Sultan Qaboos University

College of Science

Department of Computer Science

Genode Microkernel

Course Title: Fundamentals Of Operating System COMP4501

Submitted to: Prof. Ahmed Al-Kindi

Student's name and ID:

Mawada Khalfan Al-Husaini s130786

Alzahraa Hamed Al-Suleimani s127124

Marya Mohammed Al-Rahbi s131247

Kalsoom Kanwal Shoukat Ali s128453

6th.Dec.2021

Introduction:

Scientists and developers from all around the world are competing against each other to achieve optimality in certain areas. The area that will be explained in this report starts with the terminology known by operating system kernels. It is a greatly complex part of the software used to manage hardware's access, operation of storing information inside the files, and the system resources like memory, CPU, and hard drive. Therefore, such a kernel commands the privilege to control the entire machine.

Later, a better structure was invented in 1980 called microkernels. The basic idea behind the microkernels is to implement as much as possible running functions that are on the privilege mode as user space programs for which they can be turned on and off like normal programs. Moreover, this invention was certainly useful in future adjustments and developments.

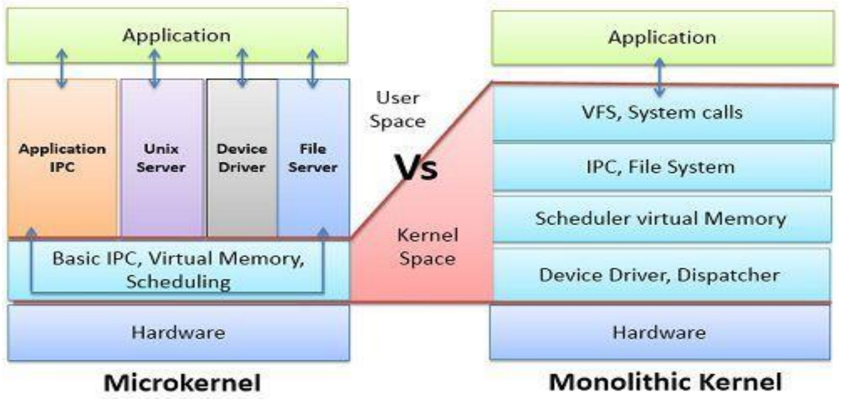
As there are many types of microkernels like redox, sel4, and others, our project focus on one type identified by Genode.

Genode operating system framework:

One of the main goals of every developed operating system is to make it simpler, regarding the fact of the complexity of an operating system due to high demands and dynamic workloads. The complexity can never be ignored but it can be controlled inefficient manner. Genode is a free open-source operating system architecture for building highly secure component-based operating systems provided by Genode labs. Genode Labs is an offspring that's founded by two former researchers of the OS research group of the Dresden University of Technology (Company). They built the Genode due to security problems that are common to current operating systems. It can handle workloads ranging from 4 MB embedded hardware to general-purpose dynamic computing. Furthermore, it can be deployed on various cores and on various clusters, where it can be used on top of the Linux kernel to deliver rapid development test cycles. In addition, the framework is accompanied by a custom-developed kernel which will reduces the complexity of the trusted computing base (TBS) compared to other kernels. The CPU architecture that Genode microkernel uses is x86 (32 and 64 bit), ARM (32 and 64 bit), RISC-V and has more than 100 ready-to-use components (About genode).

Genode microkernel:

Genode was considered as classical L4 microkernels in which the core process used to run as a user-level root task on top of the kernel. Then, because of some developments base-HW started executing Genode's core directly on the hardware with no distinct kernel underneath it, so in other words, the core and the kernel are melted into one hybrid component (Execution on bare hardware (base-HW) - genode OS framework foundations).



Difference between Microkernel and Monolithic Kernel

Genode operating system framework can have two types of kernels, the microkernel or monolithic kernel. The main difference between the two kernels is that the microkernel-based system separates the address space of the operating system services and the kernel, while the monolithic kernel-based system combines the address space for both operating system services and the kernel (Lithmee, 2018).

