SYSGO
EMBEDDING INNOVATIONS

SYSGO Whitepaper

PikeOS

Safe Real-Time Scheduling

Adaptive Time-Partitioning Scheduler for EN 50128 certified Multi-Core Platforms

Introduction

Scheduling a complex application scenario is an own area of science and handle by various scheduling schemes, which are optimized for specific use cases. The main criteria for choosing a specific scheduling scheme are [STAL98]:

- Turnaround Time: Time between the submission of a process and its completion
- Response Time / Determinism: Time from the submission of a request until the response to be received
- Deadline: The processes start or completion deadline
- Predictability: A job should run in the same amount of time
- Throughput: The scheduler should maximize the number of processes completed per unit of time
- Processor utilization: Percentage of time, the processor is busy
- Fairness: Processes should be treated the same so that no process should suffer starvation
- Enforcing priorities: The scheduler should favor higher priorities
- Balancing resources: The scheduler should keep the resources of the system busy

The complexity rises, if the CPU has multiple cores, so that applications can run concurrently on all cores in parallel. Appropriate scheduling mechanisms are able to handle this with various concepts so that ideally a scheduler should be adaptable by considering the system configuration and the application design.

If an application is safety critical, than the predictability of the system becomes evident. The related safety standards mandate a timing analysis to prove, that the system is able to react in a guaranteed time to any event. As a result of this analysis, a 'worst case execution time' (WCET) has to be calculable for the system. The complexity for determining such a value rises with the complexity of the application and the complexity of the CPU architecture. The following list names some aspects, which increase the complexity of the application:

- Application threads need synchronization
- Application threads use and share resources
- Application threads may have a deadline to start and/or finish

With multi-core CPUs the resource sharing aspects complicates the determination of the WCET because:

- Usually second level caches are shared amongst the cores
- Memory, buses and I/O are shared amongst the cores
- Depending on the load balancing algorithm, the operating system or the application may run on different cores or switch from one core to another during runtime

A safety critical application has to cope with the above-mentioned aspects and be predictable in any circumstance.

Tried and trusted safety operating systems rely on "Time Partitioning" based scheduling, which ensures that complex application scenarios running on multi-core CPU are predictable and safe. This paper describes the patented scheduling scheme of the SYSGOs PikeOS hypervisor. We will first describe the PikeOS kernel concept and than explain how the PikeOS time partitioning ensures a predictable execution of a safety application.

PikeOS – Safety & Security by Separation

One of the most critical application examples is an avionic control system. The ARINC 653 standard (for avionic systems) specifies a software framework for safety critical avionics applications using a space and time partitioning concept. It allows the hosting of multiple applications of different safety levels on the same hardware and thus is comparable to hypervisor architecture. That is hypervisor technology is a proven technology for safety and security critical applications requiring safety and security certification. Most popular hypervisor implementations like VirtualBox, VmWare etc. provide an execution environment to run a commodity operating system like Windows or Linux on top of another Windows or Linux OS (type-2 hypervisor). These hypervisors depend on the functionality and assurance of that host-OS. These implementations have been developed for performance-oriented virtualization. Thus, safety and security aspects as well as assurance needed for certification have been not considered.

Safety related hypervisor technology rely on separation Kernel architecture. A separation Kernel uses a real-time micro-kernel as the basic operating system architecture and provides means for separation on top of the microkernel. The main advantage of this architecture is, that the microkernel is a real-time operating system and a hypervisor (type-1) in one product.

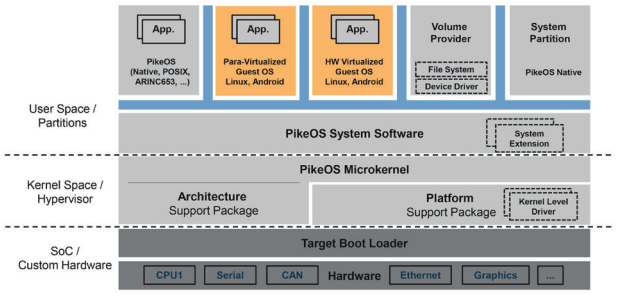

Figure 1: Separation Kernel architecture

The safety and security concept of a separation kernel is based on separating resources, so that applications cannot interfere with each other. The available resources

on computing hardware are the physical hardware components and the CPUs execution time. The separation of physical resources is called spatial separation or resource partitioning, while the separation of the available execution time is known as temporal separation or time partitioning. Time partitioning is not a classical scheduling mechanism. It is rather a means for dispatching time to partitions/virtual machines and it relies on resource partitioning, so that we need to understand the resource partitioning concept first.

