UNIVERSITY OF NAIROBI

FACULTY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTING AND INFORMATICS

Multi-Tenancy Environment Provisioning on a Non-Virtualized Cloud (Bare Metal) Platform

Anthony Kibet Ng'eno

P58/70786/2008

Supervisor:

Prof. Robert Oboko

Research Project Report Submitted in Partial Fulfilment of the Requirements of the Degree of Master of Science in Computer Science of University of Nairobi

November 2023

DECLARATION

I declare this research project report is my original piece of work and has not been presented to any university before for an academic award

Signature: [illegible] **Date** 01-12-2023

Anthony Kibet Ng'eno

P58/70786/2008

This Research Project Report has been submitted for examination in partial fulfilment of the Requirements of the Master of Science in Computer Science of University of Nairobi with my approval as the assigned Supervisor from the University

Signature: [illegible] **Date** 6/12/2023

Professor Robert Oboko

Lecturer

Department of Computing and Informatics

University of Nairobi

ACKNOWLEDGEMENT

I thank God the Almighty for sufficient grace and for the gift of life and provision and supply

To my family, I salute you for the support, encouragement and exhortation.

To my departed parents, your dream lives on…….

In a singular and profound way, I acknowledge and appreciate my esteemed Supervisor Prof. Robert Oboko for the support and guidance throughout this project period May God bless you abundantly

I appreciate and thank the faculty members of the Department of Computing and Informatics and special mention goes to my panel assessors, Prof Elisha Opiyo, Prof. Peter Wagacha and Christine Ronge for their invaluable guidance and in-depth review of my work. Be blessed beyond measure.

Finally, kudos to the University of Nairobi for remaining the premier institution and centre of academic research excellence

ABSTRACT

Cloud computing consists of compute and security and network resources provided and maintained by Cloud Service Providers (CSP) that enable cloud users to access and process workloads in a remotely located compute resources.

The cloud providers offer varied services to appeal to the would-be subscribers and the fate of any cloud provider with regard to cloud capture and market share depend on how competitively packaged, robust and agile the CSP service offerings are.

Cloud Computing is equated to access to utilities offered by utility companies. The utility companies pipe and terminate the utilities to points near consumers who use requisite equipment to tap. Cloud computing is a collection of IT resources that are accessible in multiple ways viz: via web browser or through other CSP provided remote access tools.

CSPs invest in massive infrastructure and service requestors also known as tenants are then provided with virtual resources with ordered specifications or in increments of pre-defined resource capacity shapes. This slicing of a big physical resource to create a sub resources and separation of these derivative resources is achieved through virtualization technology. More recently, isolation has been achieved at Operating System (OS) level through application of containerization technologies.

The massive infrastructure is sliced using different technologies to allow multiple and independent users access the resources without any data or access leaks. The main technology used is virtualization which is either hardware or host-based. Recently, containerization technology has been adopted but this is mostly at application layer level.

However, the existing service enablers specifically virtualization technologies provision virtual resources based on preset configurations, compute shapes and based on inbuilt capabilities. This means that customers choices are limited to pre-defined capacities forcing customers to purchase excess capacity. This limitation is causing the customers to forfeit the elasticity feature that is a defining characteristic of cloud platforms. Containerization technologies are implemented based on a small set of features that do not guarantee complete isolation as desired.

There is need to understand how the use of current cloud multi tenancy enabling technologies and use of Linux Kernel features to facilitate multi tenancy and resource isolation in cloud computing. The findings will then be used to explore an alternative way that can complement or improve current technologies or devise a better and more efficient mechanism altogether.

This study aimed at exploring the possibility of achieving multi tenancy through complete isolation using non-virtualized approach but based on In-Kernel Linux features. The study was based on CentOS Linux distribution. The study involved detailed understanding of the structure and components of the Linux Kernel and configurations that can be tuned to refine, activate, or deactivate the features and functional behaviour of the computer host. The study was based on Linux complete Code Base downloaded from the Gitub web repository. This was used to re-affirm the features, structure, modularity and moving parts of Linux as described in the existing literature. This was the drilled down to Kernel sub system which was the focus area of this study.

The full set of features of the core Kernel Linux were extracted and loaded in a relational database management system (RDBS). Each feature is supported by a module or C source code. The list of all source code files was then extracted. This constituted 100% sampling.

To narrow down the scope as defined for the study, three more configuration files were used. Two (2) header files that relate to resource and process management were identified as per the existing literature. The C source files referencing these were matched and this created a smaller set of objects to consider.

To further limit and pick key features, a Makefile which contains the build rules and compilation entries for the Kernel was referenced.

Out of above, three (3) features were selected and subjected to further review and tweaking to understand customize their behaviour. The corresponding source codes were then reviewed.

Out of above, a number of scripts were developed using bash or scripting languages to implement the features identified can be used to achieve multi-tenancy provisioning.

Keywords: Linux, kernel, cloud, technologies, virtualization, MakeFile, isolation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

1.1 Definition of Cloud Computing

Cloud computing is a computing framework that aims to allow for fast creation and set up of required computing services on demand and provide universal network access to a shared pool of isolated, definable and configurable computing resources viz servers, storage, software (*NIST: Cloud Computing Definition, 2012*).

Cloud Computing has some key characteristics, offer several service models, and support multiple deployment models (*NIST: Cloud Computing Definition, 2012*).

On-demand self-service. Ability to independently provision computing services as needed automatically by the consumer.

Broad network access. Should offer access capabilities across different devices and using different access protocols (http, https, ssh, ftp, sftp etc) over the private network connections and over the internet

Resource pooling. Implement a multi-tenant model to allow different physical and virtual resources dynamically assigned and reassigned as requested or demanded by service subscribers.

Location Transparency: The user experience should be the same regardless of the physical location of the compute resources or data centre. The subscribers should not even be aware where the resources are located.

Rapid elasticity. The cloud computing framework should allow users to increase, decrease or optimize the provisioned capacities at will or based on configured policies.

Measured service. Cloud services usage are measured using either metering consumption of resources or set quotas or use flat rates.

Hosting and access of compute resources in the Cloud is called Cloud computing (CC). The CC can be defined as a computing framework where resources host ran in remotely located servers in remote or on-premises data centres. The consumers lease or rent processing and storage resources in these environments and run their workloads in the allocated resource spaces.

The cloud providers create advanced and resilient IT infrastructures that ordinary organizations cannot match. To ensure maximum return, the cloud providers base decisions regarding locations and scale on different cost factors and demand and supply. Lately, legislation and compliance requirements also play a key role as different jurisdictions impose varying compliance and legal requirements.

Different factors drive adoption of cloud computing viz:

To achieve elasticity, the cloud platforms come with auto scaling and capacity on demand capabilities. These features enable organizations to use what is required only as opposed to massive investments that can easily be underutilized. Vendors use hypervisors to partition and isolate environments and resource allocation are based not on say physical processor cores but on units of the same.

Another key factor is an endeavour to drive down the costs. Aministrative overheads are offloaded to the service provider; Insurance, physical and cyber security costs are borne by the cloud provider

Currently, the administrators spend considerable amounts of time handling administrative tasks. In the cloud mode, Serverless Computing is achieved as Server management and administration tasks e.g. patching, routine maintenance and operational tasks are performed by the cloud vendor leaving the subscriber to focus on the core business.

Cloud platforms isubiquitous and these services are accessed over the internet from anywhere and anytime. With the level of internet penetration, any interested organization is guaranteed of internet connectivity.

There is also the option of bare metal cloud where tenants rent/lease a raw hardware platform then decide how to use it without any limitations imposed by virtualization

1.2 Study Background

Cloud computing is viewed as an evolution of virtualization which has been a technology that has been around for some time. Even organizations, have been using virtualization in on-prem environments for considerable period of time

On the other anticipated benefits of CC is a shift from server focus to non-server focus. The latter is referred to as Server-less computing. In the latter, the subscribers are relieved from the overhead of routine maintenance of the servers. Routine administration and maintenance involve such tasks as patching, optimization, indexing, purging and software upgrades. This remains elusive as the subscribers ‘own’ the virtualized environments and anything required thereon is the responsibility of the subscriber. The CSP only worries about the hardware and the hypervisor technology. The VMMs also come with other draw backs e.g. the limitation to the available capacities to predefined units of minimum configurations

An alternative solution that avoids virtualization will offer the prospect of a solution to overcome the virtualization problem.

Other key concerns and begging questions that users have to consider while evaluating cloud services are:

a. Vendor’s virtualization technology capabilities and maturity- is it heavy and chews lots of resources making it expensive for the customers. Or is it limited to which resources it can virtualize

b. Has the vendor mainstreamed virtualization technology in the product vision and public road map or it is a pedestrian product that is not evolving in tandem with other technologies
c. Does the virtualization vendor collaborate with software vendors to ensure optimal performance of the software products on virtualized as on non-virtualized environments?
d. Does the virtualization technology meet open standards and inter operability requirements?

1.3 Problem Statement

Virtualization technologies are not light weight but consume substantial resources forcing the vendors to price the cost of cloud services at marked up prices which should not be the case.

The hosted software technologies sometimes do not perform optimally on virtualization environments due to abstraction bottlenecks.

Furthermore, virtualization locks resources and forces consumers to procure idle capacity as the virtualized environments have set minimum configurations which might be higher than what a business wants for its workloads.

Linux Systems boast of key capabilities that can be harnessed to push the limits of this cross platform. It should therefore incorporate capabilities and core features that can be used to set up a bare metal multi-tenancy environment.

1.4 Multi tenancy enabling technologies

Virtualization Technology is the core of the modern Cloud computing. The success and adoption of this new innovation is based on the robustness, capabilities and features of the virtualization software. This is also a differentiating factor between the CSP services selling points.

1.4.1 Current Dominant technologies used for Environment Isolation

There are two dominant technologies used currently for environment/resource isolation. Figure 1 shows the Virtualization and Containerization component Stacks.

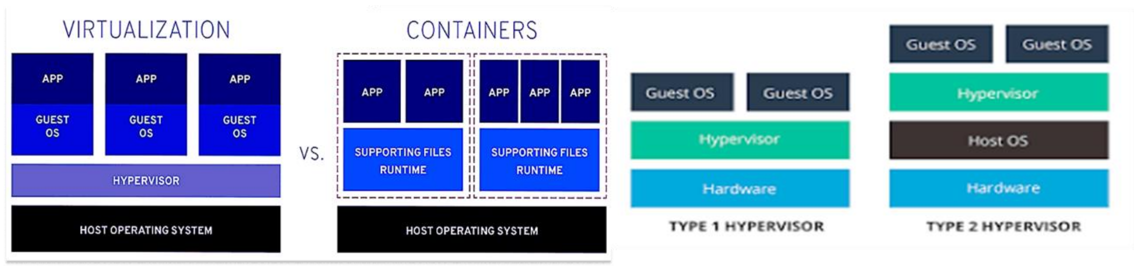


Figure 1: Virtualization and Containerization Stack

1.4.2 Alternative Potential Solution

The proposed solution avoids virtualization and Containerization and aims to implement isolation at Kernel using features already supported in the Kernel as depicted in figure 2

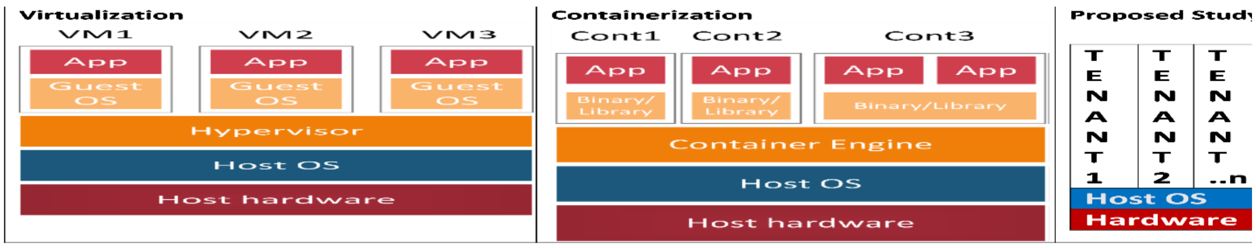


Figure 2: Alternative potential Solution

1.5 Objectives of this Study

My Research goal was to understand the existing Linux features and how they have been harnessed and identify provisioning and resource segregation features of Linux Kernel and corresponding changes or modifications in configurations to achieve a resource isolation and overcome the software virtualizers limitations

The specific objectives were:

1. To explore use of Linux Kernel based features currently for environment isolation for multi-tenancy implementation in Cloud Computing
2. To investigate how Linux based features have been used for environment separation or isolation in cloud multi tenancy set up
3. To explore a complete/integrated approach or alternative use of the Kernel features to create and isolate Multi Tenancy environments in a non-virtualized Bare Metal cloud
4. To evaluate and simulate the multi tenancy provisioning based on the proposed approach

1.6 Research Questions

To guide this Research, the following are key formulated Research.

1. Are Linux Kernel based features currently used in cloud computing to implement multi tenancy as an alternative to virtualization
2. Which and how are key Linux Kernel features implemented currently to create and isolate cloud tenancies
3. How should the selected top Kernel features be configured to achieve the isolation and create a multi tenant environments
4. What parameters can be used to assess and evaluate successful multi tenancy

1.7 Significance of the Study

One of the technology disruptors of the modern day is the cloud computing. The craze and rush with which companies are raring to jump on board and embrace this makes it an area of technology interest. The use of Information technology in foot printed in every day facet of human endeavour (Izevbizua, 2013). There is need for deep understanding, appreciation and to identify how the current features can be augmented or complemented to safeguard the consumers interests and security and guarantee value for money.

The essence of the study was to explore alternative multi tenancy provisioning that does not rely on virtualization which is an independent and resource-gobbling and limiting technology. Conclusions from the study will help improve on cloud environments isolation technologies by exploring alternative cloud services platform provisioning technology

1.8 Scope of the Study

The study shall be conducted on open-source Operating Systems. A normal PC shall be used as the host physical server to mimic high end server computer. The study shall also be limited to only two (2) Linux Kernel based features which are not currently heavily used.

1.9 Assumptions of the Study

- There are enough online materials and resources to enrich this study
- The existing technical literature is authentic
- The Open Source Operating Systems contain run time modular libraries and utilities that can be used to achieve this envisaged solution
- This research can be concluded within the set timeframe and any pending investigations can be continued by other scholars so that conclusive findings can be achieved.

CHAPTER TWO: LITERATURE REVIEW

2.1 Linux Distributions

Linux distribution is an operating system made from a software collection that includes the Linux kernel and often a package management system (*Wikipedia, Linux Distribution, 2023*).

A Linux distribution comprises a Linux kernel, an init system (such as systemd etc), GNU tools and libraries, documentation, and other software (such as IP network configuration utilities and the getty TTY setup program, among others).

The included software is free and open-source software available both in form of compiled binaries or source code form. It might also include proprietary software that is not publicly available

There are well defined rules and procedures for initiating and making changes (*Linux Online Documentation*)

The most common distributions are: Redhat (CentOS), Ununtu, Fedora etc

2.2 Feature Representation

Features of Linux Kernel are specified in the build rules contained in files known as Makefiles (*Leonardo Passos et. al: A Study of Feature Scattering in the Linux Kernel)*. Furthermore, the same can be deduced from the C preprocessor directiveswhich are used to control full Kernel compilation. There are varuous Linux Makefiles (*Corbet et.al:Kernel makefile documentation, 2003*)

2.3 Kernel Organization

The kernel is primarily organized around the core services it provides: process handling, memory management, file system management, network access and the drivers for the hardware. These areas correspond to the kernel source directories, kernel, mm, fs, net and drivers respectively (*Torvalds, Linus: Linux the Portable System, 1997)*

2.4 Structure of Linux System and architecture components

The following are the constituent parts of Linux Operating System (*Tutorial Point: https://www.tutorialspoint.com/operating_system*). The key components are depicted in figure 3:

i. **Bootloader.** After computer is powered on**,** the BIOS burned into memory chip locates a small program that is responsible for booting. This small program is called the boot loader; it manages the boot process and is the one that initiates the starting the Linux kernel. The boot loader is located in the first block of about 512KB

ii. **Kernel.** This is the engine of the OS; it handles process management (starting, scheduling, prioritization, termination etc), Input and Output (IO), devices and other key sub systems . The Linux kernel provides the interface to the hardware resources through various mechanisms

iii. **Init system or Systemd etc.** After Kernel has been loaded to RAM, it locates the first program to load. This is called Init or SystemD depending on the Linux version**.** Other programs are then started by the initial processes

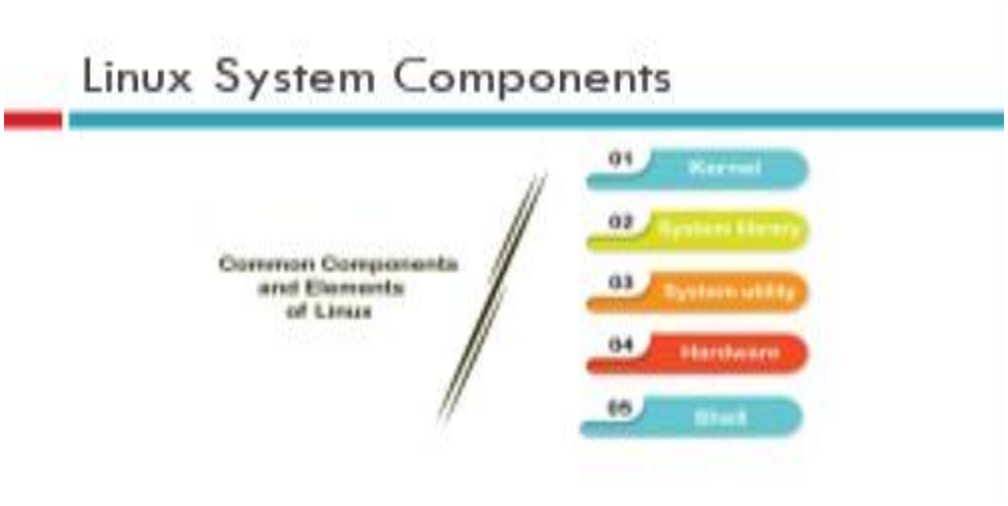


Figure 3: Linux Kernel Components

2.4.1 Linux OS Boot Process

Linux boot process entails a number of steps since the powering on and loading of BIOS from the chip as shown in figure 4 (*free code camp: https://www.freecodecamp.org/news/the-linux-booting-process-6-steps-described-in-detail/*)

BIOS loads and executes the Master Boot Record which is located in the first sector of bootable disk. The Grub is then loaded; Grub is used to specify which image to load). The selected option then locates the corresponding initial system for the selected image.

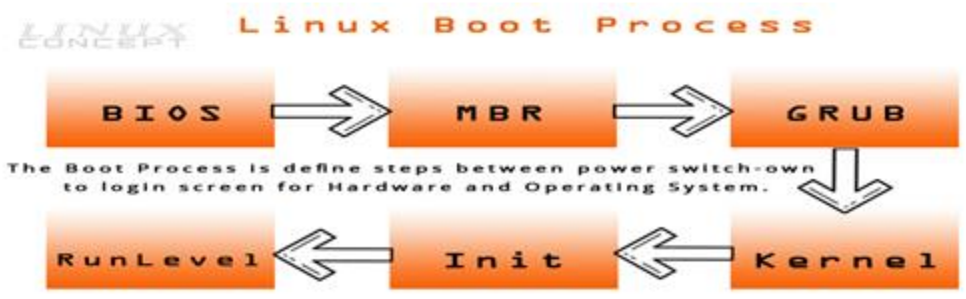

LINUX CONCEPT
Linux Boot Process
BIOS
MBR
GRUB
The Boot Process is define steps between power switch-own to login screen for Hardware and Operating System.
RunLevel
Init
Kernel

Figure 4: Linux Boort Process

2.5 Multi-Tenancy Definitions

A tenant is defined as an independent entity that signs up for cloud services offered by CSP. The compartmentalizing the available resources to accommodate different organizations or entities is what is known as Multitenancy.

The individual tenants share different classes of resources viz infrastructure, applications or software based on their individual needs. CSP must provide and implement appropriate mechanisms to isolate and separate the different tenants wholly or completely so that each is not aware of the other tenants.

This also avoids issues of noisy neighbors and security lapses as both the victim and potential attacker could be residing in adjacent environment

Multi-Tenancy utilizes different underlying technologies to share and allocate resources to the different tenants. Multi-Tenancy is engendered when multiple compute environments assigned to different customers are provisioned on the same resource in complete separation that ensures zero data leakage and full control by each tenant. Traditionally, multitenancy was largely defined as a result of Virtualization, but new innovations also achieve resource isolation without virtualization. The equation that virtualization and resource sharing result in multi tenancy holds partially (*Hussain AlJahdali et.al: Virtua;ization, 2014*)

2.5.1 Layered Multi-tenancy

At different Cloud Services layers, multi tenancy takes different forms and definitions. These are described below:

In Software Layer -SaaS, the subscribers have no visibility of the underlying and supporting hardware or software but can only access and use the software signed up for. When two or multiple subscribers access the same service with no impact on the other tenant, then multi tenancy is so created on this software as more than subscriber has been provisioned (*IEEE: 2nd International Conference on Cloud Computing ,2010*).

For its part, in Infrastructure Layer (IaaS), Customers are responsible for monitoring and maintaining the supporting provisioned infrastructure. At this layer, when multiple compute instances are created and allocated to different subscribers then multi tenancy has been created on the same physical hardware *(IEEE: 2nd International Conference on Cloud Computing, 2010).*

2.5.2 Multi-Tenancy Characteristics

There are different forms of cloud Multi Tenancy *(Isaac Odun-Ayo et.al: Multi Tenancy, 2017*). Hardware Resources Sharing relates to different customers being allocated hardware resources to configure and load and process their respective workloads. The resources allocated here are hardware in nature.