Genode's structure:

The Genode OS architecture is built around an organizational structure, it supports modularity for which it contains three main modules that are: Microkernel in the kernel space, Genode components in the user space and application layer. As the functionalities of the Genode operating system increases this affects positively in its scalability, and to keep this amount of functionality manageable it needs policies to strict some rules for particular situations. Genode organizes processes in form of a tree, as illustrated in figure1. These arrows represent how child processes are created

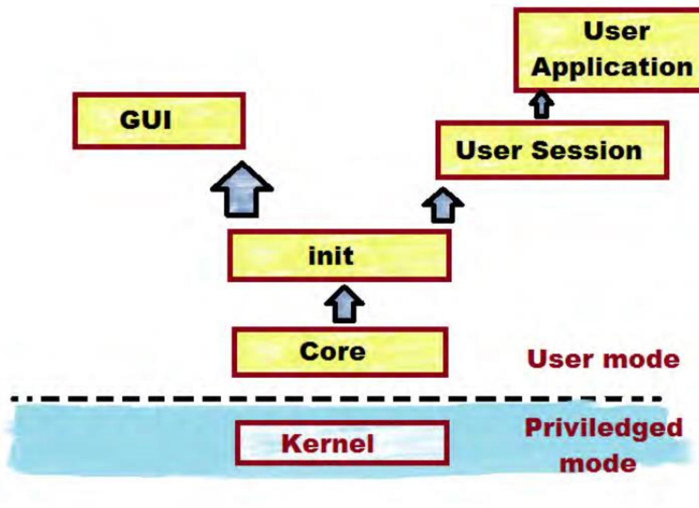


Figure 1: Structured Genode OS framework Architecture

using the resources of their parents. During the creation of a child process, the parent defines the entire virtual environment in which the new process will be executed. Likewise, each child is able to create children from its assigned resources, thereby creating an arbitrarily structured subsystem. Managing each of the subsystems that it created and defining inter-relationships for them, such as selectively granting communication between them or allocating physical resources, is the responsibility of each parent. Each hierarchy level has the same parent-child relationship, so this organizational approach is recursively applicable. Through the use of this property, we can separate policy from the mechanism by enabling a strict separation of duties (Genode Operating System Framework Foundations).

Genode major services:

The operating system has two parts, which are kernel space and userspace. In a monolithic kernel each system service runs in a kernel space, for example, process manager file system, input driver. In contrast, since kernels become unmanageable because of their large number of kernel code the microkernels appear. Its main general service is to minimize the functions and services from kernel space and move them to the user level to reduce the complexity. Similarly, Genode microkernel can reduce the system complexity for each security-sensitive application individually to minimize the system complexity, it uses a strict organizational structure for all software components and applications. In addition, Genode supports various microkernels such as fiasco, pistachio and it has a Para Virtualized Linux Layer (Tiny Core Linux) itself. Thirdly, Genode provides a secure operating environment, it protects names, processes identification numbers, etc. Since it is easier to attack with global known names and IDs, for instance, file names, process IDs, thread IDs, device nodes, or ports numbers, Genode can operate without any visible information and does not rely on globally visible names and IDs. Moreover, Genode framework can be used as an application over OS by using the Genode toolchain and Genode source. Genode toolchain is a software development tool that are used to combine with one another. It also allows building an application that directly communicates with OS and kernel, instead of using the virtual Linux layer (Genode Operating System Framework Foundations).

Genode component:

Genode has a hierarchical component architecture, with each component having its own protection domain, which includes its address space and capability space. Genode's purpose is to ensure that components cannot escape their protection domains, which means they can't access memory outside of their address spaces or conduct operations without the requisite capabilities. Genode's Core component is the most privileged component that's ultimately responsible for making calls. It is also in charge of providing operating system-level services to other components. The services include memory, CPU time, and interrupt handling. Core provides these services by talking directly to the underlying kernel (Constable, 1970).

Device drivers, resource multiplexers, protocol stacks, and applications are the four categories in which Genode components often fall. The following is a quick summary of each of them: A device driver is specialized in certain hardware platform; it converts hardware resources into session interfaces that are device independent. Secondly, resource multiplexers provide mechanisms to multiplex device resources to multiple clients. Thirdly, protocol stacks that can either be independent or libraries basically map low-level interfaces to higher-level interfaces (or sometimes vice versa). Moreover, applications are Genode components that utilize the services of other components but do not provide services themselves. README files are included with most programs and may be found in their corresponding directories. APIs given by protocol stacks are used by applications to provide functionality (Genode Components Overview).

A list of all its system programs:

System programs are designed to offer an environment for program development and execution. In our project we discovered three system programs implemented in the Genode microkernel.

File system: File system is a system software that is provided by the Genode microkernel. At the file and directory level, and storage service is given to the user who interacts with the operating system, and this service is provided by the file system session interface. This service operates by multiplexing the storage device among multiple users. This service contains many operations for file manipulations and modification, some of them are reading, writing, opening, creating, renaming, moving, deleting, and querying files, directories (Genode Operating System Framework Foundations).

Time session: Time session system program supplies the user with session local time source. A registered signal handler receives the signals that are generated by the user when scheduling timeouts. In addition, the user can ask for the elapsed number of milliseconds from the moment of creating a time session. For more clarification, suppose you are about to initialize a device driver that you have been working on and in normal circumstances, the initialization process will take a few milliseconds at some point because the device requires it (M-Stein) & (Genode Operating System Framework Foundations).

