# A SURVEY ON OPERATING SYSTEM VIRTUALIZATION METHODS AND CHALLENGES

By

ABHILASH C.B * D.V. ASHOKA **

* Assistant Professor, Department of Computer Science and Engineering, JSS Academy of Technical Education (VTU), Bangalore, India.
** Professor, Department of Information Science and Engineering, JSS Academy of Technical Education (VTU), Bangalore, India.

*ABSTRACT*

*Computational world is turning out to be substantial and complex. Distributed computing has risen as a well registering model to bolster handling substantial volumetric information utilizing groups of product PCs. Working framework (OS) virtualization can give various imperative advantages, including straightforward relocation of utilizations, server combination, online OS upkeep, and improved framework security. Nonetheless, the development of such a framework introduces a bunch of difficulties, not withstanding for the most wary engineer, that if neglected may bring about a frail, deficient virtualization. We exhibit exchange of key execution issues in giving OS virtualization in a merchandise OS, including framework call intervention, virtualization state administration, and race conditions. The authors discussed about their encounters in executing such usefulness over two note worthy variants of Linux altogether in a loadable bit module with no portion adjustment. The author exhibit trial results on both uniprocessor and multiprocessor frameworks that show the capacity of our way to deal with furnish recapture virtualization with low overhead. In this paper, the authors first developed a comprehensive taxonomy for describing operating system architecture. Then they use this taxonomy to survey several existing operating system virtualization services and challenges.*

*Keywords: Operating System, Virtualization, Uniprocess, Multi Process.*

## INTRODUCTION

PCs have ended up pervasive in scholarly, corporate, and government associations as exponential scaling laws have made PCs speedier, less expensive, and progressively joined. In the meantime, the broad utilization of PCs has offered ascent to tremendous administration many-sided quality and security dangers. Virtualization has risen as a key innovation for tending to these issues.

Virtualization basically acquaints a level of indirection with a framework to decouple applications from the fundamental host framework. This decoupling can be utilized to give critical properties, for example, separation and versatility, giving a heap of helpful advantages. These advantages incorporate sup-isolating so as to pore server union applications from each other while having the same machine, enhanced system security by disconnecting defenseless applications from other mission basic applications migrating so as to run on the same machine, issue flexibility applications of flawed hosts, element burden migrating. To adjust applications to less stacked has and enhanced administration accessibility and organization by relocating applications before host up keep, so they can keep on running with negligible downtime.

While virtualization can be performed at various diverse levels of reflection, giving virtualization at the right level to straight forwardly bolster unmodified applications is significant to empower arrangement and boundless utilization. The two primary methodologies for giving application straight forward virtualization are; equipment virtualization and working framework virtualization. Equipment virtualization methods virtualize the under-lying equipment construction modeling utilizing a virtual machine screen to decouple OS from the equipment so that, a whole OS environment and related applications can be executed in a virtualized domain. OS virtualization strategies virtualize the OS to decouple applications from the OS, so that individual applications can be executed in virtualized situations. Equipment and OS virtualization

systems give their advantages and can give correlative usefulness.

OS virtualization gives another granularity of control at the level of individual procedures or applications, which is more useful than the equipment virtualization reflection that works with whole OS occurrences. For instance, OS virtualization can empower straight forward movement of individual applications, not only relocation of whole OS occasions. This granularity relocation gives more noteworthy adaptability and results in lower overhead. Moreover, if the working framework obliges up keep, OS virtualization can be utilized to relocate the discriminating applications to another running working framework occurrence. By decoupling applications from the OS occurrence, OS virtualization empowers the basic OS to be fixed and overhauled in an auspicious way with insignificant effect on the accessibility of utilization administrations. Equipment virtualization alone can't give this usefulness, since it binds applications to an OS occurrence, and ware working frameworks unavoidably bring about down-time because of essential upkeep and security upgrades.

Given the advantages of OS virtualization, contemporary OSs are progressively intrigued by giving backing to it [1]. While a hefty portion of the ideas driving OS virtualization have been talked about in subtle element in past work, little consideration has been given to seeing how to really execute it practically speaking. Most work has concentrated on more elevated amount issues without respect for a hefty portion of the unpretentious issues and execution challenges in making OS virtualization work accurately for unmodified applications and thing OSs. While some work has concentrated on more elevated amount usage contemplations with respect to security in OS virtualization [2], we are not mindful of any past work that considers execution issues in giving more finish OS virtualization, for example, in the setting of straightforward application relocation.