On the other hand, the high degree of Configurability is a characteristic that is required for applications to support multi tenancy, the applications must be highly configurable and customizable to allow for localization and domestication and branding as may be required.

Multiplicity of instances refers to multiple instances creation must be supported and these multiple instances must be enjoined to work seamlessly

2.6 Virtualization

Virtualization refers to slicing of a given physical resource to generate multiple smaller 'copies' (*IBM: IBM systems virtualization‖, 2005*). Virtualization technologies are specialized and can be set up on commodity hardware (Type I) or on the host OS (Type II). This technology is the most dominant technology today and is synonymous with hardware resource slicing.

2.6.1 Virtualization Origins

The initial approach advanced by General Electric (GE) was to explore an approach that involved time-sharing computing (*Conroy et al: Virtualization, 2018*). With growing interest, IBM started projects around time sharing computing development as well.

2.6.2 Virtualization Technology

There is no universally adopted definition of Virtualization. In most of the reviewed literature and reference materials, there are there are large differences about the definition of virtualization (Luo, Yang & Ma, 2011). Instead of defining virtualization in terms of what it is, most definitions relate to virtualization types. To date no strict standards and definitions were created (Luo et al. (2011).

A more general definition based on existing literature and the objective has been coined as follows.

"*Virtualization abstracts underlying hardware; it has become a de facto standard for implementation of rresource isolation and server consolidation*" (Li & Kanso et. al: *Virtualization, 2015*).

Extending the above definition, below are key characteristics of virtualization: Partitioning entails slicing one physical system into multiple smaller or sub systems that can be used to host different OSs and to contain other resources

On the other hand, Isolation is the ability to share one OS or run guest OS or only required run times in isolated environments. This additionally improves security.

2.6.3 Virtualization Architecture

There are two (2) main types of server Virtualizations technologies as indicated in figure 2.4.3. Each type of virtualization consists of various components as depicted. The virtualization types are categorized into Types I and II.

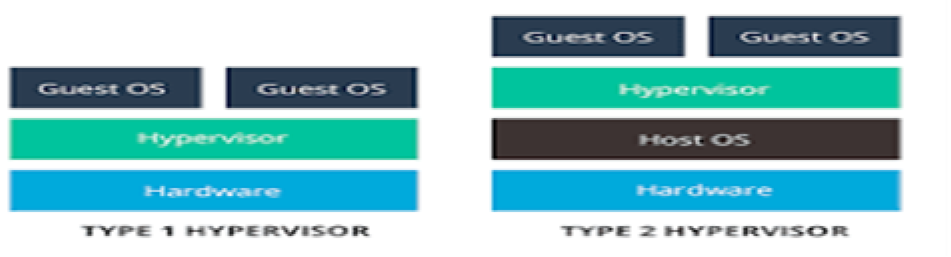


Figure 5: Virtualization Types I & II

Type I

In this type, a special software known as Hypervisor or VMM is installed on the bare metal; This special software is then used to create virtual servers; each of which runs an independent OS known as guest OS. The VMM abstracts the underlying hardware for the VMs created.

Type II

Unlike Type I, in Type II architecture, the hypervisor runs atop an OS that is installed on the physical hardware.

2.6.4 Virtualization Advantages

Virtualization provides several advantages (*Carroll, Kotzé, and Van der Merwe: Virtualization Advantages, 2010*). The main virtualization benefits include reduction in total cost of Ownership (TCO), server consolidation, and server utilization (*Carroll et al: virtualization benefits, 2010*).

VMMs also support other technologies that include that address business continuity in terms of disaster recovery and services restoration. These capabilities include VM motion to another site.

2.6.5 Virtualization challenges

There are inherent security vulnerabilities in virtualization categorized as virtual machine, hypervisor, virtual infrastructure and virtual network threats (*Timur Mirzoev et. al: Securing Virtualized Datacenters, 2010*).

The virtual machine threat manifests while processing status of virtual machine, software updates, resource contention, patching and use. Hypervisor threat rivets Virtual-Machine-Based Rootkit (VMBR) attack and Blue Pill Attack where hypervisor plays the vital role of Virtualization (*J. Rutkowska et. al: Subverting Vista Kernel For Fun and Profit, Aug 2006*).

Other challenges include hypervisor introduces a layer of resource overhead and pose constraints in capabilities such that they do not meet all the requirements

2.6.6 Virtualization and Cloud Computing

The cloud domain leverages heavily on Virtualization. Virtualization is therefore regarded as a core component in cloud computing (*Carroll et al: Virtualization, 2011*).

Virtualization can be applied on different cloud resources viz Storage, applications, networks, Servers and compute. It is virtually supported across all the cloud layers. The virtualization has enabled creation of multiple cloud services like SaaS, PaaS, IaaS etc

2.7 Containerization

Containerization was the next innovation after virtualization. Containerization is about creating resource environments or compartments. Software images are run in the containers. Containers harness on isolation within the same OS as opposed to VMs that are characterized by Machine Level Isolation.

Therefore, containers are lighter and easier to create, maintain and scale as opposed to VMs. A small compute platform can host multiple containers and so many VMs only. A case in point is a laptop. There is a limit to how many VMs can be spinned because of VM minimum configurations while it is possible to spin greater number of containers.

2.7.1 Containerization Technology

This is an OS level isolation method for deploying and running distributed applications in compartments in one VM or physical machine. The containers are ran on a single control host and access a single kernel host OS (*Kaur et al: Containerization, 2018*). Containers harness Linux kernel features to manage isolation between applications (*Li & Kanso et al: Kernel Isolation Features, 2015*). Each container is name spaced and is allocated its own process and network and other namespaces to enable it function independently. A container is a therefore a set of processes isolated from other processes in other containers (*Li & Kanso et al: Kernel Isolation Features, 2015*). Containers are made up of several components as depicted in figure 6

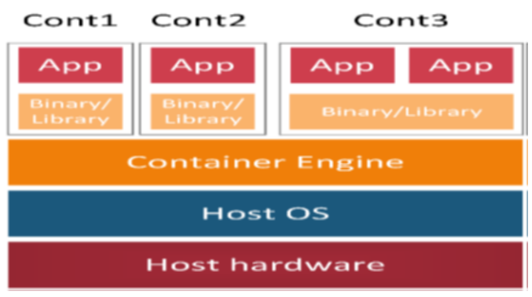


Figure 6: Containerization Components

2.7.2 Container Boot Process

Container booting up process entails application of some defined configuration and loading of listed dependencies and required files downloaded from the defined repository. After this initialization, the policies and rules defined in cgroup and namespaces are applied (*Will Wang et. al: Demystifying Containers 101, 2018*)

According to the container, only the files and binaries specified in the image are the only visible files and is blind to any other file in the host.

2.7.3 Containerization Challenges

Containers are known to suffer security challenges (*David Antonio et al: OS-level virtualization with Linux Containers, 2020*). These include Security boundary not being fully defined owing to limited namespacing features. The container engine consumes resources. The overlapping of process identification numbers across namespaces and permissions override arising from the same user being assigned to different namespaces are some of the other challenges that bedevil containers

2.8 Cloud Delivery Models

The cloud services are delivered in various forms (*IBM: Topics: IaaS, PaaS & SaaS*). All cloud services are accessed online vide web browser or other remote connectivity tools depending on what CSP provisions. These services are priced differently. The tariffs and costs are preset at a flat rate or are paid per use. These are:

2.8.1 Provision of Infrastructure

The hardware, Input and Output (I/O), devices and network resources and capabilities and services are grouped and classified as Infrastructure-as-a-Service ("IaaS"). The components of this ar:

The Facilities are brick and mortar facilities that house the computing facilities. These also include power, cooling and all environmental controls and monitoring tools. It also includes physical access devices and controls

The compute are the actual servers providing processing, memory and control features. The compute also include resources used control devices like load balances to aid auto scaling horizontally etc

Subscribers are provided with network capabilities that enable them to set up virtual networks, define routing and firewall rules. All required network capabilities are contained here; the se include IP address pools, sub netting capabilities etc

Most CSPs provide all forms of storage viz - File, Block and Object Storage. Performance challenges bedevil Block and file with scaling. Consequently, object storage is more versatile and dynamic; data can be accessed through various protocols viz HTTP, sftp etc

2.8.2 Bare Metal as a Service (BmaaS)

BMaaS facilitates lower level of control than IaaS though provisioning and consumption is similar to IaaS.

However, BMaaS does not leverage virtualization to provision virtualized compute, network, and storage (*IBM: Topics: IaaS, PaaS & SaaS*). BmaaS allocates the hardware directly allowing near total control of the hardware; since this is not affected by the virtualization issues, it offers better potential performance which is key for High Performance Computing and heavy workloads.

But BmaaS is limited in mechanisms to achieve scalability and elasticity that is so readily supported in IaaS. When mechanisms are invented to allow for scalability and elasticity, BmaaS will supercede IaaS performance-wise

2.8.3 Provision of Products

In this model, the CSP provides products and applications or resources that be used to run or process user workloads. The products are cross cutting and range from database tools, report analytics engines, middle ware, development frameworks; that is everything that ca be used to build and test end user applications

It also provides the software build integration and delivery platforms and versioning. In short anything that the customer would ordinarily run in on-prem environment

2.8.4 Provision of Application/Software Services

Under this set up, the CSP provides ready to use applications and software that only require the customer to open an account and configure and start using.

The customers access this vide the web browser and are used to process transactions end to end the way one would do in on-prem using the locally installed and set up software environments

2.9 Cloud Enabling Technologies

The modern clouds are an ecosystem of multiple components that together provide seamless access to the purchased cloud services (*Thomas Erl et. al: Cloud Computing Concepts, Technology and Architecture, 2013*)

. The key components are:

2.9.1 The Network or Inter-Networks (Internet)

The network connectivity provides the access layer and consists of the local area and wide area networks and connections.

The LAN/WAN architecture consists of networking devices – routers, switches, cabling and security devices.

For the cloud to be accessible, it should be hooked up onto a network. There are different networking requirements for different deployments:

i. **Private Cloud**
   - The cloud and corporate network are inter-connected vide LAN, VLANs or VPN;
   - The CSP connects through Internet or via VPN

   The cloud networks support Software Defined Networks and devices such as load balancers, DNS Servers etc

ii. **Public Cloud, Virtual and Community Cloud**
   - The primary connection is over the internet.
   - The connectivity involves Company networks, ISP, internet backbone providers, Cloud provider backbone networks
   - For Security considerations, some enterprises have opted for Site to Site Virtual Private Networks(VPNs) using IPsec protocol to provision secure network tunnels

iii. **Hybrid Cloud**
   - The connections are a combination of both Private and Public connectivity types.

2.9.2 Computer and Data Centre Technologies

The key technologies that enable power cloud services are:

i. Data Centre Infrastructure
   These are the housing facilities, power, cooling and server closets
ii. Automation
   The cloud platforms provide dashboards and other user screens for orchestrating and undertaking user and administrative tasks e.g. provisioning, configurations, patching, monitoring and alerting solutions
iii. Remote Operation and Management
   The cloud environments are not located within physical reach. These platforms provide capabilities and utilities that allow remote access over different protocols

iv. Computing Hardware
   These real computer servers that host resources required to process user wworkloads. They are set up in redundancy configurations for high-availability and uptime. The cloud environments are available above 99.9999%

v. Storage Hardware

Storage services are provided through different technologies. Some of the technologies involved include: Hard Disk Arrays (RAID), Hot Swappable Disks and replication mechanisms

vi. Network Hardware
   - The network components range from networking devices and storage connectivity devices.
   - These are multi-layered and clustered, or load balanced to ensure high availability

2.9.3 Virtualization Technology

As described above, virtualization technology is the main platform and engine that is used to create virtual or logical environments for assignment to different customers. This is widely used in modern computing platforms.

2.9.4 Presentation and accessibility - Web browser based and Command Line Interface (CLI) interfaces

i. Web Console & APIs

a. Web consoles and interfaces accessed via http or https protocols are main channels for accessing cloud services.

b. APIs are also supported to allow for automation

c. User authentication and authorization can be configured in variety of ways including single or multiple factors and Identity and Access policies

ii. Shell/CLI/CONSOLE

a. Privileged back-end access is facilitated through secure protocols like Secure Shell Protocol over a secure tunnel that has been configured to support the protocols and other encryption requirements

Different security access factors can be configured: key-based, single factor etc

2.10 Isolation Features in Linux

2.10.1 Chroot ()

As Linux distributions gained popularity, a need to isolate or create partitions at different levels gained momentum.

The chroot() system call was the first attempt and it was mainly intended to create a pseudo root system that mimics the primary root file system as represented in Figures 7 and 8. Chroot changes the root filesystem (*Linux: Linux Manual, 2023*)

The change root command abbreviated as Chroot creates another root file system inside the main root system. Processes invoked and ran within the pseudo root system can only see what is contained in the virtual root and not the main root system.

By default, Linux file system is set up hierarchically as follows:

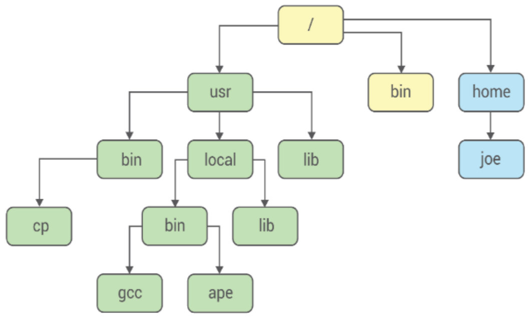


Figure 7: Linux File System Default Structure

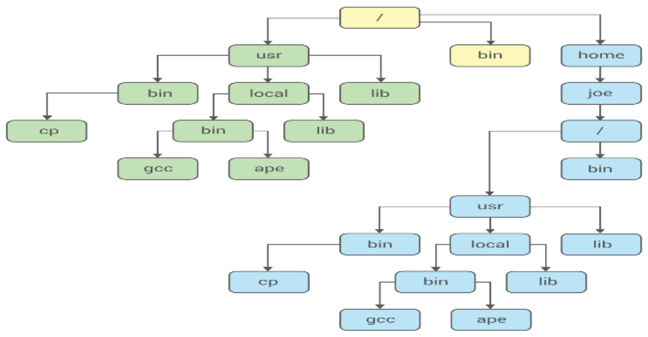


Figure 8: Linux File System with Pseudo Root

Chroot syscall generates a secondary root directory, which has a similar structure that mimics the original root as depicted below:

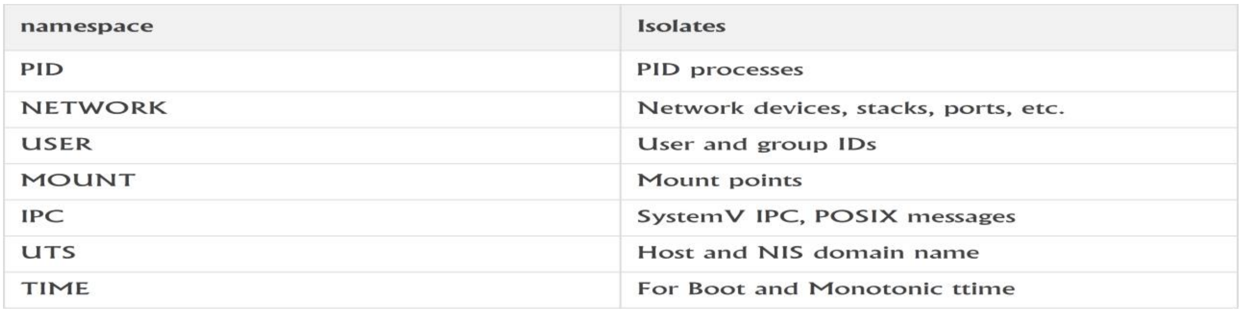
2.10.2 The Namespace Mechanism

In Chroot, all executables and dependencies need to be copied to the new directory. The pseudo root directory cannot access files and executables in the main root. This requires all the run times to be copied an duplicated in the new root. This creates inconsistencies, lack of synchronization and security issues as the there is no synching mechanism. These are major drawbacks of this approach.

This necessitated continued research which resulted in Namespaces feature being introduced as an inline Linux kernel feature to isolate processes from one another. Linux namespaces enables concurrent running of multiple applications distinctly i.e. with no cross interference or data leakages(*Toptal: A Tutorial for Isolating YourSystem with Linux Namespaces***)**

Linux kernel namespaces are defined in nsproxy data structure. The structure for this is defined in nsproxy.h. This data structure stores details of each namespace where the current process is assigned. The header file is located in /../nsproxy.h. Table 1 shows the supported namespaces currently

Table 1: Supported Namespaces in Linux



namespace	Isolates
PID	PID processes
NETWORK	Network devices, stacks, ports, etc.
USER	User and group IDs
MOUNT	Mount points
IPC	SystemV IPC, POSIX messages
UTS	Host and NIS domain name
TIME	For Boot and Monotonic ttime

2.10.3 Process Identification (PID): Process Isolation

To enable introduction of process namespace feature, Linux process trees were nested and multi-levelled as opposed to the original one process tree. The current version of Linux allows branching out of a process tree to multiple branches and leafs as depicted in Figure 9

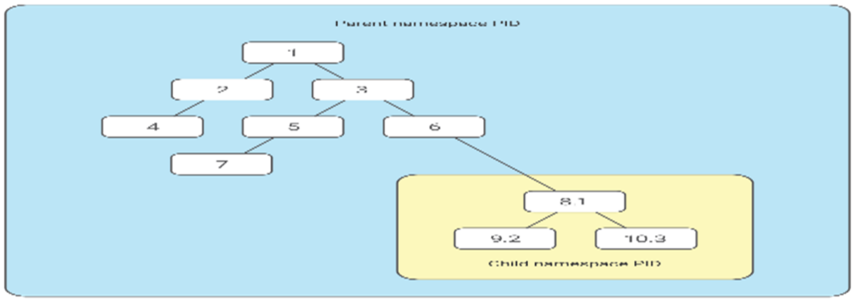


Figure 9: Linux Process Tree

Linux supports CLONE and UNSHARE to create new namespace PIDs with appropriate flags NEWPI

2.10.4 Process Identification (PID) Namespace & PID Generation

2.10.4.1 Process Table

The Linux kernel keeps track of processes initiated/created/spawned by storing relevant information and status of each process using process table. The record inserted has the following information;

- PID
- Parent Process
- Environment Variables
- Elapsed Time
- Status – one of D (Uninterruptible), R (Running), S (Sleeping), T (Stopped), or Z (Zombie)
- Memory Usage

2.10.4.2 Process ID Generation

Linux allocates process IDs in sequence, starting at 0 and assigns with the range from zero to a maximum limit. The maximum limit for PIDs is a configurable parameter

When a process is launched, a PID for the process is generated to allow uniquely identifying it. **This is done simply by incrementing the current highest PID by 1.**

2.10.4.3 System Calls to Create Processes in Linux

Linux provides a number of system calls to spawn new process as described below:

2.10.4.3.1 Fork()

Fork spawns another replica process called child process.

When Fork is invoked, it returns 3 values: A value less than zero – indicates failure to create child process; Value=0, indicates the process was successful. The child PID can be obtained by calling getpid() function while Value >0 – this is the process ID for the parent process

2.10.4.3.2 The Clone Method

The clone function uses the child stack argument, which specifies where in the stack the child process is. The child and parent processes can share memory. However, the child process cannot be executed in the same stack as the parent process.

2.10.5 NET: Network Isolation

Network namespace is used to isolate network groups or network namespaces; processes are added to the network namespace by invoking the Network FLAGs in the clone or unshare system calls

2.10.6 MOUNT: File System Isolation

MOUNT namespaces are used to create mount points for file systems that can be associated with a given processes

2.10.7 Other Namespaces

Linux supports other namespaces as well. These are: UID, IPC, and PTS (*Linux: Linux Manual*).

2.10.8 Resource Utilization Capping and Limiting Kernel Features- CGroups

Resource usage in Linux can be controlled using a Linux Kernel feature known as Control group (cgroup). Currently, cgroups controls mainly compute, I/O devices and network resources

Cgroups has some key features:

Resource limits – This is where the usage limits are set.

Prioritization – This allows for setting of priorities for access and resource usages.