Loader: The loader session interface offers the chance for the user to create Genode subsystems, and these subsystems are represented as children of the loader service. Unfortunately, the loader does not have full control of the subsystems. Furthermore, the loader can only define the binaries and configuration to start, define the position of the subsystem on the screen and kill the subsystem (Genode Operating System Framework Foundations).

Inter-component communication:

Genode supports multiple kernels, each with its own IPC mechanism. In general, inter-component communication in Genode is done through three mechanisms: synchronous remote procedure calls (RPC), asynchronous notifications, and shared memory. Most of the ICC that is done is a combination of these three mechanisms. For instance, Asynchronous notifications with RPC are used for propagating state information, RPC also can be combined with shared memory to transmit a large amount of information in a synchronous way, and asynchronous notification with shared memory to decouple producers and consumers of high-throughput data streams (Genode Operating System Framework Foundations).

In RPC, control is passed from the caller to the callee until the function returns, just like in a function call. There are times when the information supplier does not want to rely on the recipient to relinquish control. An asynchronous notification method is used to deal with such circumstances. Furthermore, it is not feasible to transfer large amounts of information between components using synchronous RPC or asynchronous notifications. Shared memory can be used in this situation. Memory sharing allows large amounts of information to be propagated between components without active participation by the kernel (Genode Operating System Framework

Foundations). In process creation, every parent process can give its resources and inter-process communication access to its childern (Genode 2021).

In the kernel inter-process-communication (IPC) mechanism at the lowest level, the kernel's IPC mechanism is used to transfer messages back and forth between client and server (Genode Operating System Framework Foundations).

Thread synchronous:

Threads interact with each other via inter-component communication that we discussed earlier. They call each other and accept RPC requests by using system calls that are provided by the kernel interface. (*send_request_msg*, *await_request_msg*, *send_reply_msg*). In the kernel, IPC is performed using user-level thread-control blocks (UTCB). Each thread has a memory page that is mapped in the kernel which is used to carry the IPC payload. The following is a simplified technique for transmitting a message

1. The sender marshals its payload into its UTCB and invokes the kernel,
2. The kernel transfers the payload from the sender's UTCB to the receiver's UTCB and schedules the receiver,
3. The receiver retrieves the incoming message from its UTCB.

Kernel Exception handling is not needed since all the UTCBs are mapped in the kernel and the flow of the execution is already predicted. Another way in which the threads communicate with each other is by using synchronization primitives that are provided by the Genode API. To put these parts of the API into action, system calls are used to control the execution of threads such as stop_thread, restart_thread, yield_thread. The kernel also supports asynchronous notifications by providing system calls for the submission and reception of signals as await_signal, cancel_next_await_signal, submit_signal, pending_signal, and ack_signal. Moreover, it provides lifetime management of signal contexts too such as kill_signal_context. Despite the fact that all core threads are in privileged processor mode, they use a kernel library to synchronize hardware interaction (Execution on bare hardware (base-HW) - genode OS framework foundations).

Scheduling:

Static priority scheduling is used in traditional L4 microkernels, where the scheduler selects the thread with the highest priority to be executed by the CPU. If more than one thread has the same priority, then they will be scheduled by a round-robin scheduler. Despite being easy and fast to implement, it may result in starvation of the lower-prioritized threads. Due to that base-hw uses a way that solves that problem without decreasing the performance. The base-hw scheduler introduces the distinction between high-throughput-oriented scheduling contexts - called *fills* - and low-latency-oriented scheduling contexts - called *claims*. Base-HW uses a concept known as the super period, which is a multiple of typical scheduling time slices, e.g., one second. The whole super period represents 100% of the CPU's CPU time. Parts of it can be assigned to scheduling contexts and a CPU quota corresponds to a portion of the super period. At the start of the super period, each claim has its whole allocated CPU quota. Within the super period, the priority determines the absolute scheduling order among current claims with quota remaining. The scheduler will remain in claim mode as long as such claims exist, and the scheduled claim's quota will decrease. At the end of a super period, the quota of all claims is replenished to the initial value. When the scheduler is unable to find an active claim with CPU-quota left, it switches to the fill mode. A round-robin scheduling with identical time slices is used to schedule the fills. The super period's progress has no bearing on the mode's scheduling order or time-slices. The claim mode's implementation of quota and priority works neatly with Genode's hierarchical resource management approach: through CPU sessions, each process gains the ability to assign sections of its CPU time and subranges of its priority band to its children, even if it is unaware of the CPU time or priority (Execution on bare hardware (base-HW) - genode OS framework foundations).

deadlock mechanisms:

Deadlock is a set of blocked processes each is holding a resource and waiting to acquire a recourse held by another process in the set. The mechanisms used in genode operating system for deadlock are specified as follows. In a multi-threaded application, if the principal thread remains responsive, a secondary thread's crash may go unnoticed. However, in many cases when a component crashes either due to page fault or stack overflow, it gets stuck in a busy loop, and here where it produces a deadlock or throws an unhandled exception(abort) (Multi-threading and synchronization - genode OS framework foundations).

