PTC Perc vs Java

INTRODUCTION

This White Paper explains how the PTC Perc Java platform is different from other popular Java platforms such as Oracle Java™ and OpenJDK.

WHERE IS PERC USED?

EMBEDDED SYSTEMS

PTC Perc is built for embedded systems rather than general purpose computers. A general-purpose computer can run many different types of applications, often concurrently, from office apps to databases to client/server to graphics and AI. They employ powerful CPUs and GPUs with lots of memory and storage to support a wide variety of potential use cases.

An embedded system is a computer that performs a dedicated function within a larger mechanical or electrical system. The CPU, memory, and I/O resources of embedded systems are sized to meet the requirements of the system. They may have a Human Machine Interface (HMI) or they may not require any human interaction. They are often expected to operate for months or years without interruption.

Some embedded systems are part of a "system of systems," such as in factory automation, energy, or smart cities where smaller devices collect sensor data and drive actuators at the "edge" of a network. These connect to edge gateways to report data and receive commands. Gateways pass summarized data to on-premise servers or cloud services. An edge gateway can analyze data and apply machine learning and artificial intelligence to make time-critical decisions, such as shutting down a tool that reports excessive temperature or vibration. A factory may have thousands of edge devices and dozens of edge gateways providing real-time data and control in a distributed supervisory system. Perc provides a foundation for edge gateways to perform local decision-making while responding to critical events.

REAL-TIME SYSTEMS

This brings up a key difference in Perc. It does real-time Java. A real-time system has timing deadlines. It must respond to events such as user input, sensor data, or performing a periodic task within a never-to-exceed time limit. Missing a deadline means a failure of the system. One common misconception about real-time systems is that they must be fast. Real time isn't about raw speed. It's about consistently executing tasks within a predetermined time frame. The word for this characteristic is determinism.

WHAT CAN I DO WITH PERC THAT CAN'T BE DONE WITH STANDARD JAVA?

First, let's be clear that Perc can run Java language programs without any special coding, so in that sense it is a standard Java platform. Java is a high-level, object-oriented programming language that is portable across many processors and operating systems. It has strong typing and memory safety to minimize programming errors, avoid security vulnerabilities, and enhance developer productivity.

With that said, there are some unique capabilities in Perc that are not found in other Java platforms. First, a tool called the ROMizer.

THE ROMIZER

The name of this tool was chosen early in the history of Perc when embedded systems stored code and data in Read-Only Memory (ROM) chips. Today it resides on disk or in flash memory, but the purpose of the ROMizer remains the same – to package a Java application and the runtime Perc Virtual Machine (PVM) into a single executable file to simplify deployment and improve startup time as well as other benefits to be mentioned later.

In contrast, a standard Java platform requires a separate Java Runtime Environment (JRE) to be installed on the target, followed by application code and supporting libraries, usually in the form of JAR (Java ARchive) files. The JRE is told where to find the application JARs and the name of a class to initially load and execute. As the application runs, the JRE can "just-in-time" compile Java "bytecode" to native instructions for faster execution.

While Perc is perfectly capable of loading, compiling, and running from JARs in the same way, the ROMizer gives the option of collecting the subset of Java classes, resources, and data files required by your application and linking them with the core Perc Virtual Machine into a standalone binary executable. The bytecode portion of Java classes can be compiled "ahead of time" to native instructions just like a C or C++ compiler would do.

Figure 1 shows the inputs and outputs of the ROMizer. From the left are the application's Java classes, resources and data files. From the right are standard Java system classes and 3rd party library classes the application depends on. The ROMizer packages internal representations of the classes, resources, and files plus the ahead-of-time compiled native instructions for the method bytecode in each class into a Linux ELF "Image" file.

Then the ELF Image file is linked with the core PVM libraries, and any static libraries required by the application to generate a custom PVM executable.

Why bother doing it this way? One reason is to make Java programs more deterministic. There are no unpredictable delays for loading Java classes or when bytecode is just-in-time compiled. Everything is present and ready to execute at startup. A second

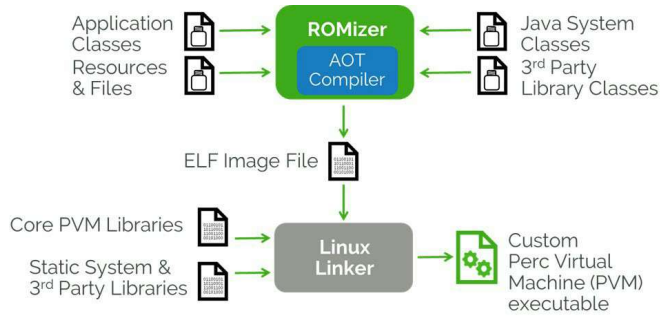

Fig. 1

reason is to avoid potential version mismatches between JAR files installed on a target. The third reason is for security. Java bytecode stored in JAR files can be reverse engineered back to Java source code. Deploying JARs puts your intellectual property at risk of malicious actors copying or tampering with it. The ROMizer removes the need to deploy any JAR files.

