
Keywords: concurrency, multi-threaded computation, multi-processor architectures, models of concurrency, distributed processes, scheduling.

Distributed processing power increases every year, with new architectures or new network-distributed systems being developed. Current models are based on processes or programs, which are individually created by the developer, and this raises issues such as the following.

- Resources required by processes vary depending on the data that they deal with.
- The efficiency of concurrent processes varies according to the Operating System scheduler, how efficiently these processes have been implemented and their susceptibility to key issues such as starvation or deadlock.
- Poor programming practice means that concurrent programming is error-prone, and may not lead to efficient scheduling.
- At the programming-language level, constructs that control concurrency, such as synchronized methods in languages such as Java can be misused, which may create issues at scheduler levels, and might lead to starvation in a queue of processes.
- There are many algorithms for scheduling, but also many ways of measuring the efficiency of scheduling, which make it hard to decide on optimal policies for pre-emption and for scheduling in general.

The aim of this project is to investigate how a sheaf-theoretic semantics for concurrent processes can throw light on these issues, and possibly provide new insights for creating and scheduling processes. Sheaf-theoretic semantics was proposed by Goguen ("Sheaf semantics for concurrent interacting objects", *Mathematical Structures in Computer Science, 2:159-191, 1992"), and provides a basis for a simple and tractable model of concurrent objects. The approach is largely topological, but in a fairly abstract sense, where the topology may arise from intervals of time, different physical or logical locations, as in a multi-processor architecture, or even the different locks used by multi-threaded operating systems to synchronize access to resources such as busses, disks, etc. The main topics of this investigation would be:

1. using the semantics to create concurrency at compile-time
2. using the semantics to evaluate existing algorithms and strategies for pre-emption and scheduling of threads, and to develop new strategies
3. evaluating these strategies by incorporating them in existing compilers and (fragments of) operating systems.

For the first topic, I’ve been doing some study on basic category theory and sheaf theory and their use with concurrent programs, and well-known algorithm problems, which will provide a basis for creating a model of low-level concurrent processes using sheaf theory. Using this semantics and static analysis, I believe
that it is possible to identify expressions within a program during the compilation process that can be split into different processes without causing starvation or deadlock, in order to maximize the use of multiprocessor technologies and reduce resources used by processes and at the same time improve the efficiency of their execution.

For the second topic, the aim would be to study local properties of processes, such as the resources and locks required, and develop notions of independence and competition between processes that would allow independent processes to be scheduled concurrently on different processors to speed up execution times. Processes that compete for resources may be ordered according to existing techniques, or it may be possible to develop new orderings that will provide novel approaches to scheduling, or new measures for evaluating the efficiency of scheduling.

Regarding the third topic, it would be interesting to implement the results of the research into Posix threads system since they can be currently used in different operating system such Microsoft Windows and Linux, mainly at the Linux level, and verify how well this approach would work with their efficient scheduler implemented by IBM.

Or maybe to use SUN java compiler that is an open source java compiler that uses the well-known java thread model.

Both these thread systems work on programming languages with well-defined syntax and semantics, would provide the best environment possible for the implementation and evaluation elements of the research project.

---------- first draft: ----------------------
Topics:

1- Concurrency issues with existing models
2- Possible parallel semantics.
3- Inclusion with current compilers and its application with Operating Systems.

1- Distributed processing power increases every year with new architectures, or new network distributed systems, current concurrent theory seems inappropriate, current models are based on processes or programs, individually created by the developer, this generates different issues such as:
   - Processes sizes vary differently depending on the data that they deal with.
- Concurrent processes efficiency varies based on the Operating System scheduler, how efficient these processes were implemented and prepared against main issues such starvation or deadlocks.
- Human made concurrent processes create inconsistency with process size and are error prone.
- Automated Synchronized methods given by Language such as Java can become easily ineffective, what will create issues at scheduler levels, and might lead to starvation in a queue of processes.
- Hard to predict pre-emption and the best scheduler.

These are the main problems that exist today with concurrent programs, and while parallel processing increases everyday this area seems to improve slowly, whereas the developer based on classic concurrency theories creates these processes. But if processes were created at the level of the compiler looking at its semantics it seems to be possible to achieve a good consistency size, starvation and deadlock avoidance. Allowing at the same time the developer to concentrate in what matters, the software.

2 - If processes achieve size and time consistency scheduler implementation can be simpler and more efficient without the need to calculate complex averages or big approximations, and making possible a true multiprocessor and distributed process implementation.

Using category theory and shieve theory I will study concurrent models applied to their semantics for automated process creating in order to maximize current schedulers and allow better approximation algorithms to be created. And try to create model that will define these concurrent semantic processes.

Using semantics I believe that its possible to identify expressions within a program at the compilation process that can be split into different processes without occurring starvation or deadlock, to maximize the use of multiprocessor technologies and reduce processes size and at the same time their process time.

3- It would be interesting to implement the results of the research into Posix threads system since they can be currently used in different operating system such Microsoft Windows and Linux, mainly at the Linux level, and verify how well this approach would work with their efficient scheduler implemented by IBM.

Or maybe to use SUN java compiler that is an open source java compiler that uses the well-known java thread model.

Both these thread systems work on programming languages with well-defined syntax and semantics, would provide the best environment possible for the implementation of the research project.

Currently I’ve been doing some study on base category theory and shieves theory and their use with concurrent programs, and well know algorithm
problems, what would provide an advantage while creating a concurrent model using shieve theory.

I believe that it's necessary to leave the current models of signals and slots used for processes and implement concurrency at lower levels.