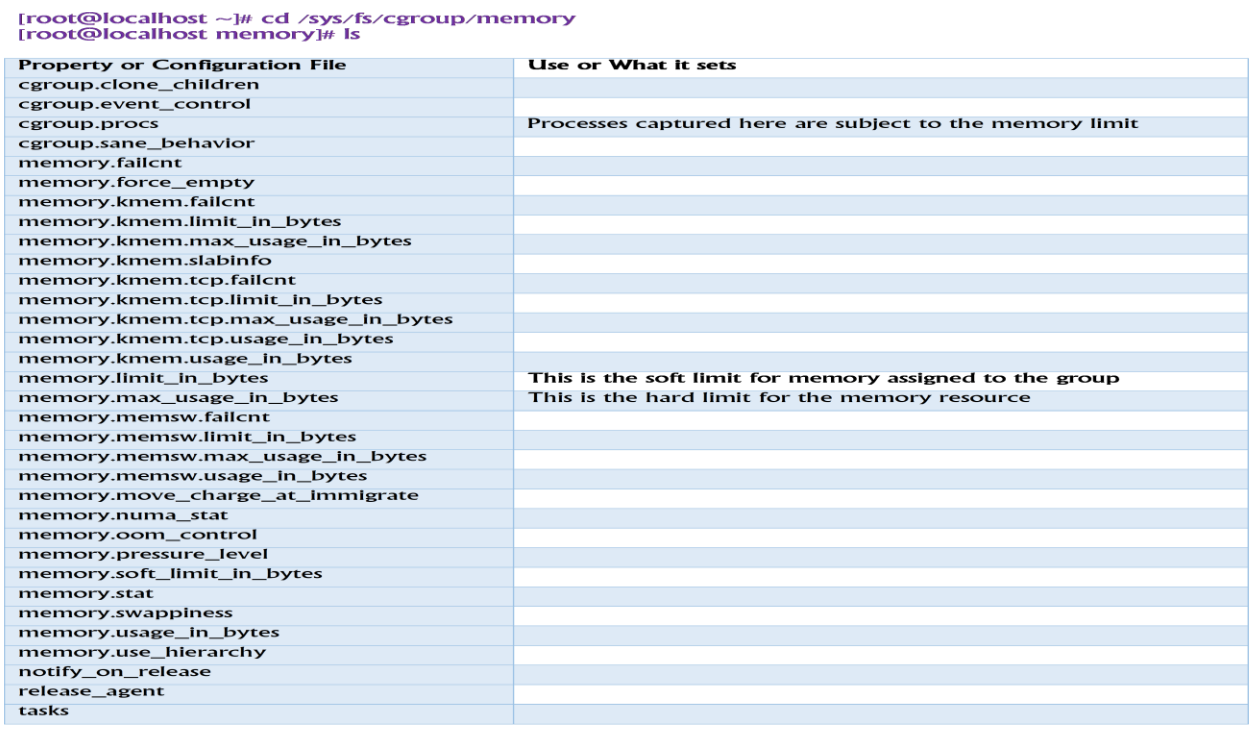
Accounting – Usage of resources are tracked using this feature

Control – Used to manage processes in a cgroup and to change their status as needed

2.10.8.1 Capping Memory using cGroups

The following Properties can be set for memory resource in the control group as depicted in table 2. These are set in the cgroups file system which is mounted on /../sys/fs/cgroup/memory/

Table 2: Linux cGroup Memory capping attributes



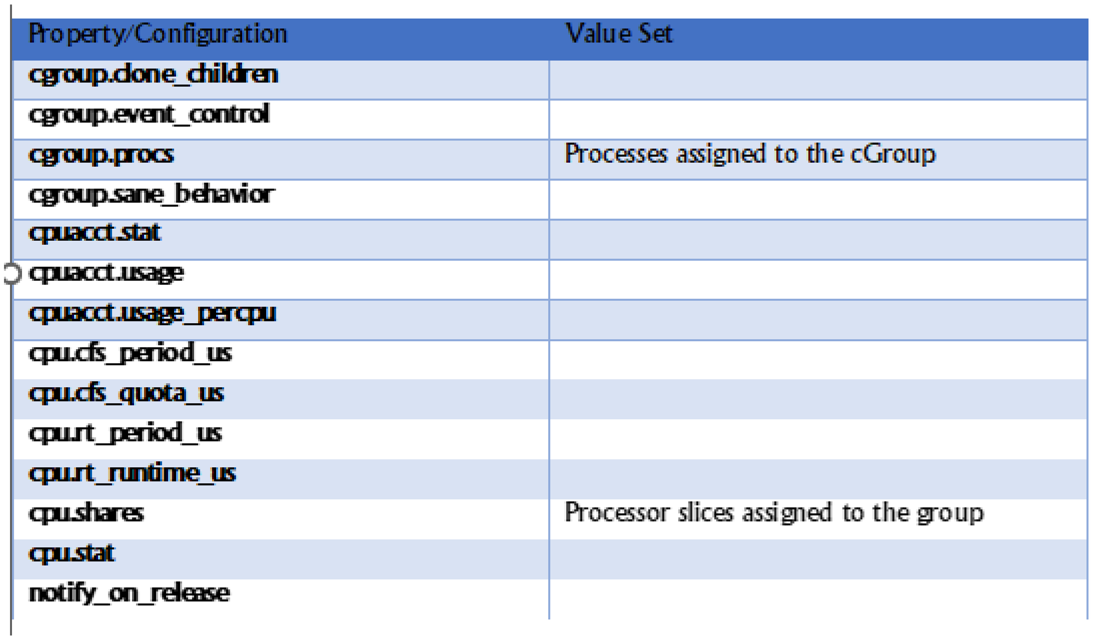
```
[root@localhost ~]# cd /sys/fs/cgroup/memory
[root@localhost memory]# ls
```

Property or Configuration File	Use or What it sets
cgroup.clone_children	
cgroup.event_control	
cgroup.procs	Processes captured here are subject to the memory limit
cgroup.sane_behavior	
memory.failcnt	
memory.force_empty	
memory.kmem.failcnt	
memory.kmem.limit_in_bytes	
memory.kmem.max_usage_in_bytes	
memory.kmem.slabinfo	
memory.kmem.tcp.failcnt	
memory.kmem.tcp.limit_in_bytes	
memory.kmem.tcp.max_usage_in_bytes	
memory.kmem.tcp.usage_in_bytes	
memory.kmem.usage_in_bytes	
memory.limit_in_bytes	This is the soft limit for memory assigned to the group
memory.max_usage_in_bytes	This is the hard limit for the memory resource
memory.memsw.failcnt	
memory.memsw.limit_in_bytes	
memory.memsw.max_usage_in_bytes	
memory.memsw.usage_in_bytes	
memory.move_charge_at_immigrate	
memory.numa_stat	
memory.oom_control	
memory.pressure_level	
memory.soft_limit_in_bytes	
memory.stat	
memory.swappiness	
memory.usage_in_bytes	
memory.use_hierarchy	
notify_on_release	
release_agent	
tasks	

2.10.8.2 Capping Processor usage limits using CGroups

Similarly, the CGroup provides a mechanism that supports limiting usage of CPU by processes. CPU limit can be set as a soft or hard limit, or both. The mount point for this is sys file system but sub directory known as cpu. The attributes are shown in table 3. These are set in the cgroups file system which is mounted on /../sys/fs/cgroup/cpu/

Table 3: Processor capping attributes Supported in cGroups



Property/Configuration	Value Set
cgroup.clone_children	
cgroup.event_control	
cgroup.procs	Processes assigned to the cGroup
cgroup.sane_behavior	
cpuacct.stat	
cpuacct.usage	
cpuacct.usage_percpu	
cpu.cfs_period_us	
cpu.cfs_quota_us	
cpu.rt_period_us	
cpu.rt_runtime_us	
cpu.shares	Processor slices assigned to the group
cpu.stat	
notify_on_release	

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Research Method

A disciplined procedure used to collect relevant data and information that will guide the research process is presented here. Methodology is the framework used to assemble, plan and direct the research process (*Oates et. al: Research Methodology, 2005*)

3.2 Research Design

The research design for this study will be based on exploratory research design.

The study shall comprises high level review and test of Linux Kernel Features that have capabilities for access, process, memory and file system isolation as per the Kernel organization by referencing literature on Linux.

Specifically, this shall involve identifying and testing relevant commands and currently supported configurations for each feature.

This shall be complemented by a review and analysis of relevant source code/files for the candidate Kernel features.

Questions shall be posed to Linux community and if promising responses are received and if this topic appears to be a topic of interest, tailored posts shall be posted to the community.

Cloud Providers market surbey based on online materials shall be used to help answer the first research question.

3.3 Research Sampling

Linux is a wide OS. The analysis of Kernel features and underlying source code or configs shall be focused on those features that are contained in Kernel and which support process and memory management.

The sample shall therefore start with the entire set of Kernel contents then drill down to the objects that relate to process and memory management.

Three of these features shall be selected for further analysis

3.4 Data Collection

The data to be collected is mainly the entire Linux code base. This shall involve extraction of the Linux code structure and code base from GitHub versioning System.

The code base shall be set up on the research and study environment and kernel sub systems was identified based on the downloaded structure or based on the literature review.

Information collected from Literature review was used to guide on selection of features used to isolate resources. The C preprocessor directives namely the header files and which source code references those header files was also used to further triangulate the selection made from the code structure.

The results collected were captured in a table for subsequent analysis.

The capabilities of the different system calls was assessed by executing the different commands that relate to the features identified as being used to manage the resources identified above. The supporting configs for these commands were tweaked to achieve a different behaviour than the default behaviour.

3.5 Raw Data analysis

To segment the Linux features, the data collected was captured in a spreadsheet and RDBMS table. The table contained different fields. The results were recorded in a matrix indicating each feature against support for the isolation mechanisms

Filtering through an SQL query was executed and the resultant result set only showed the items that fall within the scope of interest.

For system calls and commands executed, the output of the commands and optional configurations that could be set were reviewed on screen and what was relevant to support the study was captured. To address noted limitations, scripts were developed to test or modify the system behavior. These were included in the results.

Source code and header files were analyzed and code improvements to achieve new behaviour were tested.

3.6 Inputs from the Linux Open-Source Development Community

No positive input was received from the community. The initial responses evidently showed that this topic appeared farfetched.

3.7 Case Studies on how cloud providers use alternative multi tenancy methods

The three major CSPs were sampled based on market share information published online and based on industry knowledge. The three sampled were : AWS, Microsoft and Google

3.8 System Test and Validation

Testing multi tenancy comprises testing if the proposed mechanism can allocate, cap and adjust resources in line with characteristics of metering, elasticity and accounting. The summary of tests are shown on table 4.

The tenancies must be separated and another parameter used was to test if based on proposed approach, there was a way to create and define tenancies.
The new approach will be achieved either by tweaking or modification of configurations or system parameters. Evidence of these changes will be key to demonstrate the new concept.

The processes separation and identification is a key feature that will define if the new approach works. This is therefore a critical evaluation parameter.

Cloud computing is known to be ubiquitous. Accessibility of the test environment through the web (http or https) and secure shell (ssh) protocols test confirms that this feature has been achieved and the new approach has markings of cloud computing access capabilities.

Table 4: System Tests and Evaluation

Test	**Expected Result/Indicators**
Resource allocation and capping/limitations and resizing	• Resource allocation matrix per tenant • Resource limits
Process isolation and restricted visibility	• Root should see all processes • Individual users should see own processes only
Modified System Parameters	Old Vs. New Configs matrix/listing
Remote administration/commanding	• Access vide ubiquitous web • Commands executed remotely applied on the host

CHAPTER 4: RESULTS, FINDINGS & DISCUSSIONS

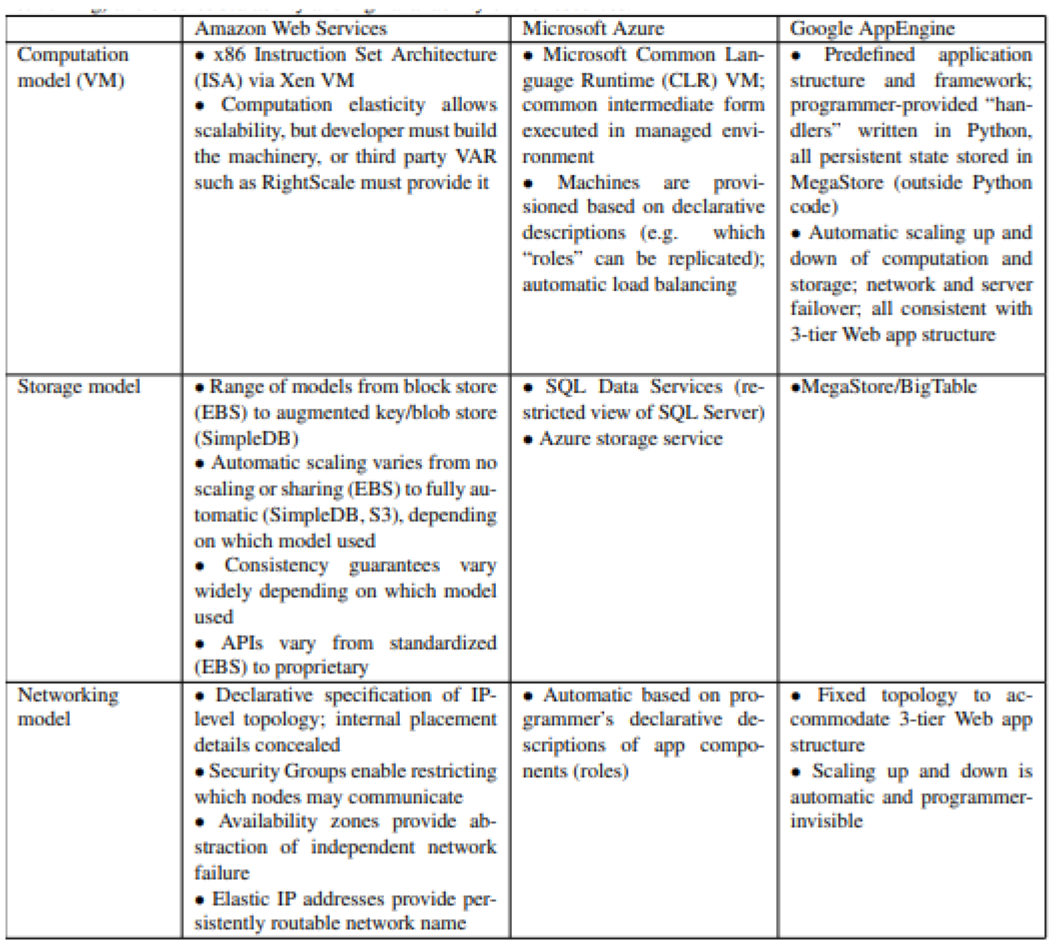
4.1 Research Objective /Question 1

"Are Linux Kernel based features currently used in cloud computing to implement multi tenancy as an alternative to virtualization"

4.1.1 Sampling the Cloud Providers - Top 3 Cloud Providers by Market Share

According to Survey undertaken by Statista findings published in April 2023 are shown in Appendix I, 80% of the cloud environments are controlled by three vendors.

Table 5: Top 3 CSPs



	Amazon Web Services	Microsoft Azure	Google AppEngine
Computation model (VM)	• x86 Instruction Set Architecture (ISA) via Xen VM • Computation elasticity allows scalability, but developer must build the machinery, or third party VAR such as RightScale must provide it	• Microsoft Common Language Runtime (CLR) VM; common intermediate form executed in managed environment • Machines are provisioned based on declarative descriptions (e.g. which "roles" can be replicated); automatic load balancing	• Predefined application structure and framework; programmer-provided "handlers" written in Python, all persistent state stored in MegaStore (outside Python code) • Automatic scaling up and down of computation and storage; network and server failover; all consistent with 3-tier Web app structure
Storage model	• Range of models from block store (EBS) to augmented key/blob store (SimpleDB) • Automatic scaling varies from no scaling or sharing (EBS) to fully automatic (SimpleDB, S3), depending on which model used • Consistency guarantees vary widely depending on which model used • APIs vary from standardized (EBS) to proprietary	• SQL Data Services (restricted view of SQL Server) • Azure storage service	•MegaStore/BigTable
Networking model	• Declarative specification of IP-level topology; internal placement details concealed • Security Groups enable restricting which nodes may communicate • Availability zones provide abstraction of independent network failure • Elastic IP addresses provide persistently routable network name	• Automatic based on programmer's declarative descriptions of app components (roles)	• Fixed topology to accommodate 3-tier Web app structure • Scaling up and down is automatic and programmer-invisible

4.2 Research Objective/Question 2

"Which and how are key Linux Kernel features implemented currently to create and isolate cloud tenancies"

Containerization utilizes a number of Linux Kernel Features to provision multiple application tenancies. kernel itself. The key ones are Linux namespaces which are inlined in Linux starting with version 2.6. that was released before 2010

Namespaces, cgroups, seccomp, and SELinux are the Linux technologies that make up the foundations of building and running a container process as depicted in figure 10.

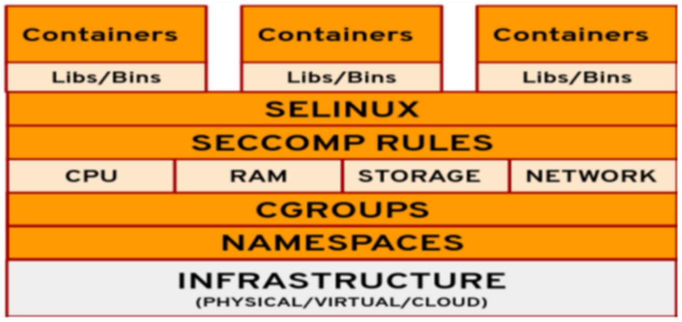
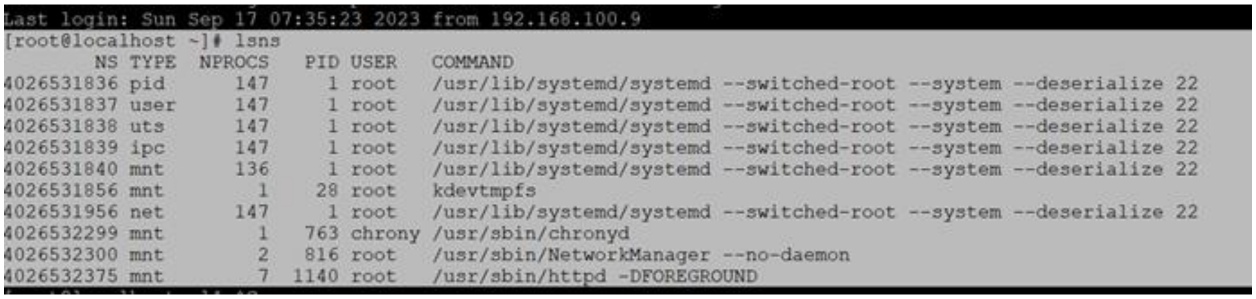


Figure 10: Linux Features used in Containers

4.2.1 Namespaces

The currently supported namespaces are as listed in Figure 11. The system call used to display these is lsns.



```
Last login: Sun Sep 17 07:35:23 2023 from 192.168.100.9
[root@localhost ~]# lsns
        NS TYPE   NPROCS   PID USER   COMMAND
4026531836 pid       147     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
4026531837 user      147     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
4026531838 uts       147     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
4026531839 ipc       147     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
4026531840 mnt       136     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
4026531856 mnt         1    28 root   kdevtmpfs
4026531956 net       147     1 root   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
4026532299 mnt         1   763 chrony /usr/sbin/chronyd
4026532300 mnt         2   816 root   /usr/sbin/NetworkManager --no-daemon
4026532375 mnt         7  1140 root   /usr/sbin/httpd -DFOREGROUND
```

Figure 11: Supported Linux Namespaces

4.2.2 Control groups (cgroups)

Containers harness cGroups to limit resources assigned to a container as highlighted in Tables 2.

4.3 Research Objective/Question 3

"How should the selected top Kernel feature be configured to achieve the isolation and create a multi-tenant environment"

4.3.1 Exploration of Linux Kernel Features

To undertake this study, different resources were used.

4.3.1.1 Exploration Environment Tools

The following tools/platformslisted in table 4 were used for this study:

Table 6: Research Tools

Software	**Purpose**
NASM (Netwide Assembler)	Assembler
QEMU	Quick Emulator
CentOS	Operating System, Source Code
Phyton	Scripting
Node	Node
C & GCC	C Compiler

4.3.2 Identification of Linux Kernel Features

The below Kernel features that support access and management of resources are potentially candidates for isolation implementations. The features are denoted with a 'Y' in the last column.