Memory management:

The kernels of today's operating systems contain a large quantity of highly-complex software that handles everything like managing resources (e.g., memory), accessing hardware, storing data on file systems, handling network packets, and controlling user processes.

Ram_allocator interface is the interface used to allocate physical memory through Genode in the form of RAM dataspaces. By implementing this interface, the PD session provides a mechanism to allow a component access its allocated RAM. The RAM dataspaces that is allocated with the Ram_allocator interface, may serve as a backing store for fine-grained component-local allocators such as the Heap module. Genode's RAM accounting is implemented using the Ram_allocator interface. In cases where a server needs to allocate RAM on behalf of a client, the interface serves as a hook for tracking and configuring the specific usage of that client's RAM (Genode Operating System Framework).

The installation/setup steps: (Download)

1. Download the virtual box from https://www.virtualbox.org/
2. Download the disk image form here sculpt-21-10.img and link it to the genode by the following steps:
   - Click new
   - Select the name and version of the operating system
   - Go to settings>storage>empty
   - Choose the downloaded disk image
3. Download the appliance that contains the Sculpt 21.10 image along with a known-to-work VirtualBox configuration from this link sculpt-21-10.ova
4. Start VirtualBox and import the OVA file as appliance (File>import applaiance)
5. Click on start to start the genode microkernel

Reference:

About genode. Genode. (n.d.). Retrieved December 6, 2021, from https://genode.org/about/index.

Company. Genode Labs - Company. (n.d.). Retrieved December 6, 2021, from https://www.genode-labs.com/company/index?lang=en.

Constable, S. D. (1970, January 1). *[pdf] extending sel 4 integrity to the Genode OS framework: Semantic scholar*. [PDF] Extending seL 4 Integrity to the Genode OS Framework | Semantic Scholar. Retrieved December 6, 2021, from https://www.semanticscholar.org/paper/Extending-seL-4-Integrity-to-the-Genode-OS-Constable/09e5e03e8585172d7945cba6e891ece929fce286.

Execution on bare hardware (base-HW) - genode OS framework foundations. (n.d.). Retrieved December 6, 2021, from https://genode.org/documentation/genode-foundations/19.05/under_the_hood/Execution_on_bare_hardware_(base-hw).html.

Download. Genode. (n.d.). Retrieved December 6, 2021, from https://genode.org/download/index.

Genode Operating System Framework. Genode. (n.d.). Retrieved December 6, 2021, from https://genode.org/.

Genode Components Overview. Genode. (n.d.). Retrieved December 6, 2021, from https://genode.org/documentation/components.

Genode Operating System Framework Foundations. (n.d.). Retrieved December 6, 2021, from https://genode.org/documentation/genode-foundations-21-05.pdf.

M-Stein. (n.d.). *GENODIAN/2021-05-31-A-SHORT-GUIDE-TO-THE-TIMER-SESSION-INTERFACE.TXT at master · m-Stein/genodian*. GitHub. Retrieved December 6, 2021, from https://github.com/m-stein/genodian/blob/master/2021-05-31-a-short-guide-to-the-timer-session-interface.txt.

Multi-threading and synchronization - genode OS framework foundations. (n.d.). Retrieved December 6, 2021, from https://genode.org/documentation/genode-foundations/21.05/functional_specification/Multi-threading_and_synchronization.html.

Lithmee. (2018, November 22). *What is the difference between microkernel and Monolithic Kernel*. Pediaa.Com. Retrieved December 6, 2021, from https://pediaa.com/what-is-the-difference-between-microkernel-and-monolithic-kernel/.

Wikimedia Foundation. (2021, November 20). *Genode*. Wikipedia. Retrieved December 6, 2021, from https://en.wikipedia.org/wiki/Genode.

Additional resources:

Difference between linux and Genode. GeeksforGeeks. (2020, July 7). Retrieved December 6, 2021, from https://origin.geeksforgeeks.org/difference-between-linux-and-genode/.

Techopedia. (2014, June 5). *What is a microkernel? - definition from Techopedia*. Techopedia.com. Retrieved December 6, 2021, from https://www.techopedia.com/definition/3388/microkernel#:~:text=Microkernels%20were%20first%20developed%20in,in%20the%20design%20and%20programming.