Cooperating Resource Managers

Hermann Härtig Lars Reuther Jean Wolter
Martin Borriss Torsten Paul

Dresden University of Technology
Department of Computer Science
01062 Dresden, Germany
{haertig,reuther,jw5,borriss,paul}@os.inf.tu-dresden.de

Abstract

Operating System designers currently face two major challenges. One is the coexistence of time-sharing and dynamic real-time components on a single machine and even in the same application. The other is that many acceleration techniques (e.g. caches) used in modern hardware architectures work very well for the average case, but not for the worst case which counts for real-time systems.

This scenario is addressed by DROPS, the Dresden Real-time OPerating System. DROPS uses resource managers which have been and still are being built in an ad hoc manner and that allow the reservation of resources, the enforcement of the reservations and the usage of unreserved or reserved but unused resources for other, non-real-time applications. While these resource managers have been built and work very well and are well published (see e.g. [5, 9, 3]), they all have their own, unique ad hoc interface. This paper describes a more unified, systematic approach that is applicable at various system levels and allows a systematic construction of applications using cooperating resource managers.

1 Introduction

Resource managers are building blocks for systematic, resource aware design and implementation of systems requiring some form of timeliness. A rather common form of timeliness is often called *Quality of Service* for media applications. In these systems, applications often consist of component chains processing a stream of data in real time, e.g. a chain consisting of a file systems that produces the data stream, an MPEG decoder and a video display. The main objective of the DROPS project is to provide such components as resource managers.

The general view taken is that components provide resources and use resources. To achieve timeliness in providing resources, the used resources must be provided in time. Since resources are limited they need to be reserved in advance to be made available in time.

The amount of resources needed to provide timeliness (or Quality of Service) varies. To guarantee timeliness, resources must be provided for the worst case. Unfortunately, the worst case / normal case ratio is rather bad

- for many applications, especially in the media applications and
- in cheap, but powerful hardware (caches, busses)[1].

Under these circumstances, a reservation for the worst case can result in very high unproductive blocking of resources. Fortunately, many applications, especially in the media applications area, exhibit an behavior that they can adapt to occasional violations of timeliness, e.g., by temporarily reducing the quality of a presentation.

This leads to a design process:

- provide guarantees for the hard core functionality
- provide promises for the normal case behavior and notification mechanisms in case the promises cannot be honored
- try to get the worst case as close as possible to the normal case (good worst case/normal case ratio) without thereby punishing the normal case too much.

The remainder of this paper is organized bottom up. Firstly, some examples are given of resource managers that have been built for DROPS. Next, a closer view is taken at resource aware design and the resulting requirements for the interfaces of resource managers are given. After discussing an example, some conclusions are drawn finally.

[1] Measurements done by the DROPS group show that for SCSI disks that situation is currently getting worse: on an IBM DGVS disk drive, an average case bandwidth of 15 MB/s can be achieve in contrast to only 2 MB/s under worst case assumptions

2 Examples of Resource Managers

Fig. 1 shows the principle architecture of the DROPS system. It contains on the left side L4Linux [5, 6], a version of the Linux operating system that runs on top of a micro-kernel and supports conventional non-real-time applications. It gets all resources that are not reserved for real-time components which are shown on the right side of the figure.

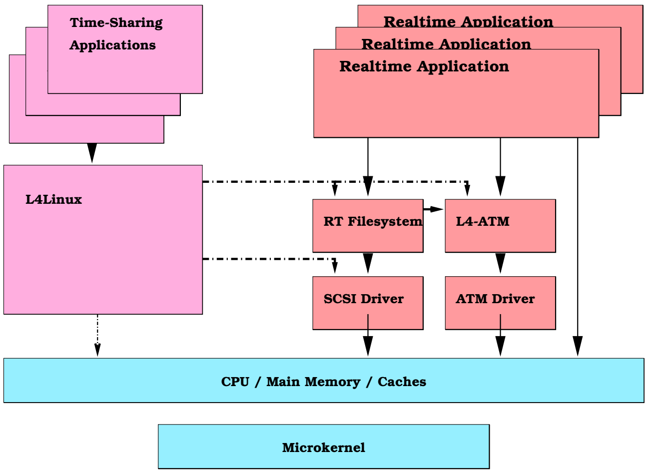

Figure 1. The DROPS Architecture

We introduce the general concept of resource managers by describing some of the resource managers in DROPS and the role they play in the DROPS architecture:

- The cache partitioning resource manager [9]
  allows to partition caches and to assign the resulting partitions either to real-time components or to L4Linux. This technique allows to eliminate the penalty caused by cache misses in level two caches which otherwise would be caused by context switches. This is especially important to bound worst case execution times for real-time processes that run besides other application processes. E.g., worst case execution times for processes like drivers, that have to maintain very short response times in spite of cache flooding as induced by L4Linux and its application, can be determined without taking cache floods into account. Measurements in [9] have shown that this may lead to significantly lower worst case execution times.

- L4ATM [3]
  allows to reserve bandwidth for some clients while leaving the remainder of the bandwidth to L4Linux and other clients. On admission, L4ATM computes the amount of resources (buffer and CPU-cycles) that are needed for the requested bandwidth, requests their reservation from the underlying resource managers for memory and CPU, and provides the reserved bandwidth using these reserved, lower level resources.

- The CPU-scheduler
  (which is still under construction) allows the reservation of resources in two levels, one nominating the resources that are effectively needed and are guaranteed, the other nominating *nice to have* cycles. An interface is provided to notify clients (i.e., threads) in the event that they cannot be given the *nice to have* partition of the resources. A simplified example is given in Fig. 2.

```
reserve(period, cycles, high priority);
reserve(period, cycles2, low priority);

while (!end) {
   if (begin_period(call_back_func, event) {
      decode_picture();
   } else {
      skip_picture();
   }
}

release_reservation();

void call_back_func(reason) {
   if (reason == TIME) {
      reduce_quality();
   }
}
```

Figure 2. Use of the scheduler as a resource manager

- The SCSI driver
  again allows to reserve bandwidth. The reservation is enforced by adding time stamps to real-time disk requests. Then, disk scheduling uses standard disk scheduling algorithms as long as all timed requests are foreseen to be executed in time. Only when one or more disk requests are endangered to be late, their execution is reordered to guarantee their deadline. Hence, if sufficient work ahead is planned and the overall load is kept below the maximum sustainable bandwidth, no penalties are induced by the real-time capabilities of the SCSI driver. Admission for real-time clients is done under worst case assumptions.

Other examples, which are under construction at the time of this writing, include a video display manager, a file system and a manager for PCI bus usage.

Resource managers and other real-time components cooperate in two ways. One is the mapping of requested resources of one level to resources of the next lower level. E.g., L4ATM heuristically computes needed CPU-cycles from the requested bandwidth and packet size. This can be viewed as vertical interaction. The other — horizontal interaction — occurs when actually passing a periodic stream of events through (mostly) a chain of real-time components, e.g., as shown in Fig. 3. The arrows indicate the direction of the flow of events. The streams of events must exhibit a certain behavior with regard to bandwidth and the jitter, which is defined as the maximal deviation from an exactly periodical stream. The properties of the stream are described using a set of parameter [4] and is part of the contract, that is made during the admission of a new client.