4.3.2.1 Extract Complete Linux Sub Systems & Code Base from Git [38]

The first step used was to extract the complete Linux Source Code from the version control system, Git. The structure is shown on table 7

Table 7: Linux Kernel Sub Systems

Linux Version	Feature or Feature Group
linux-6.2.9	usr
linux-6.2.9	virt
linux-6.2.9	.clang-format
linux-6.2.9	.cocciconfig
linux-6.2.9	.get_maintainer.ignore
linux-6.2.9	.gitattributes
linux-6.2.9	.gitignore
linux-6.2.9	.mailmap
linux-6.2.9	.rustfmt.toml
linux-6.2.9	COPYING
linux-6.2.9	CREDITS
linux-6.2.9	Kbuild
linux-6.2.9	Kconfig
linux-6.2.9	MAINTAINERS
linux-6.2.9	Makefile
linux-6.2.9	README
linux-6.2.9	arch
linux-6.2.9	block
linux-6.2.9	certs
linux-6.2.9	crypto
linux-6.2.9	Documentation
linux-6.2.9	drivers
linux-6.2.9	fs
linux-6.2.9	include
linux-6.2.9	init
linux-6.2.9	io_uring
linux-6.2.9	ipc
linux-6.2.9	kernel
linux-6.2.9	lib
linux-6.2.9	LICENSES
linux-6.2.9	mm
linux-6.2.9	net
linux-6.2.9	rust
linux-6.2.9	samples
linux-6.2.9	scripts
linux-6.2.9	security
linux-6.2.9	sound
linux-6.2.9	tools

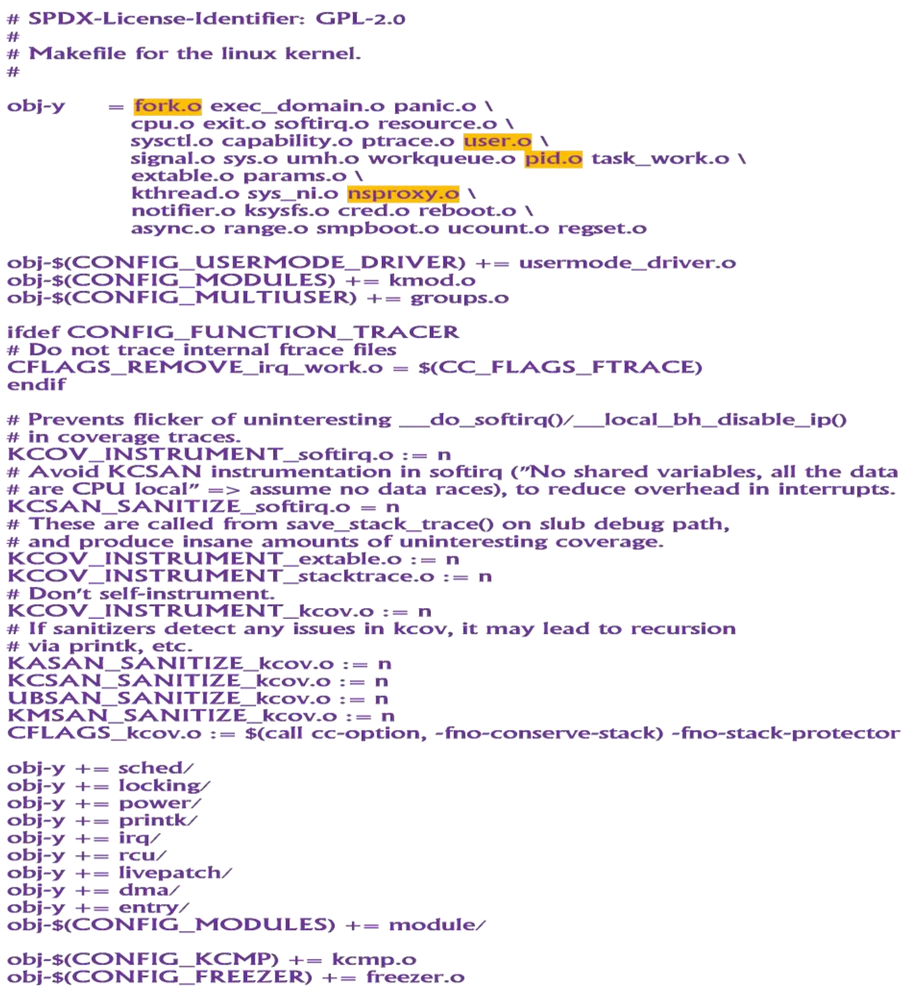
4.3.2.1.1 Extraction of the features contained in the Kernel Sub System

The extracted Linux Code contains the different Linux Sub Systens. This study is focused on the Kernel sub system. The contents of the Kernel sub system were filtered for Kernel sub system only

4.3.2.1.2 Kernel Makefile

The rules for building/generating Linux Kernel are contained in Kernel Makefile and in C preprocessor directives which control how to compile the corresponding source code (*Leonardo Passos et. al: Study of Feature Scattering in the Linux Kernel*) and (
Kbuild, "The kernel build infrastructure," www.kernel.org/doc/Documentation/kbuild, last seen: Feb. 14th, 2015.

The Linux Kernel MakeFile:). The core objects in the makefile are highlighted in the figure 12 below



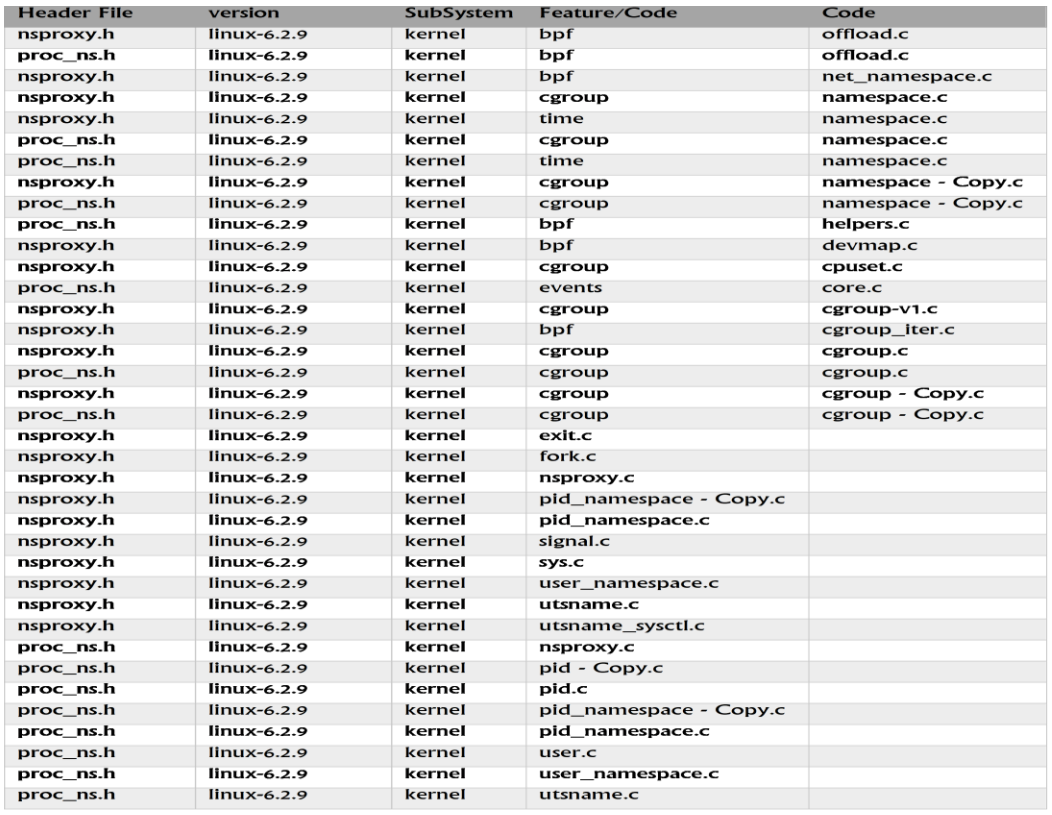
```
# SPDX-License-Identifier: GPL-2.0
#
# Makefile for the linux kernel.
#

obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o capability.o ptrace.o user.o \
            signal.o sys.o umh.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o ucount.o regset.o

obj-$(CONFIG_USERMODE_DRIVER) += usermode_driver.o
obj-$(CONFIG_MODULES) += kmod.o
obj-$(CONFIG_MULTIUSER) += groups.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace internal ftrace files
CFLAGS_REMOVE_irq_work.o = $(CC_FLAGS_FTRACE)
endif

# Prevents flicker of uninteresting __do_softirq()/__local_bh_disable_ip()
# in coverage traces.
KCOV_INSTRUMENT_softirq.o := n
# Avoid KCSAN instrumentation in softirq ("No shared variables, all the data
# are CPU local" => assume no data races), to reduce overhead in interrupts.
KCSAN_SANITIZE_softirq.o = n
# These are called from save_stack_trace() on slub debug path,
# and produce insane amounts of uninteresting coverage.
KCOV_INSTRUMENT_extable.o := n
KCOV_INSTRUMENT_stacktrace.o := n
# Don't self-instrument.
KCOV_INSTRUMENT_kcov.o := n
# If sanitizers detect any issues in kcov, it may lead to recursion
# via printk, etc.
KASAN_SANITIZE_kcov.o := n
KCSAN_SANITIZE_kcov.o := n
UBSAN_SANITIZE_kcov.o := n
KMSAN_SANITIZE_kcov.o := n
CFLAGS_kcov.o := $(call cc-option, -fno-conserve-stack) -fno-stack-protector

obj-y += sched/
obj-y += locking/
obj-y += power/
obj-y += printk/
obj-y += irq/
obj-y += rcu/
obj-y += livepatch/
obj-y += dma/
obj-y += entry/
obj-$(CONFIG_MODULES) += module/

obj-$(CONFIG_KCMP) += kcmp.o
obj-$(CONFIG_FREEZER) += freezer.o
```

Figure 12: Linux Kernel MakeFile

4.3.2.1.3 Identifying Key Features using Pre-Processor Directives

The below Script was used to Create a Database table to store the extracted details and list as shown in figure 13

```
USE [KernelFeatures]
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Linux_Code_Base](
        [Header_File] [varchar](50) NULL,
        [Linux_Version] [varchar](50) NULL,
        [Linux_SubSystem] [varchar](50) NULL,
        [Sub_Folder] [varchar](100) NULL,
        [C_Source_Header] [varchar](100) NULL,
        [Field6] [varchar](max) NULL,
        [Field7] [varchar](max) NULL,
        [Field8] [varchar](max) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```



Header File	version	SubSystem	Feature/Code	Code
nsproxy.h	linux-6.2.9	kernel	bpf	offload.c
proc_ns.h	linux-6.2.9	kernel	bpf	offload.c
nsproxy.h	linux-6.2.9	kernel	bpf	net_namespace.c
nsproxy.h	linux-6.2.9	kernel	cgroup	namespace.c
nsproxy.h	linux-6.2.9	kernel	time	namespace.c
proc_ns.h	linux-6.2.9	kernel	cgroup	namespace.c
proc_ns.h	linux-6.2.9	kernel	time	namespace.c
nsproxy.h	linux-6.2.9	kernel	cgroup	namespace - Copy.c
proc_ns.h	linux-6.2.9	kernel	cgroup	namespace - Copy.c
proc_ns.h	linux-6.2.9	kernel	bpf	helpers.c
nsproxy.h	linux-6.2.9	kernel	bpf	devmap.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cpuset.c
proc_ns.h	linux-6.2.9	kernel	events	core.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cgroup-v1.c
nsproxy.h	linux-6.2.9	kernel	bpf	cgroup_iter.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cgroup.c
proc_ns.h	linux-6.2.9	kernel	cgroup	cgroup.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cgroup - Copy.c
proc_ns.h	linux-6.2.9	kernel	cgroup	cgroup - Copy.c
nsproxy.h	linux-6.2.9	kernel	exit.c	
nsproxy.h	linux-6.2.9	kernel	fork.c	
nsproxy.h	linux-6.2.9	kernel	nsproxy.c	
nsproxy.h	linux-6.2.9	kernel	pid_namespace - Copy.c	
nsproxy.h	linux-6.2.9	kernel	pid_namespace.c	
nsproxy.h	linux-6.2.9	kernel	signal.c	
nsproxy.h	linux-6.2.9	kernel	sys.c	
nsproxy.h	linux-6.2.9	kernel	user_namespace.c	
nsproxy.h	linux-6.2.9	kernel	utsname.c	
nsproxy.h	linux-6.2.9	kernel	utsname_sysctl.c	
proc_ns.h	linux-6.2.9	kernel	nsproxy.c	
proc_ns.h	linux-6.2.9	kernel	pid - Copy.c	
proc_ns.h	linux-6.2.9	kernel	pid.c	
proc_ns.h	linux-6.2.9	kernel	pid_namespace - Copy.c	
proc_ns.h	linux-6.2.9	kernel	pid_namespace.c	
proc_ns.h	linux-6.2.9	kernel	user.c	
proc_ns.h	linux-6.2.9	kernel	user_namespace.c	
proc_ns.h	linux-6.2.9	kernel	utsname.c	

Figure 13: Linux Kernel Source Codes

4.3.2.1.4 Source Code Analysis

The Linux kernel's code base consists mainly of C implementation and header files numbering 43% C; 39% files for implementation and header files. The assembly files are about 4%. The rest are other kinds of files (*Code Project: Coding Kernel).*

4.3.2.1.4.1 Header Files

Two (2) header files referenced for process and namespace management were identified. These were used to identify the source code files as shown on Table 8

Table 8: Linux Kernel Process Header Files

Header File	**version**	**SubSystem**	**Feature/Code**	**Code**
nsproxy.h	linux-6.2.9	kernel	bpf	offload.c
proc_ns.h	linux-6.2.9	kernel	bpf	offload.c
nsproxy.h	linux-6.2.9	kernel	bpf	net_namespace.c
nsproxy.h	linux-6.2.9	kernel	cgroup	namespace.c
nsproxy.h	linux-6.2.9	kernel	time	namespace.c
proc_ns.h	linux-6.2.9	kernel	cgroup	namespace.c
proc_ns.h	linux-6.2.9	kernel	time	namespace.c
nsproxy.h	linux-6.2.9	kernel	cgroup	namespace - Copy.c
proc_ns.h	linux-6.2.9	kernel	cgroup	namespace - Copy.c
proc_ns.h	linux-6.2.9	kernel	bpf	helpers.c
nsproxy.h	linux-6.2.9	kernel	bpf	devmap.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cpuset.c
proc_ns.h	linux-6.2.9	kernel	events	core.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cgroup-v1.c
nsproxy.h	linux-6.2.9	kernel	bpf	cgroup_iter.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cgroup.c
proc_ns.h	linux-6.2.9	kernel	cgroup	cgroup.c
nsproxy.h	linux-6.2.9	kernel	cgroup	cgroup - Copy.c
proc_ns.h	linux-6.2.9	kernel	cgroup	cgroup - Copy.c
nsproxy.h	linux-6.2.9	kernel	exit.c	
nsproxy.h	linux-6.2.9	kernel	fork.c	
nsproxy.h	linux-6.2.9	kernel	nsproxy.c	
nsproxy.h	linux-6.2.9	kernel	pid_namespace - Copy.c	
nsproxy.h	linux-6.2.9	kernel	pid_namespace.c	
nsproxy.h	linux-6.2.9	kernel	signal.c	
nsproxy.h	linux-6.2.9	kernel	sys.c	
nsproxy.h	linux-6.2.9	kernel	user_namespace.c	
nsproxy.h	linux-6.2.9	kernel	utsname.c	

Header File	version	SubSystem	Feature/Code	Code
nsproxy.h	linux-6.2.9	kernel	utsname_sysctl.c	
proc_ns.h	linux-6.2.9	kernel	nsproxy.c	
proc_ns.h	linux-6.2.9	kernel	pid - Copy.c	
proc_ns.h	linux-6.2.9	kernel	pid.c	
proc_ns.h	linux-6.2.9	kernel	pid_namespace - Copy.c	
proc_ns.h	linux-6.2.9	kernel	pid_namespace.c	
proc_ns.h	linux-6.2.9	kernel	user.c	
proc_ns.h	linux-6.2.9	kernel	user_namespace.c	
proc_ns.h	linux-6.2.9	kernel	utsname.c	

4.3.2.1.5 Selected Features

4.3.2.1.5.1 Process ID Address Space Designation

Currently, processes are grouped into namespaces as away of isolating them. In the current set up, the maximum number of process identifications might not be as critical as the proposed approach where process address spaces are designated during onboarding process. The maximum limit of process identification can be set as shown in Figure 14.

4.3.2.1.5.1.1 Modification of the Process Identification Upper Limit

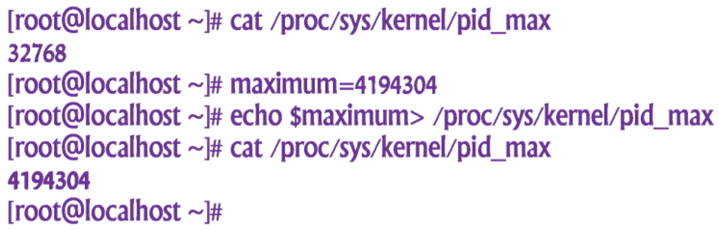
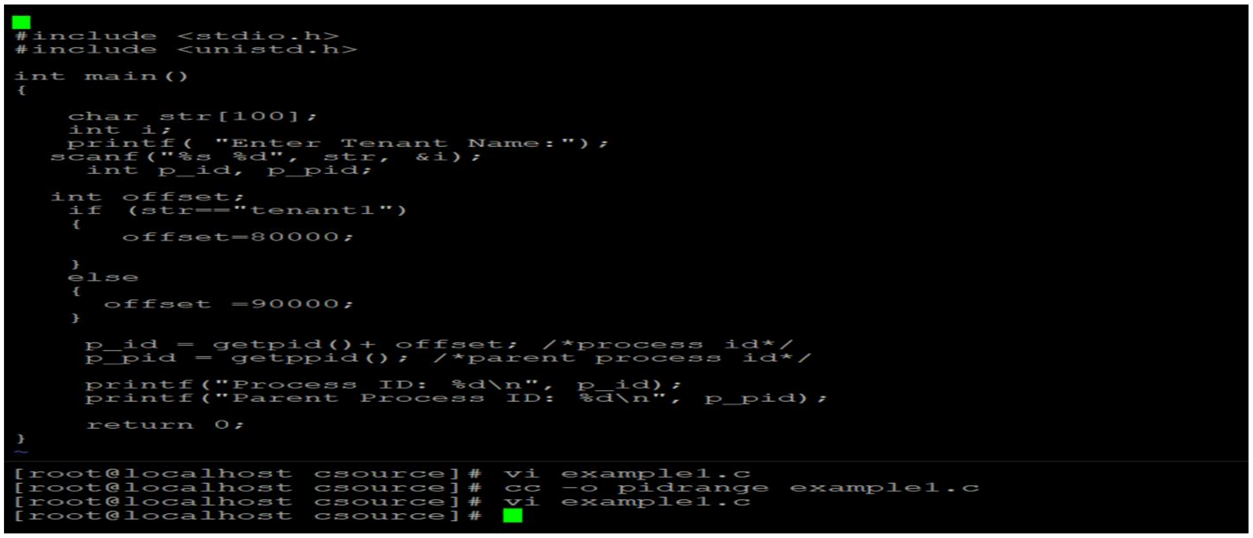

```
[root@localhost ~]# cat /proc/sys/kernel/pid_max
32768
[root@localhost ~]# maximum=4194304
[root@localhost ~]# echo $maximum> /proc/sys/kernel/pid_max
[root@localhost ~]# cat /proc/sys/kernel/pid_max
4194304
[root@localhost ~]#
```

Figure 14: Modification of PID Maximum value

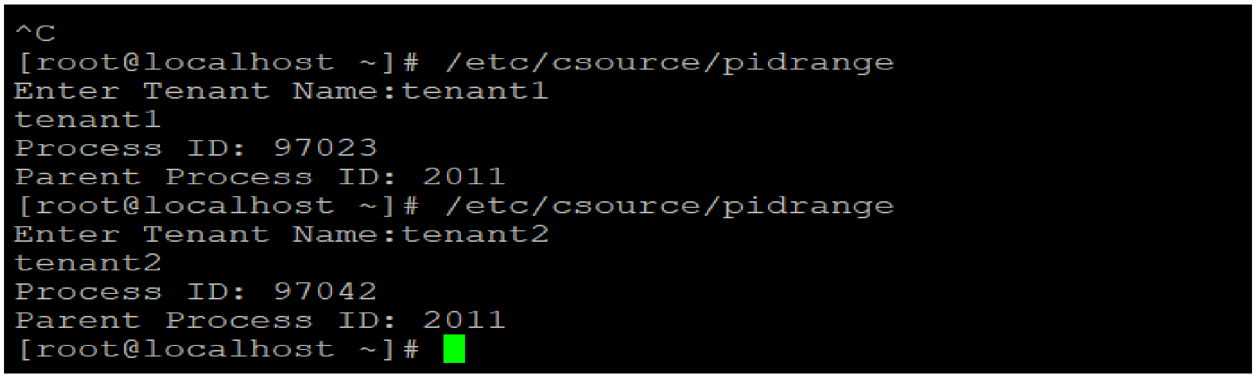
4.3.2.1.5.1.2 Code to generate Process Identification Number

The C code to generate is shown below in figure 15



```
#include <stdio.h>
#include <unistd.h>

int main()
{

    char str[100];
    int i;
    printf( "Enter Tenant Name:");
  scanf("%s %d", str, &i);
     int p_id, p_pid;

  int offset;
    if (str=="tenant1")
    {
        offset=80000;

    }
    else
    {
       offset =90000;
    }

     p_id = getpid()+ offset; /*process id*/
     p_pid = getppid(); /*parent process id*/

     printf("Process ID: %d\n", p_id);
     printf("Parent Process ID: %d\n", p_pid);

     return 0;
}
~
[root@localhost csource]# vi example1.c
[root@localhost csource]# cc -o pidrange example1.c
[root@localhost csource]# vi example1.c
[root@localhost csource]#
```

Figure 15: Allocation of PID Offset values



```
^C
[root@localhost ~]# /etc/csource/pidrange
Enter Tenant Name:tenant1
tenant1
Process ID: 97023
Parent Process ID: 2011
[root@localhost ~]# /etc/csource/pidrange
Enter Tenant Name:tenant2
tenant2
Process ID: 97042
Parent Process ID: 2011
[root@localhost ~]#
```

4.3.2.1.5.1.3 Bash Scripts to generate PID Based on the designated PID Range

Currently, the next process ID is the next number from the last number assigned to the last request.