PikeOS – Resource Partitioning

As discussed in the introduction, the sharing of resources can become a challenge for time critical scheduling. If resources are shared, delays or even deadlocks can happen, if a resource is blocked by another application. Resource partitioning primarily targets the protection of resources, which shall exclusively be used by a dedicated application running in a partition. Once a resource is assigned to a partition, other partitions will not be aware of this resource.

PikeOS Resource partitioning is achieved by statically assignment of a computing resource such as memory, I/O and file devices, secure communication channels and cores to partition / virtual machines. PikeOS ensures that during runtime an application has guaranteed access to the assigned resources and that the partitioned resources are not accessible from applications belonging to other partitions.

Resource partitioning is enforced by using the MMU to control access to the available resources. Each hardware device is somehow represented by a physical address in order to access this device. Resource partitioning is realized by using the MMU to map a certain memory area into a partitions virtual memory space and allow or deny read/write access to this memory space. The configuration of the MMU is done statically at system startup in the PikeOS microkernel and is not modifiable at run-time, ensuring that a hacker cannot modify the resource configuration later on. In summary, the separation makes sure that errors occurring in one partition cannot propagate to another partition.

Time-Partitioning based Scheduling

PikeOS partitions may host real-time as well as non-real-time guest operating systems. These two types of run-time systems have fundamentally different requirements with respect to scheduling:

- Real-time systems need to be able to make guarantees about the temporal behavior of their processes. A necessary precondition for this is that the operating system itself knows the points in time when it will be able to use the processor and for how long it will be able to use it. In other words: the schedule, which defines the switching between VMs hosting real-time operating systems, has to be strictly a function of time.

- Non-real-time operating systems, on the other hand, work by a “best effort” principle, i.e. they try to do as much as possible, attempting to use all the computational resources they can get. Hence, their goal is to minimize processor idle time. For a VM scheduler, this means that, whenever a VM is found to be idle, it should revoke the CPU from that idle VM and pass it to the next one, hoping that one will have useful work to do. The resulting VM schedule is obviously influenced to a large extent by the activities going on inside the VMs, so it is clearly not just a function of time

The ARINC 653 standard addresses the allocation of CPU time across different partitions. The approach described in this standard works on a fixed cycle time, within which the individual partitions are scheduled on the processor in a specified order for a guaranteed duration. This “time partitioning” method is suitable for real-time systems, but, as discussed above, it leads to poor system utilization.

Scheduling methods used by virtualization systems such as Xen and VMware, attempt to optimize processor utilization by revoking the CPU from VMs as soon as they become idle. But doing so, they cannot guarantee deterministic timing to their guest operating systems.

PikeOS Adaptive Time-Partitioning

The PikeOS partition scheduler uses a combination of priority-and time-driven scheduling to join together these contradictive approaches. The time-driven scheduler is a mechanism to distribute the available CPU time amongst the PikeOS partitions. It can be defined as a first level scheduler, which quantifies the available time into time partitions (tpi; i = 1..n). Depending on the application timing requirements, several time partitions can be defined and grouped into a time-frame (see Figure 2).

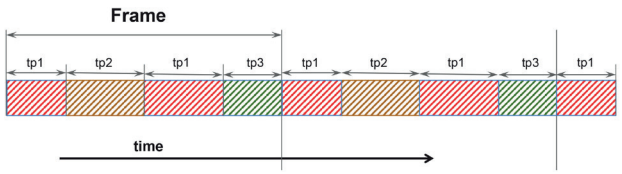

Figure 2: Time Partitioning frames

A time-partition may exist more than one time in a time-frame and can have different durations. As soon as the time-driven scheduler has completed a frame, the frame starts over gain and again (see Figure 2).