The authors introduce a point by point discourse of key execution issues and difficulties in giving OS virtualization in a merchandise OS. We think about choices for executing OS virtualization at client level versus portion level, talk about execution costs for routines for putting away virtualization state, and analyze unobtrusive race conditions that can emerge in actualizing OS virtualization. Some OSs are progressively making so as to fuse virtualization support pervasive changes to the OS piece [3]. The authors depict a methodology of actualizing OS virtualization in an insignificantly obtrusive way by regarding the OS bit as an unmodified black box. The encounters from this methodology are instrumental in showing how OS virtualization can be consolidated into product OSs with negligible changes. Utilizing this methodology, we have actualized a Linux OS virtualization model totally in a loadable piece module. The authors show quantitative results exhibiting that such a negligibly obtrusive methodology should be possible with low overhead.

## 1. Virtualization Concepts

OS virtualization disconnects forms inside of a virtual execution environment by checking their association with the basic OS example. Like equipment virtualization [4], applications that keep running inside of the virtual environment ought to show an impact indistinguishable to that exhibited as though they had been keep running on the unvirtualized framework. Furthermore, a factually prevailing subset of the applications association with framework assets ought to be immediate to minimize overhead.

OS virtualization approaches are grouped along two measurements, host-autonomy and fulfillment. Host-subordinate virtualization just confines forms while host-free virtualization additionally decouples them. The qualification is that, host-subordinate virtualization just squares or lifters out the namespace between procedures, while host-free virtualization gives a private virtual namespace to the applications' referenced OS assets. The previous does not bolster straightforward application relocation since the absence of asset interpretation tables orders that the asset identifiers of an application stay static crosswise over hosts for a mi-grinding procedure, which can prompt identifier clashes when moving between hosts. Samples of host-ward

virtualization incorporate Linux VServers and Solaris Zones [5]. Host-free virtualization exemplifies forms in a private namespace that deciphers asset identifiers from any host to the private identifiers expected by the moving application. Illustrations of this methodology incorporate Zap [6] and Capsules [7]. The authors allude to these virtual private names-pace as a case, taking into account the wording utilized as a part of Zap.

As far as fulfillment, incomplete virtualization virtualizes just a subset of OS assets. The most widely recognized sample of this is virtual memory, which furnishes every procedure with its own particular private memory namespace yet doesn't virtualize some other OS assets. Table 1 shows the Kernel Subsystems.

Inside of this scientific categorization of virtualization methodologies, finish and host-free virtualization gives the broadest scope of usefulness, which incorporates giving the important backing to both separation and relocation of utilizations. An extra qualification between the scientific categorizations is in the application's extent regarding the accessible frameworks. Virtualization approaches that are host-subordinate and/or halfway give advantages just on a solitary host, while complete, host-autonomous virtualization approaches master vide the backing for applications to abuse the accessible frameworks that are available to the whole association. The rest of this paper concentrates on the requests of supporting this more broad type of virtualization in the setting of product OSs.

## 2. Virtualization Methods

To bolster private virtual namespaces, instruments must be given to decipher between the case's asset identifiers and the working framework asset identifiers. For each asset got to by a procedure in a case, the virtualization layer partners a virtual name to a proper OS physical name. At the point when an OS asset is made for a procedure in a unit, the physical name returned by the framework is discovered, and a relating private virtual name is made and re-swung to the procedure. Likewise, at whatever time a procedure passes a virtual name to the working framework, the virtualization layer gets and replaces it with the relating physical name. To empower this interpretation, an instrument must be utilized that diverts the typical control stream of the framework so that, the private virtual namespaces are utilized as opposed to the default physical namespace.

| Subsystem | Description |
|---|---|
| Process ID | PID and related IDs: thread group, process group, session |
| Filesystem | Filesystem root (chroot) |
| SysV IPC | ID and KEY of message queues, semaphores, and shared memory |
| Unix IPC | Unix domain sockets, pipes, named pipes |
| Network | Internet domain sockets |
| Devices | Device specific resources |
| Pseudo terminals | PTS IDs and devpts pseudo lesystem |
| Pseudo systems | E.g. procfs, devpts, shmfs |
| Miscellaneous | Hostname, user/group ID, system name |

Table 1. Kernel Subsystems and Related Resources

Intervention is the key system that can give the imperative redirection expected to virtualization of namespaces. In our setting, mediation catches occasions between face in the middle of uses and the OS and performs some handling on those occasions before passing them down to the OS or up to the applications. The intervention that should be defeated actualizing OS virtualization obliges that some preprocessing be done before the local piece usefulness is executed, and some post-preparing be done after the local part usefulness is executed. The mediation usage itself is expert by wrapping the current framework calls with the capacities and deciphering between virtual names and physical names prior and then afterward the first framework call is summoned.