```
#!/bin/bash
# This script is used to get PID Offset or range for a user
lowest=400000
echo "Enter Your Tenancy Name"
read name
if [[ ( $name=="msc1" ) ]]; then
      lowest=200000
fi
if [[ ( $name=="msc2" ) ]]; then
        lowest=300000
fi
    pid=lowest + echo "$!"  " #(" ${pid##*/} + lowest") # Extract PID
```

The idea was to create an offset or a range of PID number space which is then ranged and designated to the individual tenants. This proved complex and requires more time to scour through the source code and header files to see how best to implement this

4.3.2.1.5.2 User Augmentation - Attributing the /etc/passwd to include tenancy information

The Kernel as indicated above provides for User Namespace that allows to map users in the container to different users in the host.

4.3.2.1.5.2.1 Default Fields

User details are stored in **/…/passwd** file. The file is used to store the user's information

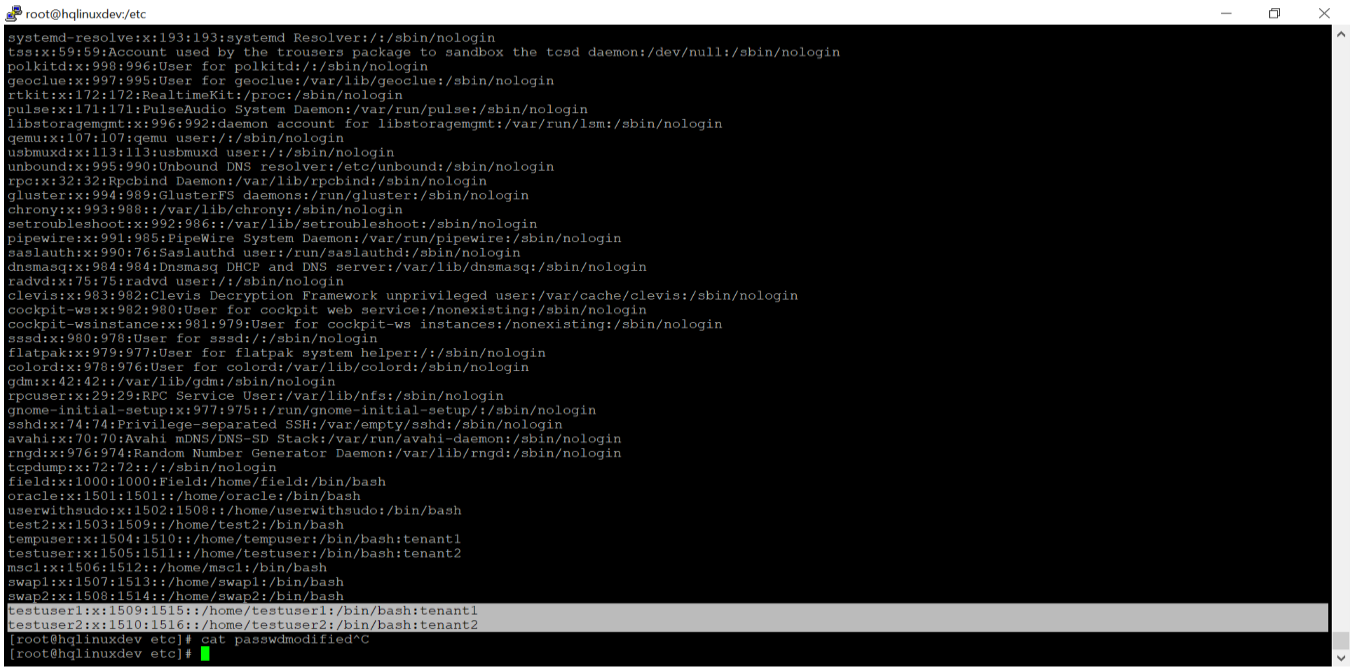
The Linux user management/creation system call , the useradd syscall in Linux, contains a set of seven colon-separated fields, each field has its own meaning. The fields are: Username, Password, User ID, Group ID, User Info, Home Directory & Shell

4.3.2.1.5.2.2 Expanded User File Structure with an additional Attribute to Store Tenant information

The above fields were expanded to include tenancy as shown below:

Tenancy: This is to store Tenancy Information

```
testuser1:x:1509:1515::/home/testuser1:/bin/bash:tenant1
testuser2:x:1510:1516::/home/testuser2:/bin/bash:tenant2
```



```
systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
geoclue:x:997:995:User for geoclue:/var/lib/geoclue:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
pulse:x:171:171:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
libstoragemgmt:x:996:992:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin
qemu:x:107:107:qemu user:/:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
unbound:x:995:990:Unbound DNS resolver:/etc/unbound:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
gluster:x:994:989:GlusterFS daemons:/run/gluster:/sbin/nologin
chrony:x:993:988::/var/lib/chrony:/sbin/nologin
setroubleshoot:x:992:986::/var/lib/setroubleshoot:/sbin/nologin
pipewire:x:991:985:PipeWire System Daemon:/var/run/pipewire:/sbin/nologin
saslauth:x:990:76:Saslauthd user:/run/saslauthd:/sbin/nologin
dnsmasq:x:984:984:Dnsmasq DHCP and DNS server:/var/lib/dnsmasq:/sbin/nologin
radvd:x:75:75:radvd user:/:/sbin/nologin
clevis:x:983:982:Clevis Decryption Framework unprivileged user:/var/cache/clevis:/sbin/nologin
cockpit-ws:x:982:980:User for cockpit web service:/nonexisting:/sbin/nologin
cockpit-wsinstance:x:981:979:User for cockpit-ws instances:/nonexisting:/sbin/nologin
sssd:x:980:978:User for sssd:/:/sbin/nologin
flatpak:x:979:977:User for flatpak system helper:/:/sbin/nologin
colord:x:978:976:User for colord:/var/lib/colord:/sbin/nologin
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
gnome-initial-setup:x:977:975::/run/gnome-initial-setup/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
rngd:x:976:974:Random Number Generator Daemon:/var/lib/rngd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
field:x:1000:1000:Field:/home/field:/bin/bash
oracle:x:1501:1501::/home/oracle:/bin/bash
userwithsudo:x:1502:1508::/home/userwithsudo:/bin/bash
test2:x:1503:1509::/home/test2:/bin/bash
tempuser:x:1504:1510::/home/tempuser:/bin/bash:tenant1
testuser:x:1505:1511::/home/testuser:/bin/bash:tenant2
msc1:x:1506:1512::/home/msc1:/bin/bash
swap1:x:1507:1513::/home/swap1:/bin/bash
swap2:x:1508:1514::/home/swap2:/bin/bash
testuser1:x:1509:1515::/home/testuser1:/bin/bash:tenant1
testuser2:x:1510:1516::/home/testuser2:/bin/bash:tenant2
[root@hqlinuxdev etc]# cat passwdmodified^C
[root@hqlinuxdev etc]#
```

4.3.2.1.5.2.3 Modified UserAdd (UserAddnew) code

A python script to create a user with a modified structure is shown in figure 16

```
import paramiko
import sys
hostname = "19.176.244.103"
username = "root"
password = "xxxxxxx"

# initialize the SSH client
client = paramiko.SSHClient()
# add to known hosts
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
try:
    client.connect(hostname=hostname, username=username,
password=password)
except:
    print("[!] Cannot connect to the SSH Server")
    exit()
    # execute the commands
tenantname = input("Type Tenant Name ")
timezzone= input("Type Time Zone Offset in hours ")
txtipaddress= input("IP Address ")
txtactivationdate=input("Activation Date ")
txtdeactivationdate=input("De-activation Date ")
txtmemorysoftlimit=input("Memory Lower Limit ")
txtmemoryhardlimit=input("Memory Max Limit ")
txtprocessorunitsmin=input("Processor units Lower Limit ")
txtprocessorunitsmax=input("Processor units max Limit ")
commands = [
    "mkdir /home/" +  tenantname,
    "mkdir /sys/fs/cgroup/memory/" + tenantname,
    "mkdir /sys/fs/cgroup/cpu/" + tenantname,
    "useradd " +  tenantname + "001 -d /home/" + tenantname
]
# prints inp

for command in commands:
    print("="*50, command, "="*50)
    stdin, stdout, stderr = client.exec_command(command)
    #stdin, stdout, stderr = con.exec_command('echo "test\ntest2" | bash
test.sh')
    print(stdout.read().decode())
    err = stderr.read().decode()
    if err:
        print(err)
```

Figure 16: Python script to create a modified user

4.3.2.1.5.2.4 Modification of the default Home Directory

Figure 17 shows inclusion of an home directory defined per tenant. Within this home directory, the individual users of the tenant are assigned own home directories

```
import paramiko
import sys
hostname = "19.176.244.103"
username = "root"
password = "xxxxxxx"
client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
try:
    client.connect(hostname=hostname, username=username,
password=password)
except:
    print("[!] Cannot connect to the SSH Server")
    exit()
    # execute the commands
tenantname = input("Type Tenant Name ")
timezzone= input("Type Time Zone Offset in hours ")
txtipaddress= input("IP Address ")
txtactivationdate=input("Activation Date ")
txtdeactivationdate=input("De-activation Date ")
txtmemorysoftlimit=input("Memory Lower Limit ")
txtmemoryhardlimit=input("Memory Max Limit ")
txtprocessorunitsmin=input("Processor units Lower Limit ")
txtprocessorunitsmax=input("Processor units max Limit ")
commands = [
    "mkdir /home/" +  tenantname,
    "mkdir /sys/fs/cgroup/memory/" + tenantname,
    "mkdir /sys/fs/cgroup/cpu/" + tenantname,
    "useradd " +  tenantname + "001 -d /home/" + tenantname
]
# prints inp

for command in commands:
    print("="*50, command, "="*50)
    stdin, stdout, stderr = client.exec_command(command)
    #stdin, stdout, stderr = con.exec_command('echo "test\ntest2" |
bash test.sh')
    print(stdout.read().decode())
    err = stderr.read().decode()
    if err:
        print(err)
```

Figure 17: Home Directory for the tenant

4.3.2.1.5.2.5 Global Tenancy Configuration File

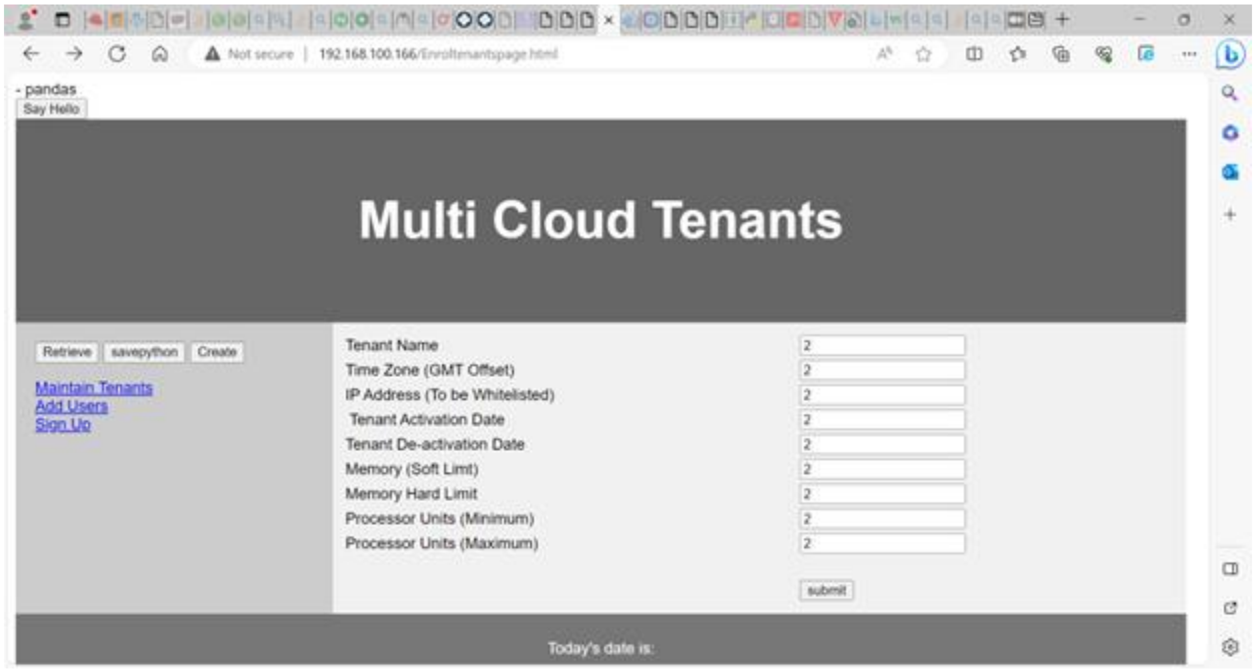
4.3.2.1.5.2.5.1 Global Tenancy Configuration File

Tenant Configuration File is created and below is a snapshot of this file:

```
[root@hqlinuxdev etc]# vi tenants.conf
    [root@hqlinuxdev etc]# cat tenants.conf
    |tenant1|192.168.100.166|tenant 1| GMT+1|01-01-2023| 31-
    12-2099| memory| swapfile| proc| diskspace|
    |tenant2|10.176.190.190|tent 2|GMT+3|04-09-2023|31-12-
    2023|100|swapfiletenant2|proc|1000|
    [root@hqlinuxdev etc]#
```

4.3.2.1.5.2.6 User Provisioning Screens

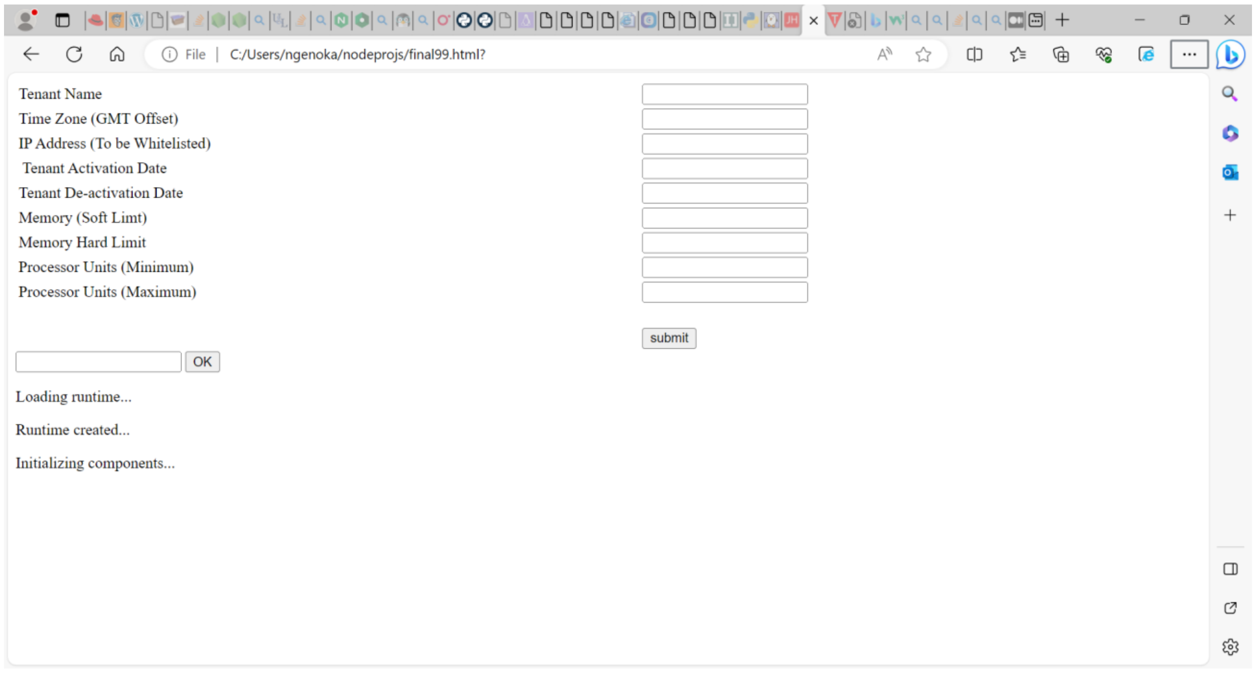
Figure 18 is a snippet of web console for provisioning tenants; web consoles are a defining characteristic of cloud computing



Multi Cloud Tenants

Tenant Name	2
Time Zone (GMT Offset)	2
IP Address (To be Whitelisted)	2
Tenant Activation Date	2
Tenant De-activation Date	2
Memory (Soft Limt)	2
Memory Hard Limit	2
Processor Units (Minimum)	2
Processor Units (Maximum)	2

Today's date is:


C:/Users/ngenoka/nodeprojs/final99.html?

Tenant Name	
Time Zone (GMT Offset)	
IP Address (To be Whitelisted)	
Tenant Activation Date	
Tenant De-activation Date	
Memory (Soft Limt)	
Memory Hard Limit	
Processor Units (Minimum)	
Processor Units (Maximum)	

Loading runtime...

Runtime created...

Initializing components...

Figure 18: User provisioning web page

4.3.2.1.5.2.6.1 Python Script to Create User

This script is used to create a tenant/user

```
import paramiko
import sys
import string
hostname = "192.168.43.20"
username = "root"
password = "Password@04"

# initialize the SSH client
client = paramiko.SSHClient()
# add to known hosts
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
try:
    client.connect(hostname=hostname, username=username, password=password)
except:
    print("[!] Cannot connect to the SSH Server")
    exit()
    # execute the commands
tenantname = input("Type Tenant Name ")
timezzone= input("Type Time Zone Offset in hours ")
txtipaddress= input("IP Address ")
txtactivationdate=input("Activation Date ")
txtdeactivationdate=input("De-activation Date ")
```

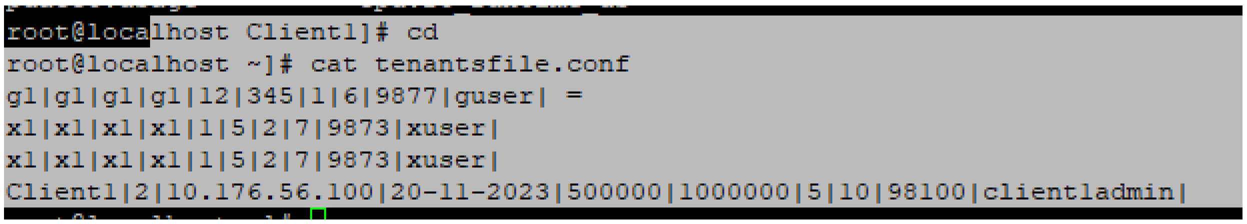
```
txtmemorysoftlimit=input("Memory Lower Limit ")
txtmemoryhardlimit=input("Memory Max Limit ")
txtprocessorunitsmin=input("Processor units Lower Limit ")
txtprocessorunitsmax=input("Processor units max Limit ")
txtpidsrange=input("Enter PIDs list......valid at least 10000")
txtrootuser=input("Enter root User for the tenant")
commands = [
    "mkdir /home/" +  tenantname,
    "mkdir /sys/fs/cgroup/memory/" + tenantname,
    "mkdir /sys/fs/cgroup/cpu/" + tenantname,
    "cd /sys/fs/cgroup/memory/" + tenantname,
    #"useradd " +  tenantname + "001 -d /home/" + tenantname
    "echo " + txtmemorysoftlimit + " >> memory.limit_in_bytes",
    "echo " + txtpidsrange + " >> cgroup.procs",
    "cd /sys/fs/cgroup/cpu/" + tenantname,
    "echo " + txtprocessorunitsmin + " >> cpu_shares",
    "echo " + txtpidsrange + " >> cgroup.procs",
    "cd ",
    "useradd " + txtrootuser + "001 -d /home/" + tenantname,
    "cd /etc",
    #vals = 'Tenant',
    #vals = "|" + tenantname +"|" + timezzone + "|" + txtipaddress +"|" + txtactivationdate + "|" +
txtmemorysoftlimit + "|" +
    #txtmemoryhardlimit + "|" + txtprocessorunitsmin +"|" + txtprocessorunitsmax + "|" +
txtpidsrange + "|" + txtrootuser + "|",
    "echo " + str("'|" + tenantname +"|" + timezzone + "|" + txtipaddress +"|" + txtactivationdate +
"|" + txtmemorysoftlimit + "|" +
    txtmemoryhardlimit + "|" + txtprocessorunitsmin +"|" + txtprocessorunitsmax + "|" +
txtpidsrange + "|" + txtrootuser + "|'")  + " >> tenantsfile.conf"
]
# prints inp

for command in commands:
    print("="*50, command, "="*50)
    stdin, stdout, stderr = client.exec_command(command)
    #stdin, stdout, stderr = con.exec_command('echo "test\ntest2" | bash test.sh')
    print(stdout.read().decode())
    err = stderr.read().decode()
    if err:
        print(err)
```

```
C:\Users\ngenoka>python C:\Users\ngenoka\nodeprojs\createtenantswithPIDs.py

Type Tenant Name Client1

Type Time Zone Offset in hours 2

IP Address 10.176.56.100

Activation Date 20-11-2023

De-activation Date

Memory Lower Limit 500000

Memory Max Limit 1000000

Processor units Lower Limit 5

Processor units max Limit 10

Enter PIDs list......valid at least 1000098100

Enter root User for the tenantclient1admin

====================================================== mkdir /home/Client1
======================================================



====================================================== mkdir
/sys/fs/cgroup/memory/Client1
======================================================



====================================================== mkdir
/sys/fs/cgroup/cpu/Client1
======================================================



====================================================== cd
/sys/fs/cgroup/memory/Client1 ============================ echo 500000 >>
memory.limit_in_bytes ======================================================
======================================================= echo 98100 >>
cgroup.procs =======================================================

======================================================= cd
/sys/fs/cgroup/cpu/Client1
=======================================================

======================================================= echo 5 >> cpu_shares
=======================================================

======================================================= echo 98100 >>
cgroup.procs =======================================================

======================================================= cd
=======================================================

======================================================= useradd client1admin001
-d /home/Client1 =======================================================

useradd: warning: the home directory already exists.

Not copying any file from skel directory into it.

