SharpMedia Threading Design

© 2008 Žiga Osolin – SharpMedia project, document version: DRAFT

Table Of Contents

[About 2](#_Toc196651512)

[Goals 2](#_Toc196651513)

[Overview 2](#_Toc196651514)

[Driver **Error! Bookmark not defined.**](#_Toc196651515)

[InputDevice **Error! Bookmark not defined.**](#_Toc196651516)

[Action Mapping **Error! Bookmark not defined.**](#_Toc196651517)

# About

We realized that good software is also about efficient parallelization. With SharpMedia Threading, we provide unified access to system processors. Different libraries may extend this to provide other processors, such as graphics computation unit or even sound processor.

# Goals

Threading should provide:

* Unified control over working units, with per SharpMedia process based controller;
* Long-lasting job paradigm, with support to interrupt, resume, estimate;
* Information about working units, runtime optimizations, profile guided optimizations;
* Memory footprint, performance and capabilities of working units;
* All threads created through this control (forbid ThreadPool or creating Thread manually);
* Inter-process “master” thread controller;
* Per-process configuration at initialization, runtime reconfiguration of work unit controller;
* Multiple machine networks included in control.

# Overview

Threading is mostly implemented in **SharpMedia.Threading** namespace. CPU work units *drivers* are implemented in its subnamespaces. Other *drivers* are specific to other libraries. For example, **IGPUWorkUnit** is implemented in **SharpMedia.Graphics**.

The users of threading expose themselves by implementing **IJob**. Any long lasting *job* must be exposed as **IJob** so it can take advantages of threading system. This way, threading will carefully execute job with specific priority and prior knowledge gain by previous runs.

# IThreadControl

The interface exposes methods for high level interaction with the system. A job or job factory can be queued for execution. Furthermore, thread control allows you to register work units and optimizers.

Thread control is responsible for merging information gathered from jobs, factories, optimizers and work units and decide how to schedule work. It then performs appropriate allocations and executes jobs on appropriate work units.

Optimizer's information are merged using sureness parameters (weights).

# IJob

A job is a unit of work that must be executed in **one thread**. Each job exposes following properties:

* Supported running work units (CPU, GPU, PPU ...)
* Priority;
* Memory requirements (based on WU, range);

A job can expose additional properties through interfaces:

* Remoting (**IRemotebleJob**)
  + Bandwidth requirements
  + Database sync requirements
  + Confguration levels, back communication etc.
* Estimation (**IEstimatableJob**)
  + Time to complete
  + Preparation estimation
  + Dynamic (runtime) estimation
* Synhronization (**ISynhronizedJob**)
  + Jobs that make togehter higher unit (usable result) and must all complete
  + Balancing between other jobs
* Interuptable (**IInteruptableJob**)
  + A job can be interupted, paused, resumed

You can freely combine several interface (remotable, estimatable job for example).

# IHardwareResource

Each processing hardware resource (mainly CPU, PPU, RAM etc.) has exposed identifier, address of computer (for remote communication) and performance metrics exposed. Performance metrics are specific to each device, but **CommunicationPerformance** in common to all hardware resources. Each hardware resource is capable of measuring its performance.

A hardware resource measures its performance by performaning different operations. All other hardware resources must be **idle** when measurment is in process.

# IWorkUnit

**IWorkUnit** (WU) is a unit that can execute jobs. Each unit has associated primary memory unit. It can also have other memory units (auxilary). This abstraction does not yet provide access to do work using WU. It mainly provides performance measurments and other statistics.

A work unit can be **master** (needs no CPU control thread in order to use it) or **slave**. This two options result in different execution of **IJob**. If WU is master, the execution of IJob is expected to consume no resources in calling thread but only to assign work to WU. Otherwise, the execution of a job is assigned a worker thread that will monitor the actual execution on WU. This will be allocated on CPU (or other master, control capable) unit. This actually means that using slave WU will always result in 2 WU allocations (master and slave) with memory mainly on slave unit.

## ICPUWorkUnit

A CPU work unit provides additional performance measurments available for normal CPU processors. It also gives you access to assign work to specific thread of unit.Job is queued on the thread and executed when it comes on order. Note that queue should not be long because thread control will not schedule work to execute until it knows WU will be soon freed from work.

## IGPUWorkUnit

A GPU work unit is a slave unit. As such, execution will assign the WU and a dedicated thread to communicate with GPU. A job should aquire a lock on GPU WU (and by doing so, obtain **GraphicsDevice**) schedule work on graphics device. Then it can release the lock on GPU WU (e.g. graphics device) and wait for the work to complete using async checking.

# IMemoryUnit

A memory unit has performance measurments of read/write operations, different levels of cache misses etc. It also holds statistics of available memory. It is important for scheduler to take memory consumption/availability into account.

# IJobFactory

**IJobFactory** is responsible for job creation. We know continious factories (never runs out of work) or factories that is responsible for splitting work into several pieces (jobs) and joining them. Each factory has desired pool interval (when it is asked to create new jobs).

A factory can also provide properties:

* Estimation (**IEstimatableFactory)**
* Remoting **(IRemotingFactory)**

# IOptimizer

Optimizer is assigned to **IThreadControl** and can guide in the following way:

* Repair job estimations (for estimatable jobs) or makes estimation;
* Decides the job's assignment to WU with specific sureness level (can be asked to pick more than one WU)
* Prepares list for interuption/abortion of jobs with specific sureness;
* Listens to job events (start, end, memory consume, WU consume ...);