In contrast to the ARINC 653 standard, PikeOS scheduling uses a one-to-n assignment of resource partitions to time-partitions. That is, one or more resource partitions can be assigned to one time-partition (see Figure 3).

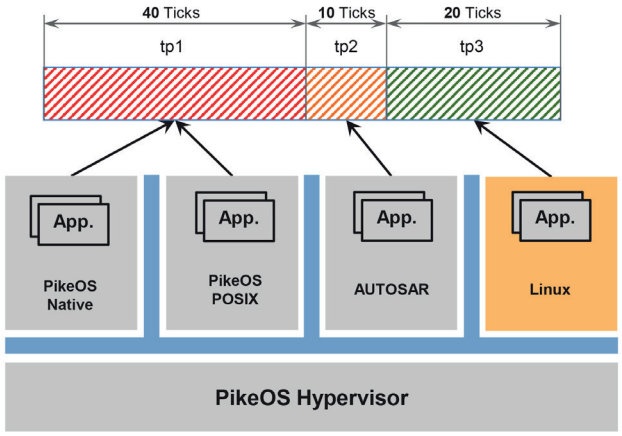

Figure 3: Time and Resource Partitioning assignment

In the PikeOS nomenclature user applications are called processes. Processes can be started in context of a task, where each task is characterized by it's own address space and a set of schedulable entities (threads) bound to it. A PikeOS partition may host more than one task (other tasks and child tasks), so that within a partition address space additional task address spaces are defined.

By assigning one or multiple resource-partition(s) to a time-partition, threads are grouped into time partitions, which are activated by the time-driven scheduler. However, in contrast to the ARINC 653 standard, one more time partition exists, that is active at all times. This time partition is referred to as the background partition, tp0, whereas the currently active one of the time-switched partitions is called the foreground partition, tpi (i = 1 ... N).

In addition to their time partition, threads do also have a priority attribute. Whenever both the foreground and background partitions have active threads, the thread to be executed is selected according to its priority. Figure 4 shows the principle of operation: Each time partition is represented as a priority-sorted list of FIFO ready queues. Each of them delivers its highest priority ready thread and of these, the highest priority one from either tp0 or the currently selected tpi gets chosen for dispatch.

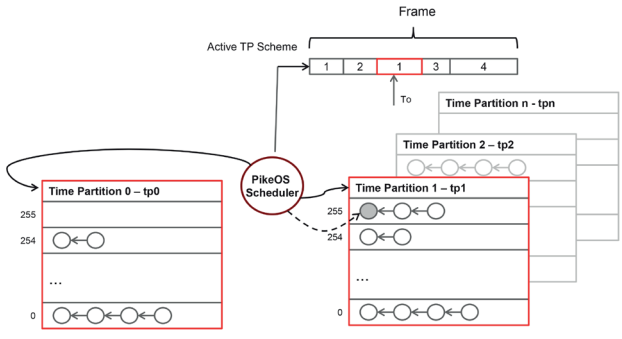

Figure 4: PikeOS Time-Partition Scheduler Principle

With this approach, priority ranges can be defined within which the CPU is assigned between the partitions using the fixed time-driven schedule defined in a time-frame.

The partitions periodically receive guaranteed time slices. But, whenever one of these partitions completes its job prior to having consumed its entire time slice, the unused time automatically falls back to the next lower priority thread in the background time partition. Thus, by running explicitly non-real-time applications with lower priority threads in the background time partition (tp0), these non-real-time applications receive all the computational resources that were assigned to, but not consumed by real-time application in the foreground partition (tpi).

The upper time-diagram in Figure 5 has 4 time-partitions tp1-tp4. As time-partition tp4 has low priority, it is put into tp0 and tp4s time slot is distributed amongst the other time-partitions, so that the higher priority partitions have more computing time. If one of the higher priority partitions (tp1-tp3) get idle before using the assigned CPU-time, tp0 receives the rest of the execution time (lower time-diagram in Figure 5).