======================================================= cd /etc
=======================================================

======================================================= echo
'|Client1|2|10.176.56.100|20-11-2023|500000|1000000|5|10|98100|client1admin|' >>
tenantsfile.conf =======================================================
```

4.3.2.1.5.2.6.2 User Record in password file (passwd) & Global Configuration

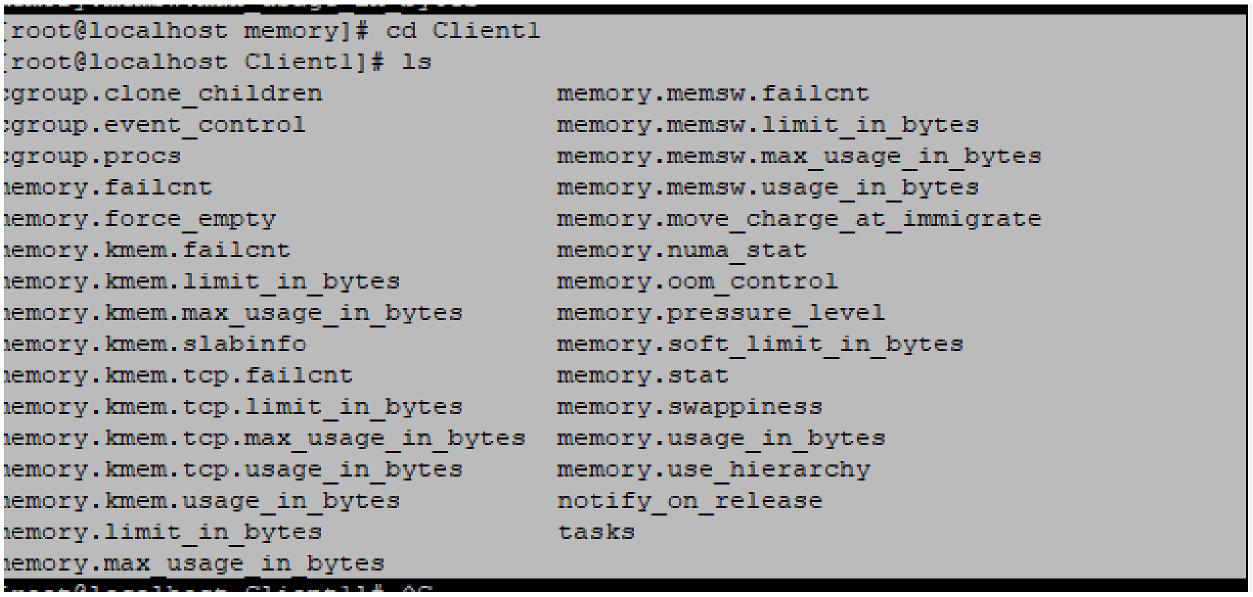
To view the contents of the user file, the following commands can be executed

```
[root@localhost ~]# cat /etc/passwd | grep client1

client1admin001:x:1014:1014::/home/Client1:/bin/bash

[root@localhost ~]#
```

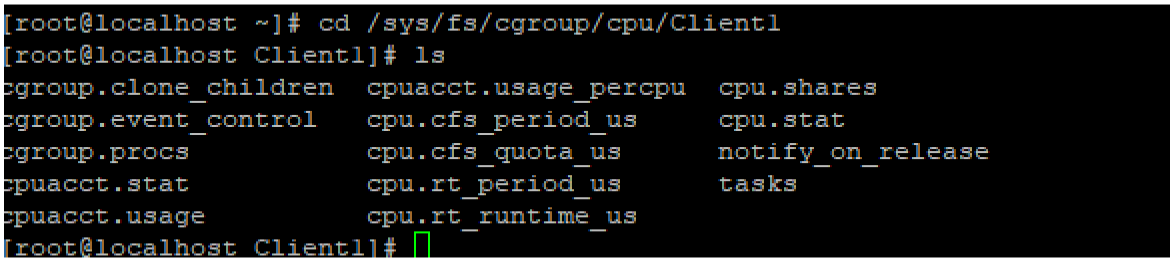


```
root@localhost Client1]# cd
root@localhost ~]# cat tenantsfile.conf
gl|gl|gl|gl|12|345|1|6|9877|guser| =
xl|xl|xl|xl|1|5|2|7|9873|xuser|
xl|xl|xl|xl|1|5|2|7|9873|xuser|
Client1|2|10.176.56.100|20-11-2023|500000|1000000|5|10|98100|clientladmin|
```

4.3.2.1.5.2.6.3 Entries in the Cgroup



```
[root@localhost memory]# cd Client1
[root@localhost Client1]# ls
:group.clone_children               memory.memsw.failcnt
:group.event_control                memory.memsw.limit_in_bytes
:group.procs                        memory.memsw.max_usage_in_bytes
iemory.failcnt                      memory.memsw.usage_in_bytes
iemory.force_empty                  memory.move_charge_at_immigrate
iemory.kmem.failcnt                 memory.numa_stat
iemory.kmem.limit_in_bytes          memory.oom_control
iemory.kmem.max_usage_in_bytes      memory.pressure_level
iemory.kmem.slabinfo                memory.soft_limit_in_bytes
iemory.kmem.tcp.failcnt             memory.stat
iemory.kmem.tcp.limit_in_bytes      memory.swappiness
iemory.kmem.tcp.max_usage_in_bytes  memory.usage_in_bytes
iemory.kmem.tcp.usage_in_bytes      memory.use_hierarchy
iemory.kmem.usage_in_bytes          notify_on_release
iemory.limit_in_bytes               tasks
iemory.max_usage_in_bytes
```



```
[root@localhost ~]# cd /sys/fs/cgroup/cpu/Client1
[root@localhost Client1]# ls
cgroup.clone_children  cpuacct.usage_percpu  cpu.shares
cgroup.event_control   cpu.cfs_period_us     cpu.stat
cgroup.procs           cpu.cfs_quota_us      notify_on_release
cpuacct.stat           cpu.rt_period_us      tasks
cpuacct.usage          cpu.rt_runtime_us
[root@localhost Client1]#
```

4.3.2.1.5.2.6.4 Read Scripts

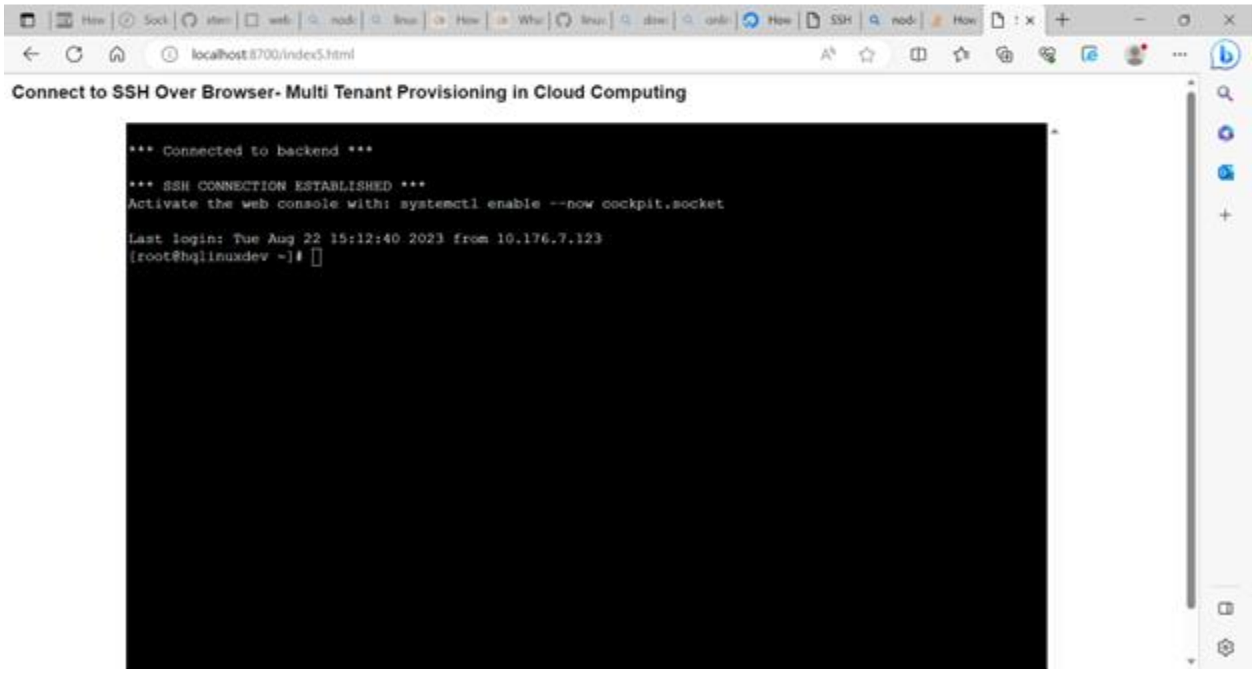
```
[root@hqlinuxdev bin]# vi readglobal
        #!/bin/bash
        cat /etc/tenants.conf
chmod +x readglobal

[root@hqlinuxdev bin]# readglobal

|tenant1|192.168.100.166|tenant 1| GMT+1|01-01-2023| 31-12-2099| memory| swapfile| proc| diskspace|
|tenant2|10.176.190.190|tent 2|GMT+3|04-09-2023|31-12-2023|100|swapfiletenant2|proc|1000|

[root@hqlinuxdev bin]#
```

<u>Python Script</u>

```
pswd = file( "/etc/tenants.conf", "r" )
for aLine in pswd:
    fields= aLine.split( ":" )
    print fields[0], fields[1], fields[2], fields[3], fields[5]
pswd.close()
```

4.3.2.1.5.2.6.5 Modified Terminal Login Shell – Web accessible SSH

One of the key distinguishing features of Cloud Computing is access over the internet. As part of this study, a node web server and web page as shown in figure 19



Connect to SSH Over Browser- Multi Tenant Provisioning in Cloud Computing

```
*** Connected to backend ***

*** SSH CONNECTION ESTABLISHED ***
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Tue Aug 22 15:12:40 2023 from 10.176.7.123
[root@hqlinuxdev ~]# 
```

Figure 19: Access Linux Shell over web browser

4.3.2.1.5.2.6.6 Server5.js

The Nodejs code for creating a web server to listen on port 8700 is shown in figure 20.

```
//node js server
var fs = require('fs');
var path = require('path');
var server = require('http').createServer(onRequest);
var io = require('socket.io')(server);
var SSHClient = require('ssh2').Client;

// Load static files into memory
var staticFiles = {};
var basePath = path.join(require.resolve('xterm'), '..');
staticFiles['/xterm.css'] = fs.readFileSync(path.join(basePath,
'../css/xterm.css'));
staticFiles['/xterm.js'] = fs.readFileSync(path.join(basePath,
'xterm.js'));
basePath = path.join(require.resolve('xterm-addon-fit'), '..');
staticFiles['/xterm-addon-fit.js'] =
fs.readFileSync(path.join(basePath, 'xterm-addon-fit.js'));
//staticFiles['/'] = fs.readFileSync( ba, '../index5.html'));

staticFiles['/index5.html'] = fs.readFileSync(path.join(__dirname,
'/index5.html'));

// Handle static file serving
function onRequest(req, res) {
  var file;
  if (req.method === 'GET' && (file = staticFiles[req.url])) {
    res.writeHead(200, {
      'Content-Type': 'text/'
        + (/css$/.test(req.url)
        ? 'css'
        : (/js$/.test(req.url) ? 'javascript' : 'html'))
    });
    return res.end(file);
  }
  res.writeHead(404);
  res.end();
}

io.on('connection', function(socket) {
  var conn = new SSHClient();
  conn.on('ready', function() {
    socket.emit('data', '\r\n*** SSH CONNECTION
ESTABLISHED ***\r\n');
    conn.shell(function(err, stream) {
      if (err)
        return socket.emit('data', '\r\n*** SSH SHELL ERROR: ' +
err.message + ' ***\r\n');
      socket.on('data', function(data) {
        stream.write(data);
      });
      stream.on('data', function(d) {
        socket.emit('data', d.toString('binary'));
      }).on('close', function() {
        conn.end();
      });
    });
  }).on('close', function() {
    socket.emit('data', '\r\n*** SSH CONNECTION CLOSED
***\r\n');
  }).on('error', function(err) {
    socket.emit('data', '\r\n*** SSH CONNECTION ERROR: ' +
err.message + ' ***\r\n');
  }).connect({
    host: '***************',
    port: 22,
    username: '********',
    password: *********
    //privateKey: require('fs').readFileSync('path/to/keyfile')
  });
});

let port = 8700;
console.log('Listening on port', port)
server.listen(port);
```

Figure 20: NodeJS Server code

4.3.2.1.5.2.6.7 Web page – index5.html

HTML code to render the web page used to access ssh session over browser on http port 8700 is highlighted in figure 21

```
<html>
  <head>
    <h1>
    <title>SSH Terminal - Masters Project</title>
    <link rel="stylesheet" href="/xterm.css" />
    <script src="/xterm.js"></script>
    <script src="/xterm-addon-fit.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      window.addEventListener('load', function() {
        var terminalContainer = document.getElementById('terminal-
container');
        const term = new Terminal({ cursorBlink: true });
        const fitAddon = new FitAddon.FitAddon();
        term.loadAddon(fitAddon);
        term.open(terminalContainer);
        fitAddon.fit();

        var socket = io() //.connect();
        socket.on('connect', function() {
          term.write('\r\n*** Connected to backend ***\r\n');
        });

        // Browser -> Backend
        term.onKey(function (ev) {
          socket.emit('data', ev.key);
        });

        // Backend -> Browser
        socket.on('data', function(data) {
          term.write(data);
        });

        socket.on('disconnect', function() {
          term.write('\r\n*** Disconnected from backend ***\r\n');
        });
      }, false);
    </script>
    <style>
      body {
        font-family: helvetica, sans-serif, arial;
        font-size: 1em;
        color: #111;
      }
      h1 {
        text-align: center;
      }
      #terminal-container {
        width: 960px;
        height: 600px;
        margin: 0 auto;
        padding: 2px;
      }
      #terminal-container .terminal {
        background-color: #111;
        color: #fafafa;
        padding: 2px;
      }
```

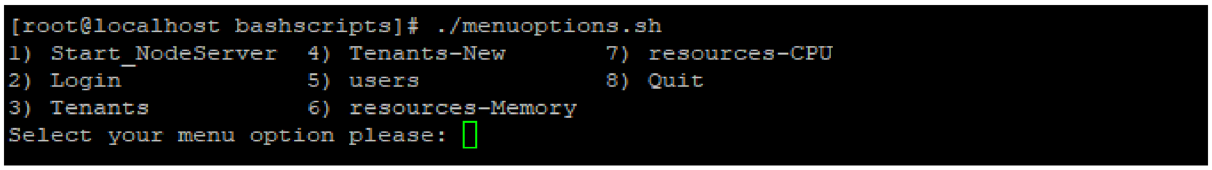
Figure 21: HTML Page to access the web console

```
      }
      #terminal-container .terminal:focus .terminal-cursor {
        background-color: #fafafa;
      }
    </style>
  </head>
  <body>
    <h3>Connect to SSH Over Browser- Multi Tenant Provisioning in
Cloud Computing </h3>
    <div id="terminal-container"></div>
  </body>
</html>
```

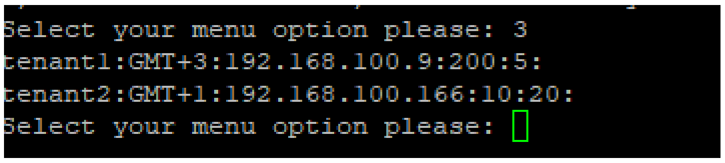
4.3.2.1.5.2.6.8 Menu

A menu was created via bash script for accessing the different menu options as shown in figure 22

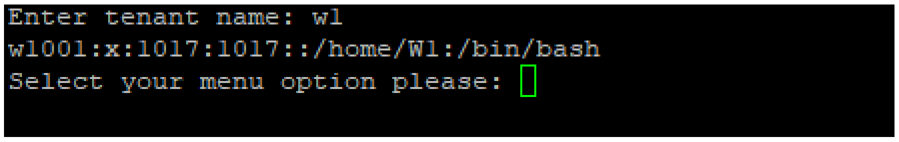
```
#!/bin/bash

PS3="Select your menu option please: "

select opt in Start_NodeServer Login Tenants Tenants-New users resources-Memory resources-
CPU Quit
do
    case $opt in
        "Start_NodeServer")
            node /var/www/cgi-bin/server5.js;;
        "Login")
           curl http://192.168.100.193:8700/index5.html ;;
        "Tenants")
          cat /etc/tenants.conf;;
        "Tenants-New")
           python createtenantswithPIDs.py;;
        "users")
           read -p "Enter tenant name: " tenant
           cat /etc/passwd | grep $tenant;;
        "resources-Memory")
           read -t "Enter tenant name: " tenant
           tenant2="/sys/fs/cgroup/memory/$tenant"
           ls $tenant2;;
        "resources-CPU")
           read -u "Enter tenant name: " client
           client2="/sys/fs/cgroup/cpu/$client"
           ls $client2;;
        "Quit")
           echo "We're done"
           break;;
    *)
      echo "Ooops";;
  esac
done
```



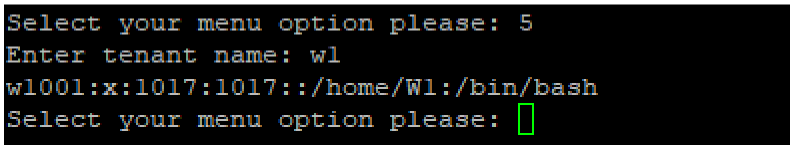
```
[root@localhost bashscripts]# ./menuoptions.sh
1) Start_NodeServer  4) Tenants-New        7) resources-CPU
2) Login             5) users              8) Quit
3) Tenants           6) resources-Memory
Select your menu option please:
```



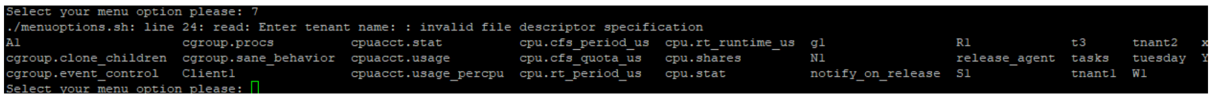
```
Select your menu option please: 3
tenant1:GMT+3:192.168.100.9:200:5:
tenant2:GMT+1:192.168.100.166:10:20:
Select your menu option please:
```



```
Enter tenant name: wl
wl001:x:1017:1017::/home/Wl:/bin/bash
Select your menu option please:
```

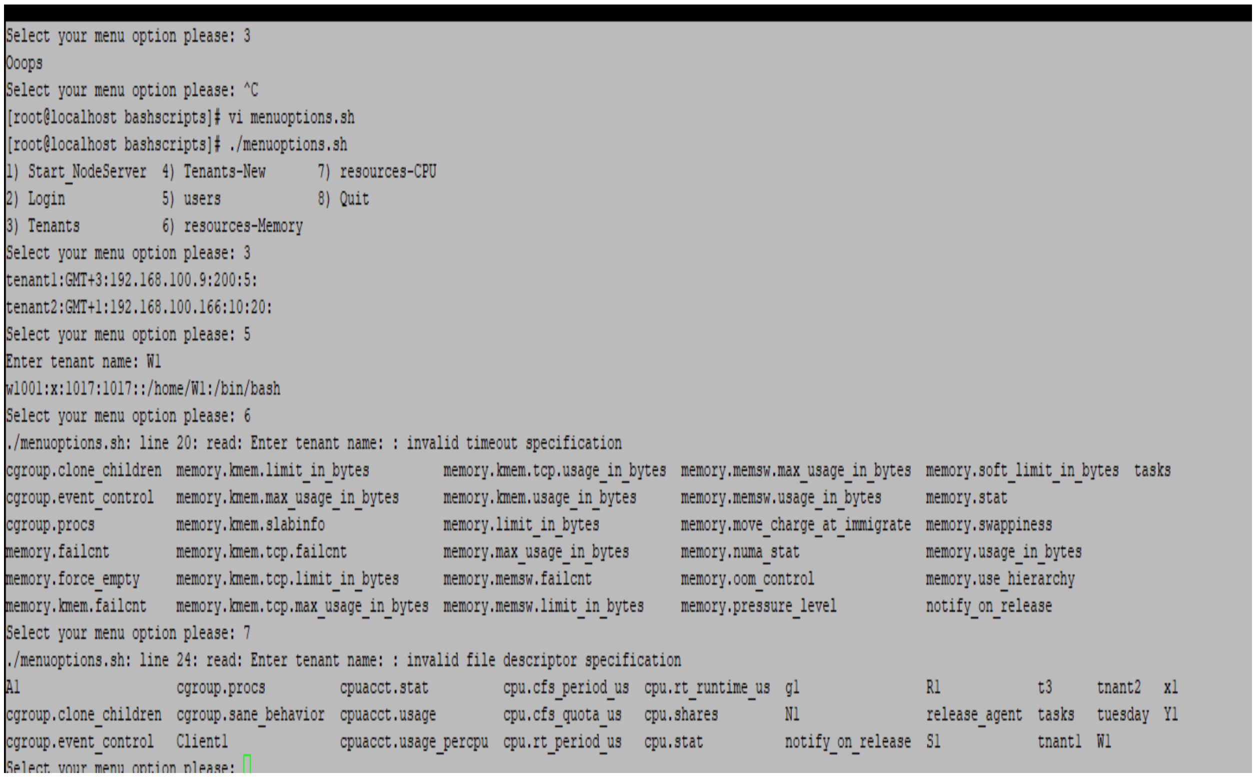


```
Select your menu option please: 5
Enter tenant name: wl
wl001:x:1017:1017::/home/Wl:/bin/bash
Select your menu option please:
```