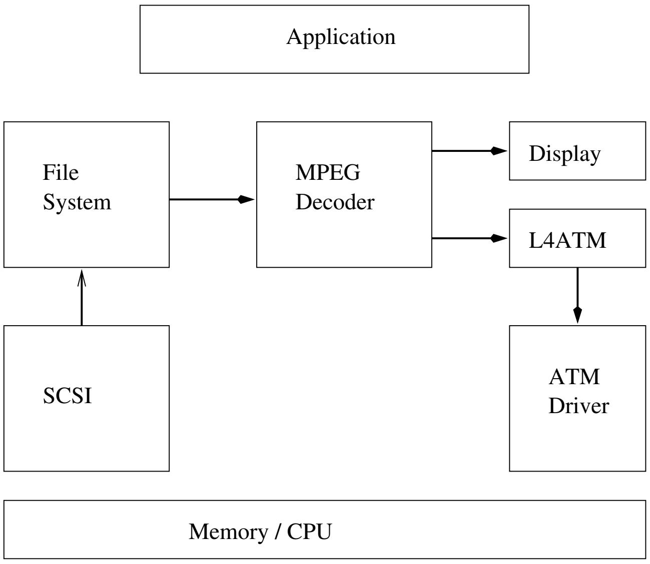

Figure 3. A Real-time Scenario

These resource managers have been constructed intuitively and lack an uniformed interface. E.g., while the scheduler notifies its clients if the availability of resources changes, which is considered a very important property for a large class of real-time applications where the value of the timeliness of events changes, the SCSI driver currently has no such interface. The purpose of this paper is to demonstrate a preliminary design of such an interface in the hope, that good interaction at the workshop will shorten the time needed to converge on a suitable interface.

3 Towards Interface Uniformity

Systems based on resource managers are set up in two steps that closely resemble the steps as taken in the setup of network connections with quality of service requirements, but had to be generalized for other, non-network related resources as well.

- In the first step (admission), all participating resource managers, e.g., the file system, the MPEG decoder, L4ATM and the Display manager in the example as shown in Figure 3, are requested to reserve the required resources at a certain priority; the requests are either granted or refused by the resource managers. If refused, the highest possible amount of resources is reserved and returned as a parameter. During this first step, lower level resource managers are queried in the same way if such resources are needed (e.g., the SCSI, main memory, and CPU managers are queried by the file system). If granted, the requested resources are reserved until the second step occurs, but only for a certain maximum of time. An entity is created to represent the reserved resources, depending on the type of the requested resource. (E.g. in the case of the file system or L4ATM, the entity is a thread, in case of the cache manager it is a partition identifier.)
  The resources can be requested with priorities. If in worst case situations not enough resources are there to grant all requested resources to all clients, then the priorities determine which client gets which resources.

- In the second step, the participating components are either started or aborted. The resources that are finally requested (they may be less than the originally requested reservations) and the upcall procedure for the event that the reservations cannot be honored are provided as parameters. Where necessary, communication partners, e.g. threads to send a stream of events to, are additional parameters.

Two interesting challenges are posed in the usage and construction of resource managers.

- Resource managers at different resource levels have different parameters to specify requested resources. An interesting challenge is to systematically derive mappings from higher level resources to lower level ones, this mappings need to be done within resource managers.

- To reserve resources for the worst case will result in enormously bad usage of resources. As a simple means to support adaptation with regard to CPU usage, reservation priorities have been introduced pragmatically, where high priority reservations take precedence over lower priority reservations and clients with low priority reservations are notified if these reservations cannot be honored. However, the generalization towards other resources and their use for smooth adaptation in application remains to be discussed.

These challenges will be discussed in the following and some properties of unified interface are derived.

3.1 Mapping of Resources

So far we have discussed two levels of resources, the low level physical resources and the higher level resources such as a file system and L4ATM. We describe how reservations are requested for the higher level resources and how these reservations are mapped to the lower level physical resources.

Resource managers like file systems and protocols interact by forwarding periodic, jitter constraint streams of events. It has been mathematically proven in [4] that numerous parameter sets used for the purpose of describing jitter constraint event streams as e.g. the leaky bucket scheme used for ATM [2] or the LBAP model used in [1] can be expressed by a generalized model.

The model describes jitter constraint streams (slightly simplified for this paper) using three parameters:

- R^M : maximum bandwidth
- R^A : average, sustainable bandwidth
- τ : max deviation of a single byte from tR^A

The model of interaction is such that each component buffers the data belonging to the input stream, while transforming it to an output stream. Then, resource managers of this layer commit to contracts of the form: if an input stream conforms to an agreed description and if the requested resources are provided by lower layers, then the output stream will conform to another description as agreed upon.

