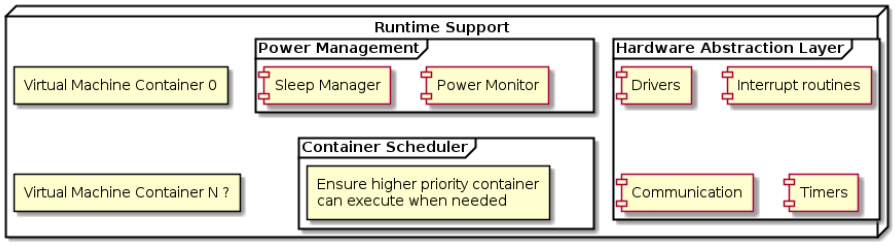
Sense-VM architecture

Single Threaded VM

Concurrent VM



Hardware Abstraction Layer

The hardware abstraction layer (HAL) will probably be implemented by building upon an existing HAL system such as ChibiOS, ContikiOs or ZephyrOS. Other HALs could also be candidates: CMSIS, HAL from ST for STM32 platforms for example. We should try to keep as much of the code as possible portable so that it can be ported to HALs that parhaps have more desirable features. It will be a collection of functions (and perhaps routines running on a timer interrupt periodically if needed) that can be called from the runtime support system.

Sleep Manager

If there is no processing going on (reported from the various schedulers) the sleep manager puts the system to sleep for a period of time such that it wakes up when it is time for the context/container with the earliest "wake-up" time to start executing.

The sleep manager may also be prompted to wake the system up by for example a sensor driver when the sensor has new samples to process.

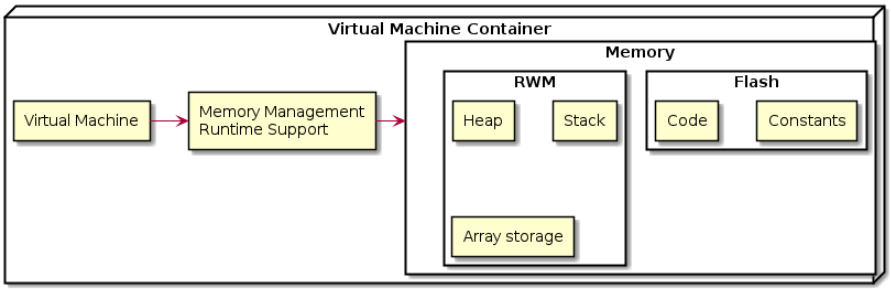
It is not unthinkable that there are applications that will just sleep indefinitely unless there is outside stimulus. It may still be desirable that these applications wake up periodically and just let the monitoring system know that it is still alive and ok (a heartbeat).

Container Scheduler and Virtual Machine Containers

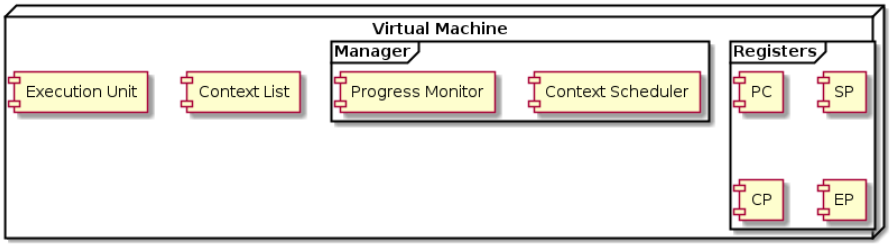
A virtual machine container should be an isolated executing environment for a virtual machine. (TODO: Should there really be more than one container?)

The container scheduler ensures that each container gets a time slot to execute. Possibly, these containers could be implemented using a "thread" feature of an underlying OS/RTOS/HAL or be given a certain number of iterations of some main loop.

Containers should be isolated from eachother to allow different level of criticality to different containers. High criticality containers should be prioritised over low criticality containers.



A VM container consist of a virtual machine instance and a set of dedicated and private memory resources.



Virtual Machine

The virtual machine is based upon the Categorical Abstract Machine (CAM) and consists of number of registers and and an execution unit. (More details later)

Execution Unit The execution unit reads instructions from code memory at location stored in PC register and evaluates each instruction over the state.

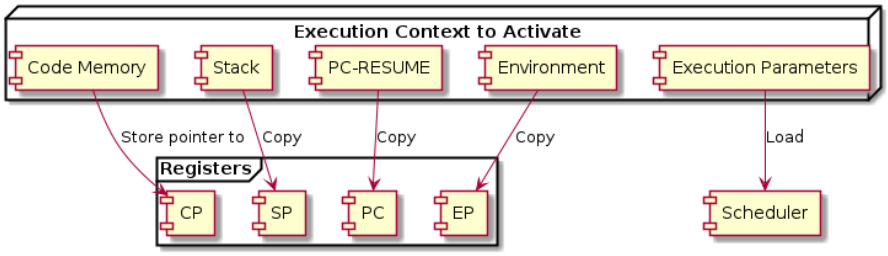
Context Scheduler and Context List

There is a list of contexts (separate instances of code and state) that can execute in a time-shared fashion on the execution unit.

(TODO: Maybe cooperative scheduling at this level? This would require "go to sleep" operations in the bytecode.)

The contexts that are executed on one VM are sharing the memory resources of the Virtual Machine Container.

Context Switching



The context list consists of some number of "Execution Context to Activate". A context is activated by copying the stored register state into the VM. Putting a context to sleep works in the reversed way.

Thoughts

Splitting concurrency up between internal to VM container via contexts and external between containers open up to running VM containers on different cores in parallel.

Management of other resources is also important. Communication interfaces, sensors etc. I think it would be beneficial to assign such resources to a VM Container and never allow the same interface to be connected to more than one VM Container.