```
Select your menu option please: 7
./menuoptions.sh: line 24: read: Enter tenant name: : invalid file descriptor specification
A1                     cgroup.procs           cpuacct.stat           cpu.cfs_period_us  cpu.rt_runtime_us  gl                 R1             t3      tnant2   x
cgroup.clone_children  cgroup.sane_behavior   cpuacct.usage          cpu.cfs_quota_us   cpu.shares         N1                 release_agent  tasks   tuesday  Y
cgroup.event_control   Client1                cpuacct.usage_percpu   cpu.rt_period_us   cpu.stat           notify_on_release  S1             tnant1  W1
Select your menu option please:
```

Figure 22: Bash Menu Script and Menu screens



```
Select your menu option please: 3
Ooops
Select your menu option please: ^C
[root@localhost bashscripts]# vi menuoptions.sh
[root@localhost bashscripts]# ./menuoptions.sh
1) Start_NodeServer  4) Tenants-New       7) resources-CPU
2) Login             5) users             8) Quit
3) Tenants           6) resources-Memory
Select your menu option please: 3
tenant1:GMT+3:192.168.100.9:200:5:
tenant2:GMT+1:192.168.100.166:10:20:
Select your menu option please: 5
Enter tenant name: W1
w1001:x:1017:1017::/home/W1:/bin/bash
Select your menu option please: 6
./menuoptions.sh: line 20: read: Enter tenant name: : invalid timeout specification
cgroup.clone_children  memory.kmem.limit_in_bytes          memory.kmem.tcp.usage_in_bytes  memory.memsw.max_usage_in_bytes  memory.soft_limit_in_bytes  tasks
cgroup.event_control   memory.kmem.max_usage_in_bytes      memory.kmem.usage_in_bytes      memory.memsw.usage_in_bytes      memory.stat
cgroup.procs           memory.kmem.slabinfo                memory.limit_in_bytes           memory.move_charge_at_immigrate  memory.swappiness
memory.failcnt         memory.kmem.tcp.failcnt             memory.max_usage_in_bytes       memory.numa_stat                 memory.usage_in_bytes
memory.force_empty     memory.kmem.tcp.limit_in_bytes      memory.memsw.failcnt            memory.oom_control               memory.use_hierarchy
memory.kmem.failcnt    memory.kmem.tcp.max_usage_in_bytes  memory.memsw.limit_in_bytes     memory.pressure_level            notify_on_release
Select your menu option please: 7
./menuoptions.sh: line 24: read: Enter tenant name: : invalid file descriptor specification
A1                     cgroup.procs         cpuacct.stat          cpu.cfs_period_us  cpu.rt_runtime_us  g1                 R1             t3      tnant2   x1
cgroup.clone_children  cgroup.sane_behavior cpuacct.usage         cpu.cfs_quota_us   cpu.shares         N1                 release_agent  tasks   tuesday  Y1
cgroup.event_control   Client1              cpuacct.usage_percpu  cpu.rt_period_us   cpu.stat           notify_on_release  S1             tnant1  W1
Select your menu option please: 
```

CHAPTER 5: RESULTS & DISCUSSIONS

5.1 Features selected for Isolation at Kernel

Three key features were selected and tested. These are the user and process numbering. A new approach to process identification that is based on designated address space as opposed to using process namespaces to cluster processes was simulated and once perfected, this can achieve more than isolation as it can also be used to forestall cyber attacks. In the simulation, an offset to be added to generated Process number was used.

5.2 User Hierarchy

Linux CentOS supports creation of users and assignment of default home directories. The users can be grouped into groups for management of permissions. The current user hierarchy cannot support multi tenancy as all users are created at the same level but can be assigned to different groups.

To cluster users at a higher level, a new root node for users is required. This was achieved by creating a tenant configuration file; the high level was called tenancies.

5.3 Process Identification Generation

Without affecting any of the current kernel processes, it was simulated that the process number generation can be stepped and the step value is the value assigned to a tenant.

The designated address or number space can as well also be a resource that can be used to limit resources a user can use in the system. Once the address space is depleted, the system cannot spawn additional processes until current processes are released

5.4 Access to Linux Secure Shell Through Web browser

Using the scripting languages libraries, packages and modules, a web client was developed that could connect to the Linux host using some socket connections and web containers and elements to render the secure shell session on the web browser.

5.5 cGroup Limitations

The cGroup only limits the namespaced features. Any feature that is not currently namespaced, is not supported by the cGroups.

5.6 Scripting Languages Support

Scripting languages are comparable to bash scripts in terms of performing the same functions that can be performed on the shell prompt. The cloud services use these scripting platforms to provision accessibility and control to cloud services.

CHAPTER 6: CONCLUSIONS & FUTURE WORK

6.1. Achievements

In this Study, inter alia, the following were achieved

Understanding of the Linux kernel structure and how to identify the key features of Linux; Identification of the Kernel features that are already inlined that can be used to support process and resource Isolation and impose usage control.

A new approach to process isolation based on pre-allocated and pre designated Process IDs (PIDs) as opposed to relying on random PIDs that require tracking mechanism and consumes resources.

Using Nodejs and Python packages to implement web access to Linux Shell prompt and to automate execution of the various Linux commands and system calls

Automation of user management through use of scripting languages and mainly node and Python

6.2 Conclusions

The isolation and multi tenancy concepts date back to many decades. There have been several attempts, initiatives, and technologies to comprehensively address this technology requirement. Virtualization at hardware level has been widely used to date. For other cases, OS-Based virtualization has been adopted.

However, research findings have revealed that the above two technologies come up with an overhead and hog critical resources that would otherwise go to data processing. There are also concerns around security isolation to ensure that isolated environments are secure and not ‘porous’

Isolation at the Linux Kernel level for Linux environments seems to present a promise to resolve the challenge

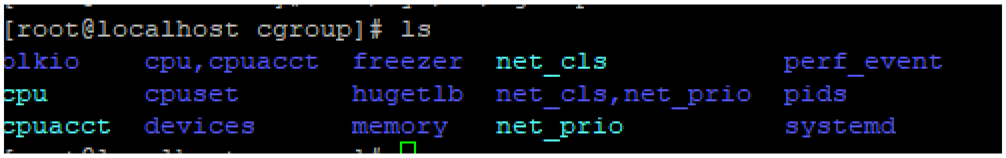
In this study, it has been demonstrated that isolation can be implemented at different levels by changing the Kernel in a number of ways:

i. Tweaking the system configurations
ii. Reconfiguration of Kernel Primitives for example the user file structure and Process ID Generation by modifying the PID generation source code

6.3 Future Work

In this study, comprehensive implications of using predesignated PIDs as opposed to random PIDs for process tracking was not explored. Further work is required in the following areas:

i. Research to determine the performance and optimality tradeoff between using predesignated PIs for ease of process identification and the ease and flexibility of using random PIDs as is currently implemented.
ii. cGroups for Resource control is limited to below resources. These are not the only resources that need to be controlled in terms of usage. There is need for further studies to identify and incorporate additional control features that can be used to support multi tenancy
iii. The main limitation is how to define a range of PIDs defined in the cgroup. In this study, only sample PIDs were used. A way needs to be devised to allow for definition of a range of PIDs as opposed to listing individual PIDs.
iv. The number of allocated PIDs is a potential resource metric that can be applied in future. This is a recommended area for further research



```
[root@localhost cgroup]# ls
blkio    cpu,cpuacct  freezer  net_cls           perf_event
cpu      cpuset       hugetlb  net_cls,net_prio  pids
cpuacct  devices      memory   net_prio          systemd
```

GLOSSARY & ACRONYMS

Server: It is any combination of hardware or software designed to provide services to clients.

Client: It requests and consumes the services provided by another having the role of server.

Virtualization: It is the ability to separate the OS from the hardware that operates it.

Private Cloud: It is an approach for designing, implementing and managing servers, applications and data center resources by reducing complexity, increasing standardization and automation, and provide elasticity.

VM (Virtual Machine): this is a virtual server

VMM – Virtual Machine Monitor

OS – Operating System

LAN Local Area Network

WAN : Wide Area Network

API – Application Programming Interface

VM- Virtual Machine

BIOS – Basic Input /Output System

GNU – Group Not Unix

REFERENCES

1. Thomas Erl with Zaigham Mahmood and Ricado Puttini: Cloud Computing Concepts, Technology and Architecture
2. Evi Nemeth.Garth Synder.Trent R Hein.Dan Mackin: Unix and Linux Administration Handbook
3. Connor, D. (2004). Server virtualization is on the rise. Network World Canada, 14(23), 18.
4. Conroy, S. (2018, January 25).
5. History of virtualization: https://www.idkrtm.com/history-of-virtualization/.
6. Dawson, P., & Bittman, T. J. (2008). Virtualization changes virtually everything. Gartner Special Report.
7. Dua, R., Raja, A. R., & Kakadia, D. (2014, March). Virtualization vs containerization to support paas. In Cloud Engineering
8. Politecnico Di Torino: OS-level virtualization with Linux containers: process isolation mechanisms and performance analysis of last generation container runtimes
9. (IC2E), 2014 IEEE International Conference on(pp. 610-614). IEEE.
10. Firesmith. (2017, September 25). Virtualization via Containers. Retrieved from https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-containers.html.
11. KamyabKhajehei "Role of virtualization in cloud computing,"International Journal of Advance Research
12. Bovet . Daniel P. AND Marco Cesati: Understanding the Linux Kernel (3rd Edition),
13. Love. Robert: Linux Kernel Development (3rd Edition)
14. Linux Online Community: LWN.net
15. International Journal Of Scientific & Technology Research Volume 3, Issue 11, November 2014: A Study On Virtualization Techniques And Challenges In Cloud Computing: http://www.ijstr.org/final-print/nov2014/A-Study-On-Virtualization-Techniques-And-Challenges-In-Cloud-Computing.pdf
16. A-Study-On-Virtualization-Techniques-And-Challenges-In-Cloud-Computing.pdf (ijstr.org)
17. Developers: Separation Anxiety: A Tutorial for Isolating Your System with Linux Namespaces: https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces
18. Mr Akshay Ghanashyam Sawant , Prof. Prasanna Rajaram Rasal: Research on the Virtualization Technology in Cloud Computing Environment: https://www.jetir.org/papers/JETIR2106731.pdf
19. TechTarget: bare-metal cloud: https://www.techtarget.com/searchstorage/definition/bare-metal-cloud
20. Leonardo Passos, Rodrigo QueirozLeonardo Passos, Rodrigo Queiroz et. al: A Study of Feature Scattering in the Linux Kernel
21. Joshua S. White, Adam W. Pilbeam, 2010 "A Survey of Virtualization Technologies With Performance Testing"
22. Padhy, Rabi & Patra, Manas & Satapathy, Suresh. (2020). "virtualization techniques & technologies: state-of-the-art"
23. [16] Glenn Willen, Mike Cui, 15-410 Fall 2006 "Virtualization" http://www.cs.cmu.edu/~410-f06/lectures/L31_Virtualization.pdf

24. Mount namespace
https://man7.org/linux/man-pages/man7/mount_namespaces.7.html
25. IPC namespace
https://man7.org/linux/man-pages/man7/ipc_namespaces.7.html
26. Time namespace
https://man7.org/linux/man-pages/man7/time_namespaces.7.html
27. Paul Menage, “Cgroups“, Linux Kernel documentation
https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt
28. Tejun Heo, “Control Group v2“, Linux Kernel documentation
https://www.kernel.org/doc/Documentation/cgroup-v2.txt
29. Michael Barcarella, 2002, “Taking advantage of Linux capabilities “, Linux Journal
30. Adrian Mouat, 2019, “Linux Capabilities: Why They Exist and How They Work“
31. Chroot jail escape example
https://filippo.io/escaping-a-chroot-jail-slash-1/
32. getpid, Linux manual
https://man7.org/linux/man-pages/man2/getpid.2.html
33. Namespaces
https://man7.org/linux/man-pages/man7/namespaces.7.html
34. PID namespace
https://man7.org/linux/man-pages/man7/pid_namespaces.7.html
35. "Linux Kernel Namespace Implementation: Introduction to namespace API"
https://titanwolf.org/Network/Articles/Article?AID=740fc46c-d325-
4c22-a720-b5c259551c87#gsc.tab=0
36. Nsproxy reference count
https://github.com/torvalds/linux/blob/master/include/linux/
nsproxy.h
37. Process Namespace, Mahmud Ridwan
https://www.toptal.com/linux/separation-anxiety-isolating-yoursystem-
with-linux-namespaces
38. Linux kernel
https://github.com/torvalds/linux
39. Clone Linux man
40. https://man7.org/linux/man-pages/man2/clone.2.html
41. Net Solutions: 7 Challenges in Multi-Tenancy Testing and Their Solutions:
https://www.netsolutions.com/insights/multi-tenancy-testing-top-challenges-and-solutions/
42. Leonardo Passos, Rodrigo Queiroz, Mukelabai Mukelabai, Thorsten Berger, Sven Apel, Krzysztof Czarnecki, and Jesus Alejandro Padilla: A Study of Feature Scattering in the Linux Kernel: https://www.infosun.fim.uni-passau.de/publications/docs/PQM+18.pdf
43. NginX: What Are Namespaces and cgroups, and How Do They Work?:
https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/
44. TutorialsPoint: Linux Admin - Resource Mgmt with crgoups:
https://www.tutorialspoint.com/linux_admin/linux_admin_resource_mgmt_with_crgoups.ht
m
45. TutorialsPoint: How to run a script on startup in Linux:

https://www.tutorialspoint.com/run-a-script-on-startup-in-linux#:~:text=Make%20the%20script%20file%20executable,scriptname%20defaults%22%20in%20the%20terminal.

46. Engineers Garage: How to run a python code on boot- (Part 7/12): https://www.engineersgarage.com/how-to-run-a-python-code-on-boot-part-7-12/

47. Github: Python Script to automatically generate a bootable Image file with a specifiable partition table for embedded Linux distributions: https://github.com/robseb/LinuxBootImageFileGenerator

48. Linux HandBook: How to create a systemd service in Linux: https://linuxhandbook.com/create-systemd-services/

49. WikiPedia: Kernel (operating system) https://en.wikipedia.org/wiki/Kernel_(operating_system)#:~:text=The%20kernel%20performs%20its%20tasks,area%20of%20memory%2C%20user%20space.
50. Science Direct: Computer Science Resource Isolation: https://www.sciencedirect.com/topics/computer-science/resource-isolation
51. Linux Questions Organization: https://www.linuxquestions.org/linux/answers ; https://www.linuxquestions.org/questions/showthread.php?p=6443994#post6443994
52. The Python Code: How to Execute Shell Commands in a Remote Machine in Python https://www.thepythoncode.com/article/executing-bash-commands-remotely-in-python
53. Digital Ocean: How to Connect to a Linux terminal from Web browser: https://www.digitalocean.com/community/tutorials/how-to-connect-to-a-terminal-from-your-browser-using-python-webssh
54. Educative: How to run a Python script in Linux: https://www.educative.io/answers/how-to-run-a-python-script-in-linux
55. Wikipedia: Linux Namespaces: https://en.wikipedia.org/wiki/Linux_namespaces
56. Wikipedia: Linux Kernel Features: https://en.wikipedia.org/wiki/Category:Linux_kernel_features
57. Kir Kolyshkin: Containers and Namespaces in Linux Kernel: https://events.static.linuxfound.org/slides/lfcs2010_kolyshkin.pdf
58. Proceedings of NetDev 1.1: The Technical Conference on Linux Networking (February 10th-12th 2016. Seville, Spain): https://www.netdevconf.org/1.1/proceedings/slides/rosen-namespaces-cgroups-lxc.pdf
59. Rami Rosen: Resource management: Linux kernel Namespaces and cgroups: http://www.haifux.org/lectures/299/netLec7.pdf
60. CloudFare: Using Go as a scripting language in Linux: https://blog.cloudflare.com/using-go-as-a-scripting-language-in-linux/
61. Wikipedia: Category: Interfaces of the Linux Kernel:

https://en.wikipedia.org/wiki/Category:Interfaces_of_the_Linux_kernel

62. Geeksforgeeks: Add a User in Linux using Python Script: https://www.geeksforgeeks.org/add-a-user-in-linux-using-python-script/
63. IBM: Enforcing IBM® Spectrum LSF job memory and swap with Linux cgroups: https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=tips-enforcing-job-memory-swap-linux-cgroups
64. Selectel: Containerization Mechanisms: Namespaces: https://selectel.ru/blog/en/2017/03/09/containerization-mechanisms-namespaces/
65. IBM: “Topics: IaaS, PaaS & SaaS”: https://www.ibm.com/topics/iaas-paas-saas
66. IBM: “What is cloud computing?.” https://www.ibm.com/cloud/learn/cloud-computing

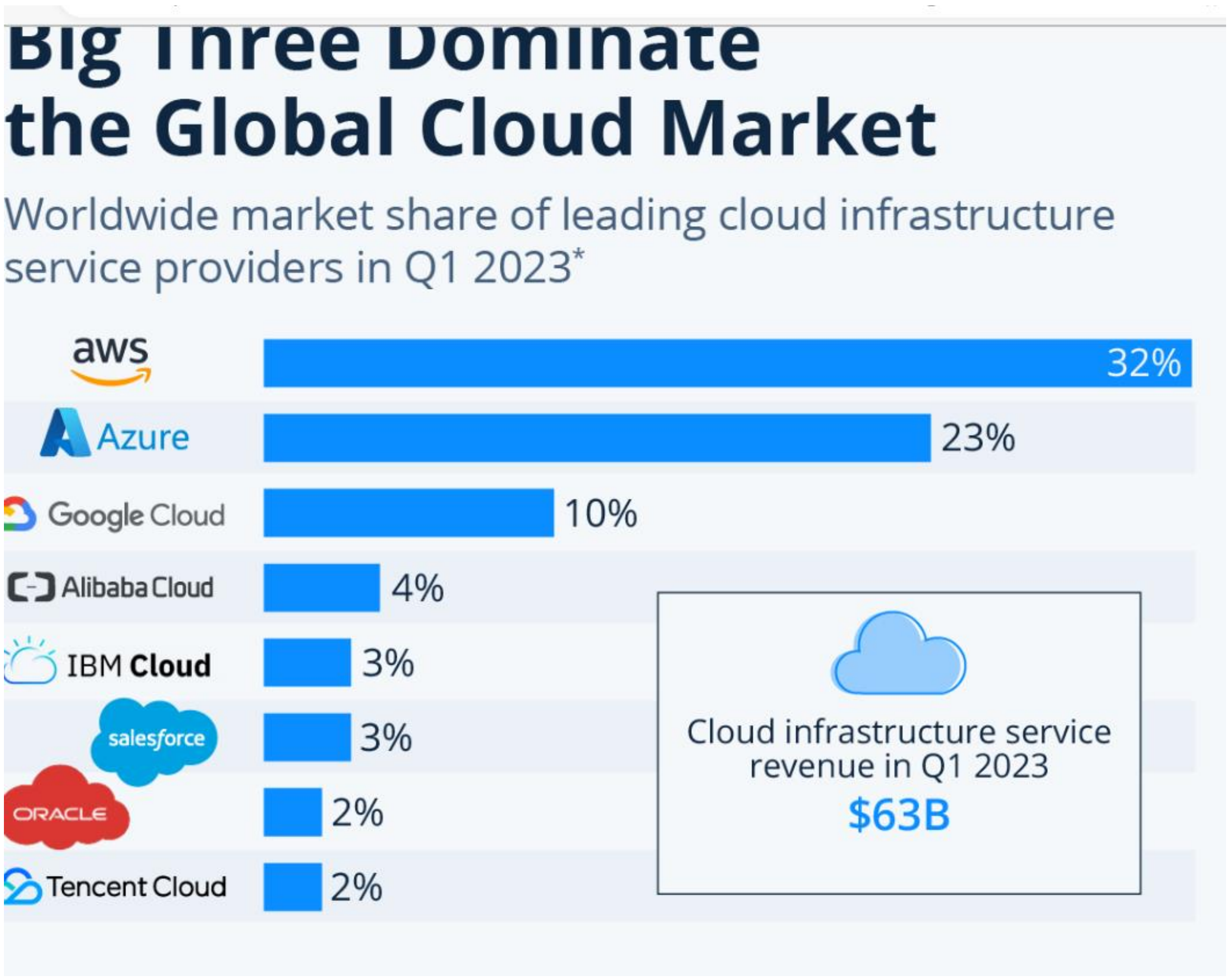
67. Politecnico Di Torino: Masters Thesis: OS-level virtualization with Linux containers: process isolation mechanisms and performance analysis of last generation container runtimes: 2019-2020
68. Leonardo Passos et. al: Study of Feature Scattering in the Linux Kernel

69. Linux Kernel Documentation: Namespaces compatibility list — The Linux Kernel documentation

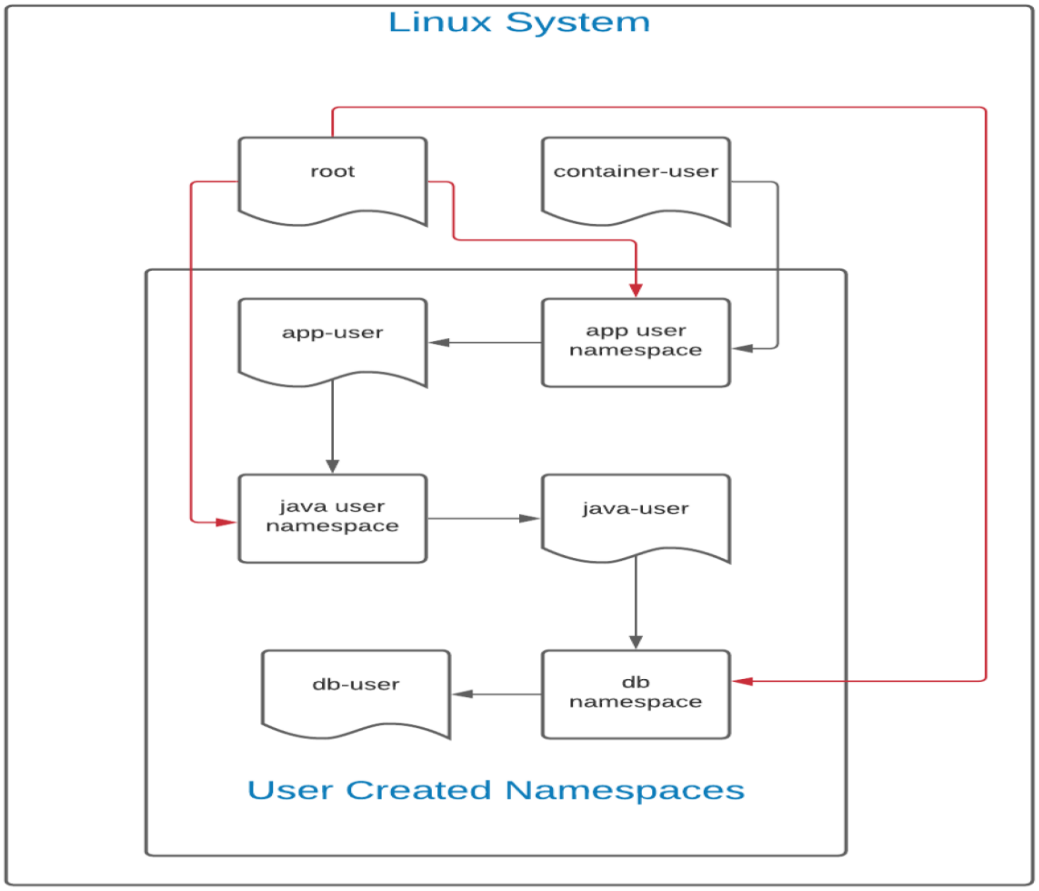
70. Cloud Multi-Tenancy: Issues and Developments: Session: UCIoT 2017 Workshop Presentation
71. Hussain Al-Jahdali, Abdulaziz Albatli, Peter Garraghan, Paul Townend, Lydia Lau, and JieXu (2014). “Multi-tenancy in cloud computing,” In proceedings of the 8th IEEE International symposium on service-oriented system engineering.
72. Multi Tenancy in Cloud Computing: 2014 IEEE 8th International Symposium on Service Oriented System Engineering
73. NIST Definition of Cloud Computing https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
74. Kbuild, “The kernel build infrastructure,” www.kernel.org/doc/Documentation/kbuild, last seen: Feb. 14th, 2015.
75. The Linux Kernel MakeFile: https://lwn.net/Articles/21835/

APPENDICES

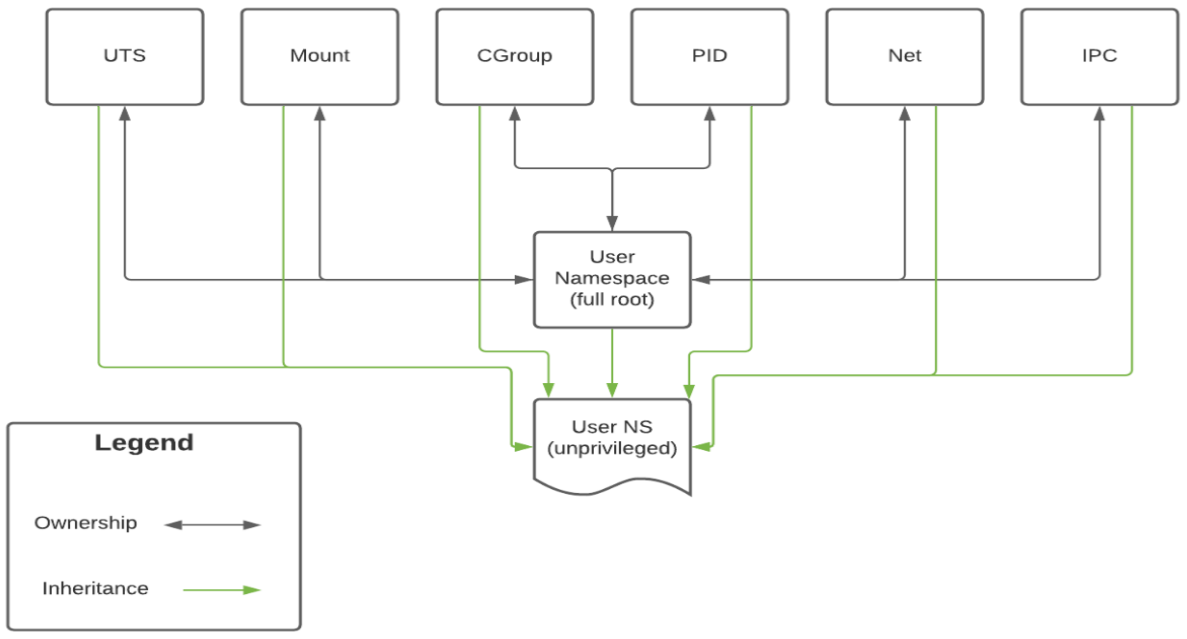
Appendix I – Cloud Providers Market Share



Appendix II – Linux System User Namespace

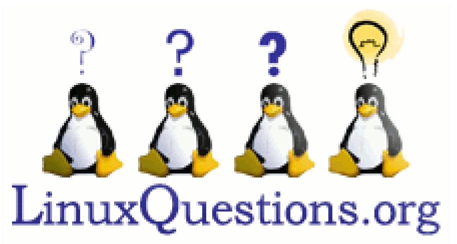

Linux System
root
container-user
app-user
app user namespace
java user namespace
java-user
db-user
db namespace
User Created Namespaces

Appendix III – Supported Namespaces



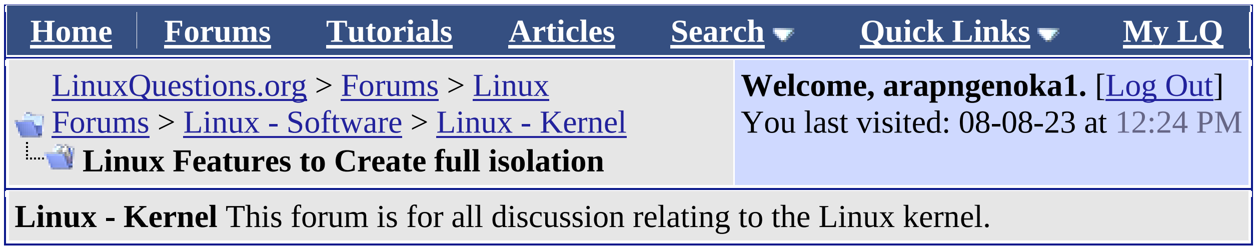
Appendix IV – Inputs from Linux Community

There was not much help from Linux community because the community seems to primarily focus on extensions and improvements but re-devising things perhaps in fidelity to the mantra *"if ain't broken, do not fix it"*



Welcome to the most active **Linux Forum** on the web.

1.



Home | **Forums** | **Tutorials** | **Articles** | **Search** | **Quick Links** | **My LQ**

LinuxQuestions.org > Forums > Linux Forums > Linux - Software > Linux - Kernel
Linux Features to Create full isolation

Welcome, arapngenoka1. [Log Out]
You last visited: 08-08-23 at 12:24 PM

Linux - Kernel This forum is for all discussion relating to the Linux kernel.

Please Mark this thread as solved if you feel a solution has been provided.

Post Reply

	Thread Tools	Search this Thread	Rate Thread

07-24-23, 08:58 AM #1

arapngenoka1

LQ Newbie

Registered: Jul 2023

Posts: 2

Rep:

Linux Features to Create full isolation

I have 2 questions:

1. Where in cgroups.c do I modify to create another field/flag for maximum allowed size so that I provide for a range?

2. I am looking to achieve full isolation without virtualization. I have explored that this can be achieved with namespaces. How Can I merge all namespaces to create one global setting.

Report Quote

07-24-23, 10:41 AM #2

pan64

LQ Addict

Registered: Mar 2012

Location: Hungary

Distribution: debian/ubuntu/suse ...

Posts: 20,640

Rep:

Hi,
where are these questions coming from?
see LQ rules: https://www.linuxquestions.org/linux/rules.html

Quote:

> We're happy to assist if you have specific questions or have hit a stumbling point, however. Let us know what you've already tried and what references you have used (including class notes, books, and searches) and we'll do our best to help. Keep in mind that your instructor might also be an LQ member.

Otherwise would be nice to define what kind of size and range do you mean?
What do you mean by full isolation?
What do you mean by merging namespaces?

A program will never do what you wish but what was implemented!

Happy with solution ... mark as **[SOLVED]**
If you really want to say **thanks** => *click on* Yes *(bottom right corner).*

Report Quote

Did you find this post helpful? Yes

08-02-23, 12:54 PM #3

arapngenoka1

LQ Newbie

Registered: Jul 2023

Posts: 2

Original Poster

Rep:

Thanks for your response.

I am sorry, my asks may have been unclear. Allow me to reframe the questions. What I am looking to achieve is to extend kernel features in more ways than a namespacing tool -e.g. containerization tool. Currently, in cgroups, only a hard limit is set. I wish to set a low and highest limit and define some scaling rules so that a process can scale resources to a maximum limit subject to available resources and without suffocating other processes

I will appreciate your guidance

Report Quote

08-03-23, 01:28 AM #4

pan64

LQ Addict

Registered: Mar 2012

Location: Hungary

Distribution: debian/ubuntu/suse ...

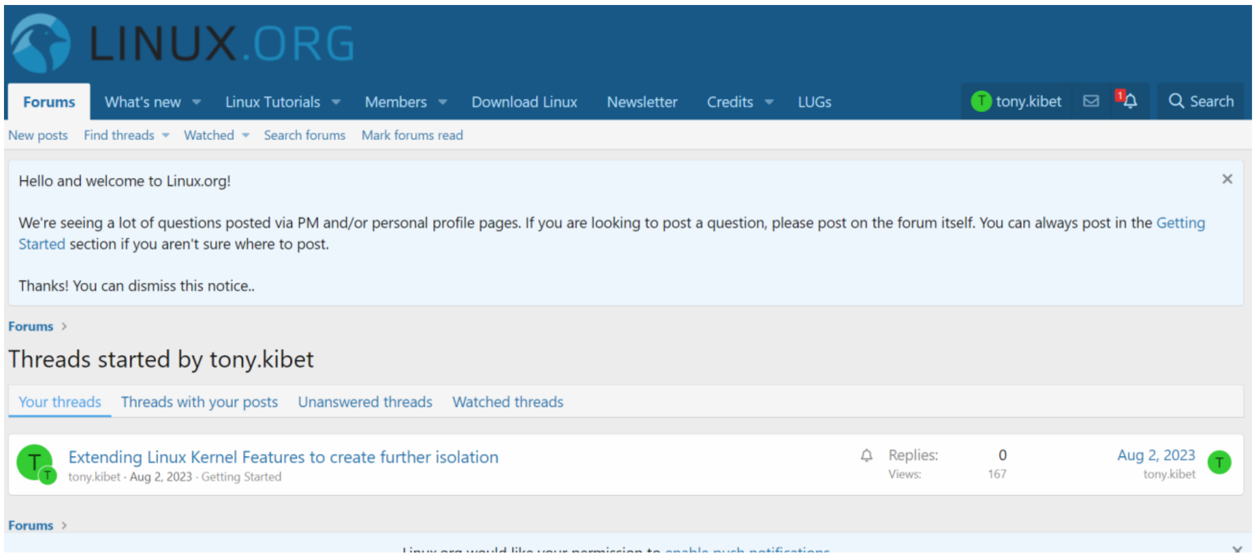
Posts: 20,640

that sounds good. The usual question is what have you done so far, where did you stuck, what kind of help do you need at all?

A program will never do what you wish but what was implemented!

Happy with solution ... mark as **[SOLVED]**
If you really want to say **thanks** => *click on* Yes *(bottom right corner).*

Rep: 🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩🟩	



Hello and welcome to Linux.org!

We're seeing a lot of questions posted via PM and/or personal profile pages. If you are looking to post a question, please post on the forum itself. You can always post in the Getting Started section if you aren't sure where to post.

Thanks! You can dismiss this notice..

Forums >

Threads started by tony.kibet

Your threads | Threads with your posts | Unanswered threads | Watched threads

Extending Linux Kernel Features to create further isolation tony.kibet · Aug 2, 2023 · Getting Started	Replies: 0 Views: 167	Aug 2, 2023 tony.kibet	

Forums >

Appendix V– Results of Forked process – showing the PID numbers – for default process