An admission request may be of the following form. For an input stream with known characteristics, an output stream with some required characteristics is requested:

- $R^M_{in}, R^A_{in}, \tau_{in}$
  describes the input stream,
- $R^M_{out}, R^A_{out}, \tau_{out}$
  is a description of the required output stream.

Then, under the assumptions

- that work in the resource manager is done by a dedicated "worker thread", and
- that the number of CPU cycles needed in a component[2] is (roughly) known for a given bandwidth and a given duration τ:
  $Component.CPU(R^A_{in}, R^A_{out}, \tau)$

then the following resource requirements can be derived:

- the CPU-scheduling period τ_s for the worker thread must be less or equal than τ_{out}.
- the memory needed to buffer the incoming stream is $R^a * (\tau_s + \tau_{in})$
- the number of cycles needed per period is $Component.CPU(R^A_{in}, R^A_{out}, \tau_s)$

This estimates lead to a much more systematic approach regarding the planning of the resource usage.

[2]For L4ATM such a formula is derived in [3]

3.2 Adaptation

For practical purposes, it makes sense to do adaptation at several levels:

- Short term resynchronization
  is used to adapt the data input and output interfaces of linked components. E.g, the receiver of a stream can force the sender to delay some events if it cannot keep pace with the sender.
- Temporary adaptation
  allows applications to react on temporary unavailability of resources with lower reservation priorities. For each resource, an upcall must be provided to notify clients of resources about the unavailability or the new availability. This is straight forward for CPU.
  For resources providing mainly bandwidth it usually does not make much sense to reduce the bandwidth. E.g., if a file contains MPEG encoded data, a simple reduction of bandwidth does not enable the MPEG decoder as a client to skip intermediate frames. It is also not advisable to interpret a file format at the file system to distinguish the priority of data within files. Hence, a straight forward solution is to put high priority and low priority data in different files and requesting bandwidth for that data with different priorities. Then, if a certain file temporarily cannot provide access with the promised bandwidth, parts of the data are simply skipped. The induced resynchronization has — in absence of information in the file system — to be done at the application level.
- Renegotiation
  of reservations may become necessary if — in contrast to these short intervals of unavailability — long term changes are occurring, e.g. if additional high priority load is put onto a system. In that case, interfaces are needed to renegotiate the admission.

3.3 Resource Manager Interface

Based on the experience with the intuitive design of various resource managers in DROPS, a Resource Manager Interface was defined. It consists of two parts: a management and a data interface. Fig. 4 shows the interaction of these interfaces.

3.3.1 Management interface

This interface must be implemented by every component of the DROPS system. It has several tasks:

- Admission Control
- Resource mapping
- Short term adaptation
- Long term adaptation (renegotiation)

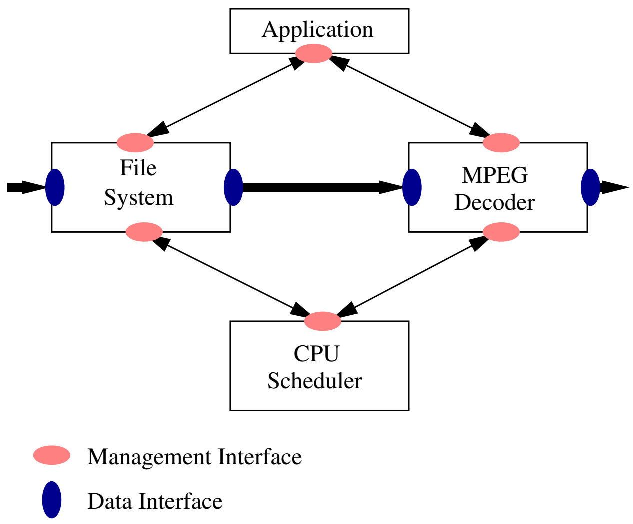

Figure 4. Resource Manager Interface

3.3.2 Data interface

This interface must only be implemented by components which may be a part of a component chain, e.g. the file system or the MPEG decoder. Its tasks are:

- the transport of the application data stream
- the synchronisation of the components

4 An extended example

Fig. 5 shows a part of the component chain described in section 2.