Framework call mediation can be actualized at

| Method | Description |
|---|---|
| System-wide hash table | Convert physical host identifiers to virtual pod identifiers |
| Per-pod hash table | Convert virtual pod identifiers to physical host identifiers |
| Direct reference | Per -process fast reference to augmented virtualization state |
| PID reference count | Protect PIDs of processes that insides or outside pods from reuse |
| in-pod process | Indicate that a process is running inside a pod |
| init-pending process | Indicate that a process in a pod is pending initialization |
| Outside-pod table | Track identifiers used by processes running outside pods |
| Restricted-ID table | Track identifiers without a reference count that are in use |
| init-complete process | Indicate that the virtualization state of a resource has been initialized |
| Filesystem stacking | Virtualize per-pod pseudo view |

Table 2. Virtualization Methods

distinctive layers of the framework. We support utilizing the loadable portion module innovation that is presently accessible with all significant ware OSs. A part module can give application-straightforward virtualization without base bit changes and without giving up versatility and execution. Moreover, by working in advantaged mode, virtualization can give the security important to guarantee right confinement. By working at the level of piece modules, the virtualization module can use the arrangement of sent out portion subroutines, which is a very much characterized interface. Utilizing the bit API likewise means a sure level of convenience and soundness in the execution, since changes in the piece API are rare.

At the end of the day, virtualization versatility is shielded to an expansive degree from part changes in a comparative manner as legacy applications are ensured.

There are different ways to deal with actualizing framework call intervention. One methodology is to actualize intervention as a client level library [8] such that mediation code is executed in the process' setting executing the system call. This is generally simple to execute, possibly yields more convenient code, and uses the unmistakable limit between client level and bit level. Sadly, it doesn't give viable confinement of utilizations and can be effectively subverted whenever. It rather requires their collaboration and does not work for statically-connected libraries or specifically executed framework calls.

Another methodology is to utilize a part process following office, for example, ptrace [9], which permits a client level procedure to screen another procedure [10]. By utilizing accessible part usefulness, this procedure following methodology can uphold an OS virtualization deliberation more successfully than entirely client level methodologies. In any case, ptrace has numerous restrictions regarding execution and security [11], and the semantics of ptrace are profoundly framework particular, which brings about a non-versatile strategy.

A third approach is to change the bit specifically to actualize mediation. This offers most extreme edibility, with the least mediation overhead. Be that as it may, has the written work code straightforwardly in the bit is more convoluted and awkward than in client level, harder to investigate, and the outcome is well on the way to be non-compact. Binds the usage to the piece internals obliges following, in detail, all resulting portion overhauls. Moreover, forcing a part fix, re-accumulation and reboot procedure is a genuine down to earth hindrance to organization and usability.

Given the constraints of different methodologies, the auhors have actualized OS virtualization as a loadable bit module that works with major Linux piece adaptations, including both Linux 2.4 and 2.6 bits. The usage keeps away from changes to the working framework part, and means to manufacture entirely on its traded interface however much as could reasonably be expected. It supports the case reflection additionally permits different procedures to keep running outside the virtualized situations to straightforwardness sending on frameworks which require such legacy usefulness.

### 3. Virtualization Challenges

Given this part module, mediation construction modeling, the authors now examine key execution challenges in supporting virtualized framework calls. Virtualization obliges that some state be kept up by the virtualization module. The fundamental express that should be kept up is the unit's asset names, the basic framework physical asset names, and the mapping in the middle of virtual and physical names. Throughout this discourse, they accentuate that execution is an essential concern and a large portion of the methodologies that are designed to accomplish low execution overhead. Table 2 gives a techniques' outline and information structures used to keep up virtualization state effectively.

A RST rough guess methodology utilizes two sorts of hash tables that can be immediately listed to perform the vital interpretation. One is a framework wide hash table listed by physical identifiers on the host OS, that profits the comparing unit and virtual identifier. The other is a for every unit hash table listed by virtual identifiers particular to

a case that profits the relating physical identifiers. A different pair of hash tables would be utilized for every OS asset that should be virtualized, including PIDs, SysV IPC, and pseudo terminals. For multiprocessor and multi-strung frameworks, legitimate hash table upkeep obliges locking instruments to guarantee state consistency. Taking care of these locks to dodge gridlock and to lower execution over-head is a non-unimportant matter.

The utilization of these hash tables alone can bring about imperfect execution. While hash tables give consistent time lookup operation, there is a non-immaterial execution over-head because of included lock dispute, additional calculation needed to do the lookup, and some subsequent reserve contamination.

Specifically, the framework wide hash table is utilized for every asset access to focus the unit connected with the running procedure. The incessant utilization of this hash table can bring about lock dispute and weaken adaptability.

To minimize the expense of deciphering between case name spaces and the basic working framework namespace, the authors connect with every local procedure information structure an immediate reference to the procedure's expanded virtualization state and the procedure's case. These immediate references go about as a reserve advancement that wipes out the need to utilize the table to get to the virtualization information of a procedure, decreasing the hash table lookup rate.