```
Process ID: 6771

Parent Process ID: 1

Process ID for Process: 8  PID Is: 6831

Process ID for Process: 9  PID Is: 6837

Process ID for Process: 9  PID Is: 6839

Process ID for Process: 9  PID Is: 6891
Process ID for Process: 10  PID Is: 6892

Process ID for Process: 10  PID Is: 6894

Process ID for Process: 10  PID Is: 6893

Process ID: 6761

Parent Process ID: 1

Process ID: 6755

Parent Process ID: 1

Process ID: 6555

Parent Process ID: 1

Process ID: 6765

Parent Process ID: 6608

Parent Process ID: 1

Process ID: 6892

Parent Process ID: 1

Process ID: 6893

Process ID: 6772

Process ID: 6783

Parent Process ID: 6606

Parent Process ID: 1

Parent Process ID: 6654

Process ID: 6775

Parent Process ID: 1

Parent Process ID: 1
Parent Process ID: 6660

Parent Process ID: 6662

Process ID for Process: 10  PID Is: 6895

Process ID: 6891

Process ID: 6798

Parent Process ID: 1

Parent Process ID: 6660

Parent Process ID: 6605

Process ID: 6803

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6860

Process ID: 6806

Parent Process ID: 6708

Process ID: 6894

Parent Process ID: 1

Process ID: 6895

Parent Process ID: 6743

Parent Process ID: 1

Parent Process ID: 6685

Process ID for Process: 10  PID Is: 6861

Process ID: 6565

Parent Process ID: 1

Parent Process ID: 1
Parent Process ID: 1

Process ID: 6818

Process ID: 6770

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 6751

Parent Process ID: 6425

Process ID: 6829

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 6695

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 6555

Process ID: 6847

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 1

Process ID for Process: 9  PID Is: 6882

Process ID: 6865

Parent Process ID: 6814
```

```
Process ID for Process: 10  PID Is: 6896

Process ID: 6859

Parent Process ID: 1

Process ID for Process: 9  PID Is: 6878

Process ID: 6845

Parent Process ID: 6772

Process ID: 6830

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6897

Process ID: 6852

Parent Process ID: 1

Process ID: 6897

Parent Process ID: 6852

Process ID: 6896

Parent Process ID: 1

Process ID: 6868

Process ID: 6849

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6883

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6884

Process ID: 6833

Parent Process ID: 1
```

```
Process ID: 6866

Parent Process ID: 1

Process ID: 6869

Process ID: 6835

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6885

Parent Process ID: 1

Process ID: 6857

Parent Process ID: 1

Process ID: 6867

Parent Process ID: 1

Process ID: 6875

Process ID: 6861

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6886

Parent Process ID: 6565

Process ID: 6840

Parent Process ID: 1

Process ID: 6877

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6887

Process ID: 6864

Process ID: 6842
Process ID: 6871

Parent Process ID: 1

Parent Process ID: 1

Parent Process ID: 1

Process ID: 6872

Parent Process ID: 1

Process ID for Process: 10  PID Is: 6888

Process ID: 6876

Process ID: 6843

Parent Process ID: 1

Process ID: 6879

Parent Process ID: 1

Parent Process ID: 1

Process ID: 6878

Parent Process ID: 1

Process ID: 6881

Parent Process ID: 1

Process ID: 6846

Process ID for Process: 10  PID Is: 6889

Parent Process ID: 1

Process ID: 6880

Parent Process ID: 1

Process ID: 6873
```

```
Parent Process ID: 1

Process ID: 6887

Process ID: 6851

Parent Process ID: 6871

Parent Process ID: 1

Process ID: 6874

Process ID: 6884

Parent Process ID: 1

Parent Process ID: 6866

Process ID: 6885

Process ID: 6886

Parent Process ID: 6857

Parent Process ID: 6840

Process ID for Process: 10  PID Is: 6890

Process ID: 6882

Parent Process ID: 1

Process ID: 6889

Process ID: 6888

Parent Process ID: 6873

Parent Process ID: 6878

Process ID: 6883

Parent Process ID: 6833

Process ID: 6863
```

```
Parent Process ID: 1

Process ID: 6890

Process ID: 6870

Parent Process ID: 6882

Parent Process ID: 1
```

Appendix VI – Results of Forked process – showing the PID numbers – for tenant assigned a PID Offset of 80000

```
Process ID for Process: 10  PID Is: 80000

Parent Process ID: 1

Process ID: 8298

Parent Process ID: 1

Process ID: 8302

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8238

Parent Process ID: 1

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 8  PID Is: 80000

Process ID for Process: 10  PID Is: 88456

Process ID for Process: 8  PID Is: 80000

Process ID for Process: 9  PID Is: 88457

Process ID: 8300
```

```
Parent Process ID: 1

Process ID for Process: 9  PID Is: 88458

Process ID for Process: 10  PID Is: 88459

Process ID: 8295

Parent Process ID: 1

Process ID: 8453

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88460

Process ID: 8433

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8242

Parent Process ID: 1

Process ID: 8301

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88461

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8304

Parent Process ID: 1
```

```
Process ID: 8459

Parent Process ID: 8295

Process ID: 8303

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8454

Process ID: 8456

Process ID for Process: 10  PID Is: 80000

Parent Process ID: 1

Process ID for Process: 9  PID Is: 80000

Process ID: 8245

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88462

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8458

Parent Process ID: 1

Process ID: 8308

Process ID: 8306

Parent Process ID: 1

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88463
Process ID: 8305

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8248

Parent Process ID: 1

Process ID: 8462

Parent Process ID: 1

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8311

Parent Process ID: 1

Process ID: 8310

Parent Process ID: 1

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 10  PID Is: 88464

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 10  PID Is: 88466

Process ID for Process: 10  PID Is: 88465

Process ID for Process: 10  PID Is: 80000

Process ID: 8257
```

```
Parent Process ID: 1

Process ID: 8307

Parent Process ID: 1

Process ID: 8463

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 9  PID Is: 80000

Process ID: 8312

Parent Process ID: 1

Process ID: 8266

Parent Process ID: 1

Process ID: 8309

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88467

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 8  PID Is: 80000

Process ID: 8264

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8313

Parent Process ID: 1
```

```
Process ID: 8299

Parent Process ID: 1

Process ID for Process: 9  PID Is: 88468

Process ID: 8457

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88315

Process ID for Process: 10  PID Is: 88469

Process ID: 7963

Parent Process ID: 1

Process ID: 8280

Process ID for Process: 10  PID Is: 80000

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8467

Parent Process ID: 1

Process ID: 8469

Parent Process ID: 8280

Process ID for Process: 9  PID Is: 88314

Process ID for Process: 10  PID Is: 88470

Process ID: 7980

Process ID: 8327

Parent Process ID: 1

Parent Process ID: 1
```

```
Process ID: 7956

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 88336

Process ID for Process: 10  PID Is: 88357

Process ID: 8326

Process ID: 8350

Parent Process ID: 1

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8466

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88361

Process ID: 8197

Process ID: 8410

Parent Process ID: 1

Process ID for Process: 9  PID Is: 80000

Parent Process ID: 1

Process ID: 7759

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88471

Process ID for Process: 8  PID Is: 80000

Process ID: 8378
```

```
Parent Process ID: 1

Process ID for Process: 9  PID Is: 88472

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 88473

Process ID for Process: 10  PID Is: 80000

Process ID: 8465

Parent Process ID: 1

Process ID: 8412

Parent Process ID: 1

Process ID: 8471

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8379

Parent Process ID: 1

Process ID: 8473

Process ID: 8418

Parent Process ID: 1

Parent Process ID: 1

Process ID: 8417

Parent Process ID: 1

Process ID for Process: 9  PID Is: 80000
Process ID for Process: 10  PID Is: 80000

Process ID for Process: 9  PID Is: 80000

Process ID: 8357

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88474

Process ID for Process: 10  PID Is: 88475

Process ID for Process: 9  PID Is: 80000

Process ID: 8413

Parent Process ID: 1

Process ID: 8416

Parent Process ID: 1

Process ID for Process: 10  PID Is: 88476

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID: 8414

Parent Process ID: 1

Process ID: 8338

Parent Process ID: 1

Process ID: 8422

Parent Process ID: 1

Process ID: 8464

Parent Process ID: 1
```

```
Process ID: 8420

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8472

Parent Process ID: 1

Process ID: 8424

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8225

Process ID: 8470

Parent Process ID: 1

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID: 8475

Parent Process ID: 1

Process ID: 8284

Process ID for Process: 10  PID Is: 80000

Parent Process ID: 1

Process ID: 8460

Parent Process ID: 1

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000

Process ID for Process: 10  PID Is: 80000
Process ID: 8476

Parent Process ID: 1

Process ID: 8461

Parent Process ID: 1

Process ID: 8474

Parent Process ID: 1

Process ID for Process: 9  PID Is: 80000

Process ID for Process: 10  PID Is: 88477

Process ID for Process: 10  PID Is: 80000

Process ID: 8468

Parent Process ID: 1

Process ID: 8477

Parent Process ID: 8468

Process ID for Process: 10  PID Is: 80000

Process ID: 8455

Parent Process ID: 1
```