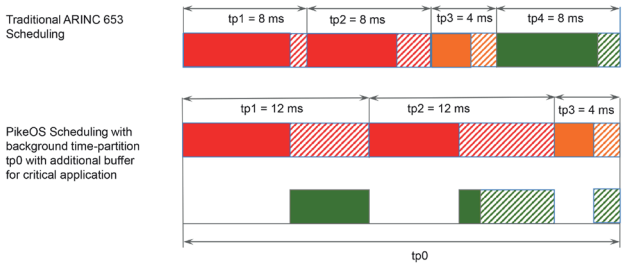

Figure 5: PikeOS Scheduling with low priority background application

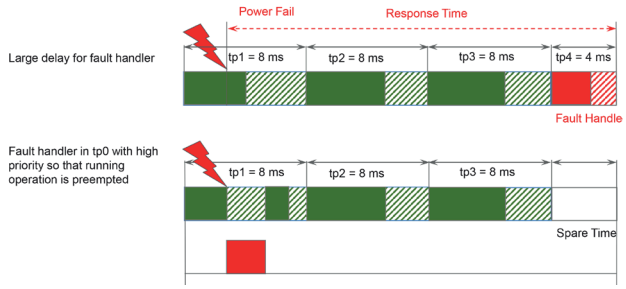

Figure 6: PikeOS scheduling with high priority background application

In the opposite scenario, it is possible to define threads in the background domain, which can override time partitioning by assigning them a priority above those of the foreground domain threads.

The upper time-diagram in Figure 6 has a fault handler, which is called in a dedicated time-partition tp4. In case of a failure the response-time of the fault-handler can be quite high, if the error happens right after the fault-handler time-partition has ended. By assigning the fault-handler to tp0, the handler is starter immediately, as it

has the highest priority among all active threads (lower time-diagram of Figure 6).

Clearly, such high priority threads must be considered as trusted code from the point of view of the threads that they can preempt. Therefore, it is important that the priority assignment functions implemented in the micro-kernel guarantee that no thread can incorrectly obtain rights to which it is not entitled. PikeOS achieves this by assigning a maximum priority to each thread. This value is set at thread creation time and is based on the maximum priority of the creating thread. It cannot be changed at any point during the thread's life cycle.

PikeOS 4.x Meeting Deadlines

The PikeOS scheduler combines deterministic behavior with optimized time resource usage with a static scheduling scheme. The optimal run-time behavior of the system is achieved by a deep analysis of the involved application components (threads) and the following static configuration of the system.

Meeting start or completion deadlines of threads must be part of the analysis and the time partitioning configuration. In non-safety real-time systems a dynamic planning based scheduler takes care of meeting the deadlines within the application. But as we have learned in the chapters before, a dynamic scheduler may not be usable for highly critical safety systems. Thus PikeOS provides a deadline monitor, which invokes the health monitor (HM) in case a deadline is missed.

The PikeOS health monitoring (HM) system is designed to handle errors at system runtime and to execute recovery actions as configured in the HM configuration tables and user-level error handlers.

In order to catch a deadline mismatch a thread has to be configured with specific flags indicating a release time (starting deadline) or a deadline (for completion). If the deadline parameter is set to future time value the thread will be notified when the registered deadline expires. The deadline-miss notification causes a health monitoring exception. For the handling of this special exception in the user-space, the health-monitoring tables must be configured with an users-pace exception handler for the faulting thread.

Certifiable Multi-Core Scheduling

The following example explains the usage of the PikeOS time and resource partitioning to control a multi-core processor, so that it fulfills the strict requirements of the EN 50128 (Software for railway control and protection systems).

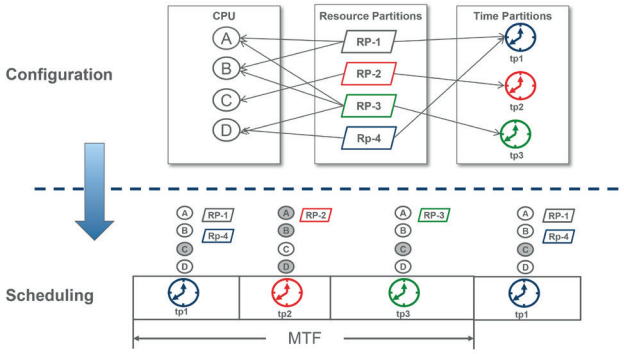

Figure 7: Configuration of a certifiable multi-core system

The assumed platform is based on a quad-core CPU. The major time frame is divided into three partition windows (tp1 to tp3). One critical single core application (RP2) shall have exclusive access to one of the cores (core_c) and have exclusive access to the entire platform during its time window (tp2). One performance-demanding partition (RP3) shall have exclusive access to the remaining three cores during its time window (tp3). One time slot (tp1) is shared between two resource partitions (RP1 and RP4). RP1 running on two cores and RP4 running on one core.