While this immediate affiliation just obliges two references, it is improbable that, the local portion process information structure has two unused references which can be utilized for this reason. Rather, a successful arrangement is to develop the zone possessed on the procedure's part stack by two pointers that reference the significant information structures. In this way, once a piece process information structure is gotten, there is no compelling reason to allude back to any hash tables to make an interpretation of from physical to virtual identifiers. Since this operation is so basic, this decreases the virtualization overhead of the framework over an expansive scope of virtualized framework calls and wipes out a noteworthy potential hotspot for lock conflict as shown in Figure 1.

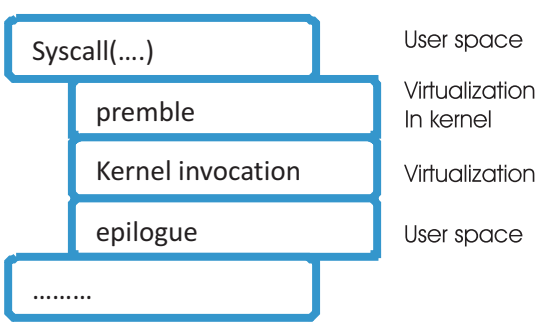


Figure 1. Virtualization Wrapper

## Conclusion

While OS virtualization ideas have been already talked about, the past work does not address vital execution issues in supporting OS virtualization in the connection of merchandise OSs. To the best of insight, the authors' work investigate these usage issues inside and out for the rst time. The authors discussed about talk about the requirement for framework call intervention for actualizing OS virtualization and analyze different methodologies for giving this usefulness. They exhibit the advantages of a loadable portion module usage and demonstrate that the overhead of this methodology is considerably not exactly different methodologies, for example, utilizing procedure following usefulness. Also they discussed about how OS virtualization state ought to be put away and depict a few critical enhancements for guaranteeing low execution overhead.

## Acknowledgment

The authors would like to thank JSS Mahavidyapeetha, Mysore & JSS Academy of Technical Education, Bangalore, India for their continuous support.

## References

**[1]. S. Bhattiprolu, E. W. Biederman, S. Hallyn, and D. Lezcano. (2008).** "Virtual Servers and Checkpoint/Restart in Mainstream Linux". *SIGOPS Operating Systems Review,* Vol. 42, No. 5.

**[2]. T. Garnkel, (2003).** "Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools". In *Proceedings of the Network and Distributed Systems Security Symposium*, San Diego, CA.

**[3].** PID Namespaces in the 2.6.24 Kernel. Retrieved from

http://lwn.net/ Articles/259217/.

**[4]. G.J. Popek and R.P. Goldberg. (1974).** "Formal Requirements for Virtualizable Third Generation Architectures". *Commun. ACM*, Vol. 17, No. 7, pp. 412-421.

**[5].** How to Break Out of a Chroot Jail. retrieved from http://www.bpfh.net/simes/computing/chroot-break.html.

**[6]. O. Laadan and J. Nieh, (2007).** "Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems". In *Proceedings of the 2007 USENIX Annual Technical Conference,* Santa Clara, CA, June 2007.

**[7]. B.K. Schmidt, (2000).** "Supporting Ubiquitous Computing with Stateless Consoles and Computation Caches". Ph.D thesis, CS Department, Stanford University.

**[8]. M. Jones, (1993).** "Interposition Agents: Transparently Interposing User Code at the System Interface". In *Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP),* Asheville, NC.

**[9]. M. McKusick, K. Bostic, M.J. Karels, and J.S., (1996).** *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley,

**[10]. D. Wagner. (1999).** "Janus: An Approach for Con nement of Untrusted Applications". Master's thesis, University of California, Berkeley, Aug.

**[11]. A. Whitaker, M. Shaw, and S. D. Gribble. (2002).** "Scale and Performance in the Denali Isolation Kernel". In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA.

## ABOUT THE AUTHORS

*Abhilash C B is presently working as an Assistant Professor in the Department of Computer Science and Engineering at JSS Academy of Technology, Bangalore, India. He got his M.Tech in Software Engineering from VTU, Belgaum, India and B.E. in Computer Science and Engineering from VTU, Belgaum, India. He has a Professional Membership with MISTE and MIAENG. His research interests are in Operating System Virtualization and Data Mining.*

*Dr. D.V. Ashoka is presently working as a Professor in the Department of Information Science and Engineering at JSS Academy of Technology, Bangalore, India. He got his PhD in Computer Science and Engineering from Dr. MGR University, Chennai, India, M.Tech in Computer Science and Engineering from VTU, Belgaum, India and B.E. in Computer Science and Engineering from Kuvempu University, India. He is an approved research guide for many Indian universities and has over 20 years of experience in Teaching, Research and Administration. He has professional membership with FELLOW IEI, IEEE, MISTE, MCSI and MIAENG. He is one of the National Award winners "Rashtriya Ekta Samman-2013". His research interests are in Operating System Virtualization, Knowledge Engineering, Communication, Network Security and Data Mining.*