(00)88612-5.

[19] Forte, D. (2009). Security audits in mixed environments, *Network Security, 3*(3), 17-19.

[20] G. Posta and A. Kaganb, "Computer security and operating system updates,"Information and Software Technology, Vol. 45, pp. 461– 467, January 2003.

[21] Godofredo, Julio, "System performance evaluation by combining RTC and VHDL simulation: A case study on NICs," Journal of Systems Architecture, Vol. 59, 1277–1298, 2013.

[22] Herbert, Scheldt. (2010). *The Complete Java Reference*. New Delhi, India: Tata McGraw Hill.

[23] Hwang, Kai. (2008). *Advance Computer Architecture*. New Delhi, India: Tata McGraw Hill.

[24] Richard (2010) Unix Network Programming New Delhi India, PHI

[25] Jan Camenisch and Maria (2014). Concepts and languages for privacy preserving attribute-based authentication," Journal Of Information Security And Applications, 19, 25-44.

[26] Khalid, A. (2008). *Programming Guide to Java*. Delhi, India: Pearson India.2008.

[27] KVN Sunitha & N. Kalyani (2008). Programming in Unix & Computer Design. Hyderabad: BSP Publication.

[28] Kumar Saurabh (2008). Unix Programming New Delhi India Wiley India.

[29] Kumar N Rao Java Programming Dream Tech New Delhi 2009.

[30] Maurice J. Bach ( 2012). The Design of Unix Operating System New Delhi India.

[31] Mathew NichoShafaq (2014). Identifying Vulnerabilities of Advanced Persistent Threats: An Organizational Perspective," International Journal of Information Security and Privacy,8(1), 1-18.

[32] Martin Johns, "Script-templates for the Content Security Policy," Journal of information security & application, 19, 209-223, 2014.

[33] Marimuthu Karuppiah and R. Saravanan, "A secure remote user mutual authentication scheme using smart cards," Journal Of Information Security And Application, Vol.19, 282-294, 2014.

[34] Nassima Kamel and Jean-Louis Lanet (2013). Risks induced by Web applications on smart cards, Journal of Information Security & Application, 18, 148-156.

[35] O' Reilly. (1995). Essential of system administration. O' Reilly Media. USA

[36] Paul Love & Paul Weinstein (2005). Beginning Unix, New Delhi India, Wiley India.

[37] Pressman. (2001). Software engineering. New Delhi, India: Tata McGraw Hill.

[38] Poorna Chandra, Sarang.(2009). *Object Oriented Programming with C*. Delhi, India: PHI.

[39] Ravichandran, D.(2009). *Programming with C++*. New Delhi, India: Tata McGraw Hill.

[40] Robert, Lafore. (2002). Object Oriented Programming in C++. New Delhi: Person India.

[41] Shon, H. (2002). *Security mgmt practices*. New Delhi, India: Wiley Publishing Inc.

[42] Stephane Paul, Raphae and Vignon-Davillier, "Unifying traditional risk assessment approaches with attack trees," Journal of information security & application, Vol. 19, 165-181, 2014.

[43] Seyed.H. Roosta.(2004). Foundation of Programming Languages Design & Implementation. Delhi: Cengage 2004.

[44] Shujun Li, Konra Rieckand Alan Woodward, "Special issue on threat detection, analysis and defense," Journal Of Information Security and Application Vol. 19, 163-164, 2014.

[45] Srimanta Pal .(2011). *System Programming* " Delhi: Oxford University Press.

[46] Sumitabh, Das. (2009). *UNIX system V UNIX concept & application*. Delhi, India: Tata McGraw Hill.

[47] Sun-Microsystems. (2002). *UNIX Sun Solaris system administration*. USA,

[48] Stalling, William. (2006). *Cryptography and network security*. New Delhi, India: Person India.

[49] Stalling. (2009). *Operating System Internals & Design Principle*. New Delhi, India: Person India.

[50] Stephen Prata (1986). Advanced Unix Prammer Guide, New Delhi India, BPB Publication.

[51] Tanenbaum. (2010). *Operating System Design And Implementation*. New Delhi, India: Person Education India (PHI).

[52] Tanenbaum. (2009). *Computer Network*. New Delhi, India: Person Education India (PHI).

[53] Tiago and António Augusto (2014). A meta-programmed C++ framework for hardware/software component

integration and communication," Journal of Systems Architecture Vol. 60, 816–827.

[54] Weber, Ron. (2002). Information system control & audit. New Delhi, India: Person Education India (PHI).

[55] Younis A. Younis, Kashif Kifayat and Madjid Merabti, "An access control model for cloud computing,"Journal Of Information Security And Application.Vol.19, 45-60, 2014.

## Authors' Profile

**Prof. (Dr.) Padma Lochan Pradhan,** received his M Sc (Physics with Electronics) from Sambalpur University in 1983, M Tech in Computer Science in 2012 from Berhampuer University & Ph D in Computer Science & Engineering in 2017 from SOA University Bhubaneswar, India. He is interested in Big Data, Data Science, System security, cryptography, operating system, system programming & Risk mgmt. He has 18 year experience in IT industries & 14 years in academic & research in various capacities in IBM, Sun Micro system, Thomson scientific (ISI) in USA and Indian Telephone Industries etc. At present he is working as a Professor in Information Technology Dept. TGPCET under RTM Nagpur University, India.

Apart from this, Dr. Pradhan completed Diploma in Business Administration in 1997; Certified UNIX Sun Solaris in 2002 from Sun Microsystems USA & Certified Information System Auditor from ISACA USA. He published twenty four research papers in internationaljournal on access control mechanism, operating system security & risk mgmt.

**How to cite this paper:** Padma Lochan Pradhan,"Role of Scripting Language on Unix Operating System for Risk Assessment", International Journal of Computer Network and Information Security(IJCNIS), Vol.10, No.9, pp.47-59, 2018.DOI: 10.5815/ijcnis.2018.09.05