THE DISPATCHER

Unlike other Java platforms, PTC Perc provides deterministic dispatching of Java application threads. A traditional Java Virtual Machine does little to influence when the underlying Operating System (OS) runs a thread. New threads are created and handed to the OS for scheduling at its discretion. Perc, on the other hand, can use Linux SCHED_RR or SCHED_FIFO real-time scheduling policies with Linux priority levels ranging from 1 to 99 mapped to Java thread priorities. Linux gives precedence to threads assigned to a real-time policy over all other processes in the system and strictly enforces the priority of threads within real-time policies.

In addition, the Perc dispatcher controls the processor core assigned to each Java thread. Only one thread at a time is allowed to run on a given core. At any moment, the N highest priority runnable threads will

be assigned to execute on N available cores. In this way, Perc overrides any discretionary behavior of the Linux scheduler.

Another feature of the dispatcher is priority inheritance. Priority inheritance solves the problem of a "priority inversion" scenario in which a high priority thread is blocked from running while it waits for a low priority thread to release a resource lock it requires. In Perc, the low priority thread is temporarily boosted to the high priority until the lock is released. This happens automatically, without any special coding by the software developer to assure real-time determinism.

THE GARBAGE COLLECTOR

A major feature of Java Virtual Machines is automatic memory management. One of the biggest differences between Perc and other Java platforms is how it reclaims memory for objects allocated by programs but no longer referenced. Garbage Collection (GC) relieves developers from having to keep track of allocated objects and deciding when to de-allocate them. Traditional Java garbage collectors scan "live" objects referenced by threads, marking them as reachable from program code and then "sweeps" memory for any un-marked objects to be reclaimed. This leaves free blocks in memory to be used for new objects, but fragmentation over time can make it impossible to find a block large enough to satisfy an allocation. The GC must coalesce free memory by compacting live objects in memory. Traditional collectors run at random times and prevent Java threads from running when live objects are being moved and pointers to them are updated. This results in unpredictable "stop the world" pauses and thus non-deterministic behavior.

Perc has a patented real-time garbage collector that does not suffer from "stop the world" pauses. It can relocate objects concurrently with running Java programs. The garbage collector is implemented with parallel worker threads running at a user-designated priority and baseline CPU time allocation. Furthermore, GC worker threads can be preempted by a higher priority Java thread at any time.

PVM PROTECT

One more unique feature of Perc that's been added recently– the PVM Protect tool. As mentioned earlier, deploying JAR files puts your application software at risk of reverse engineering. While the ROMizer and ahead-of-time compiler reduces this risk, a ROMized binary still contains names of Java classes, methods, and fields so they can be referenced by dynamically loaded code or Java reflection APIs. The Linux 'strings' command will show these names to anyone who has access to the executable. Furthermore, the contents of Java resource files and data files are visible within a PVM binary. Developers of embedded systems for defense, infrastructure, or other security-sensitive applications may need to prevent viewing of these items.

The PVM Protect tool fully encrypts and wraps a PVM binary into a self-decrypting launcher using strong cryptographic algorithms and Trusted Platform Module (TPM) 2.0 hardware. TPM 2.0 chips are used in modern servers and laptops to provide secure boot and disk encryption. The Perc customer generates an RSA 2048 key pair, using the public key to encrypt the binary and provisioning the TPM 2.0 chip in each target system with the private key to decrypt it. At execution time, the launcher uses the TPM 2.0 chip to decrypt itself directly into RAM and launch the

application in Linux without storing plaintext on a hard disk or other persistent storage. The private key is "sealed" in the chip with a policy that allows it to be accessed only when the target system has validated bootstrap, kernel, and related software components. Any tampering of the boot process renders the RSA private key inaccessible.

PVM Protect delivers end-to-end encryption. One can safely transmit and deploy the encrypted PVM launcher without concern about threat actors accessing it either in transit or as it resides on media.

CONCLUSION

To summarize, PTC Perc is uniquely designed for embedded systems running Java language applications with real-time determinism and enhanced security. Perc has been deployed in thousands of real-world systems in industrial control, energy, transportation, telecom, aerospace, and defense for over two decades. Learn more about PTC Perc at:

https://www.ptc.com/en/products/developer-tools/perc

547055_Perc Real-Time Java_10_24