The selected configuration focus on a maximum level of isolation for the critical application accepting a waste of CPU time. Resource Partition 2 is the only partition executing on core 'C' and during the time slice of time partition tp2 there is no other partition execution. This eliminates any interference on hardware and software level. The level of determinism in this configuration is even better than on a RTOS based platform since the critical application does not share the core with other partitions which also keeps the state of the private caches unchanged.

Nevertheless the setting of caches and TLBs need to be considered. The PikeOS hypervisor provides means to invalidate instruction caches and TBLs and to flush the data cache between time partition switches. This ensures that caches and TLBs are in a defined state when a partition starts execution. The cache / TLB flush and invalidate operation takes place during the time partition switch, so it will steal the CPU cycles from the partition to be activated and thus cause a jitter. A possible approach is to define a small time partition window, which is allocated to an unused time partition ID and to insert this before the time critical application. This eliminates the jitter of the time critical application.

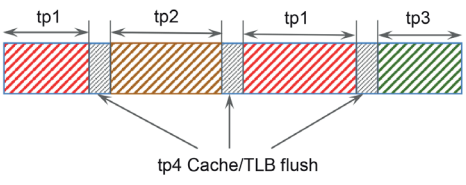

Figure 8: Avoiding jitter through cache and TLB Flushing

SYSGO Patent

Using a background time-partitions tp0 in parallel to the active time-partition is an invention by SYSGO and patented under the active patent numbers:

- EP 1831786 A1: Valid in CH, FR, DE, GB
- EP 1831786 B1: Valid in CH, FR, DE, GB
- US 20090210879 A1: Valid in the United States of America

The SYSGO approach adds the following flexibility to the ARINC time-partitioning based scheduling:

- The ARINC 653 standard requires a 1-to1 assignment between resource partitions and time-partitions. That is, each time partition is logically connected with a resource partition. PikeOS allows the assignment of multiple resource partitions to one time-partition, so that threads, which need to be scheduled together can be isolated if required. The ARINC 653 scheduler is just a subset of the PikeOS scheduler.
- The PikeOS time-partition scheduler can make use of idle time occurring in the active time-partition. Assuming that the threads in the active time-partition tpi have a higher priority than the priority of the threads in the background time-partition tp0, the higher priority will consume all time of the time-partition unless there is no thread left for execution. In this case, the scheduler will allow the highest priority thread in tp0 to execute.
- The PikeOS time-partition scheduler can execute an event-handler timely by assigning it a high priority in tp0. As tp0 threads are overlaid to the current active time-partition at any time, the event handler is called with a maximum delay of one system tick.

The ARINC 653 scheduler has been designed with the main focus on predictability in mind. That is, the response time must be calculable and the system must behave in the same way under all circumstances. Despite bringing flexibility to the strict ARINC scheduler, the PikeOS scheduler adheres to guaranteed predictability of the system behavior at any time. Even more, the combination of PikeOS resource partitioning and time-partitioning is perfectly suited to control a multi-core system so that it can be certified to the highest safety level of various safety standards.

SYSGO has used the above-explained configuration for its PikeOS real-time hypervisor to achieve the first SIL4 certificate on a multi-core system (dualcore Intel i7). This principle is not limited to dual-core, as long as a fitting time and resource partitioning can configure the processor in a deterministic way.

SYSGO Headquarters
Phone +49 6136 9948 500
sales-de@sysgo.com

SYSGO France
Phone +33 1 30 09 12 70
sales-fr@sysgo.com

SYSGO Czech
Phone +420 222 138 111
sales-cz@sysgo.com