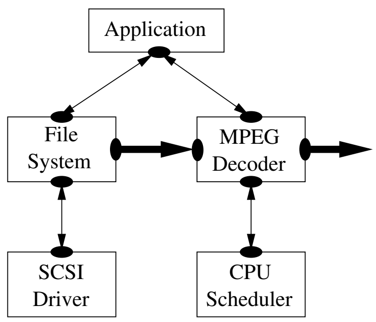

Figure 5. An example for the usage of Resource Managers

If the application wants to create a new video stream, it performs the following steps:

1. It calls the appropriate *open*-functions of the management interfaces to open the MPEG stream at the file system and to create a new decoder thread. To perform the admission test, the file system requests the required bandwidth at the SCSI driver; the decoder calculates the required CPU time according to the resolution and the frame rate of the MPEG stream and requests it at the CPU scheduler. If these reservation requests can be fulfilled, the open calls return successfully, otherwise the requests are rejected.

   The reservation requests to the SCSI driver and the CPU scheduler contain an callback function, which is used to inform the file system and the decoder of a resource shortage.

2. If all open requests return successfully, the application connects the file system and the decoder and starts the data transfer.

4.1 Adaptation

To enable the short term adaptation, a component must inform the resource manager of its different requirements. The MPEG decoder may be able to reduce its CPU requirements by displaying only the I-frames of a MPEG stream and skipping the P- and B-frames. Therefore, the decoder requests the CPU time required to display the I-frames at a high priority. This reservation must be fulfilled at every time. Additionally, the decoder requests the time required to display the P- and B-frames at a lower priority, this reservation may be missed sometimes.

While processing the MPEG stream, the decoder tries to display all frames of the stream. If the CPU scheduler detects that the decoder exceeds its time slice, it uses the callback function to notify the decoder about it, which then skips some frames.

The file system uses a similar mechanism to handle resource shortage. The different frames of the MPEG stream are stored in separate files. The bandwidth required to read the file of the I-frames is requested at a high priority while the bandwidth required by the P- and B-frames is requested at a lower priority. Thus, the file system can skip some disk requests of the P- and B-frame file if the SCSI driver detects the possible violation of deadlines of disk requests.

4.2 Renegotiation

If the CPU scheduler or the SCSI driver detect a permanent resource shortage, a renegotiation of the reservation is required. The MPEG decoder can reduce its requirements by scaling down the resolution of the stream.

The renegotiation can also be used if a new stream is created by the application, which high priority requirements cannot be fulfilled with the current reservations.

5 Related Work

The Rialto Operating System [7] defines a model for distributed real-time resource management. A Resource

Provider exports two functions: it provides the resource (the data interface) and it provides an operation to determine the required lower level resources (the resource mapping part of the management interface). This operation is used by a central Resource Planner to perform the resource negotiation. Rialto partly provides adaptation, e.g., the CPU scheduler described in [8] uses a similar mechanism to that discussed in section 2, but resources cannot be allocated at several priority levels.

A model for adaptive resource allocation is described in [10]. The resource requirements of an application are described by a Resource Usage Model (RUM). An adaptive application has several RUMs, a Resource Controller selects an applicable one and performs the corresponding reservations. The long term adaptation described above is similar to this adaptation model; the RUMs describe the results of the renegotiation.

6 Conclusions

Based on resource managers which have been built in the DROPS project, a somewhat more systematic approach to design resource managers and to built application systems based on such resource managers has been discussed in this paper. It is characterized by taken the usage of resources into account from very early stages in the design process.

References

[1] D. P. Anderson. Metascheduling for Continuous Media. *ACM Transactions on Computer Systems*, 11(3):226–252, August 1993.

[2] ATM Forum Technical Commitee. *ATM: Traffic Management Specification, Version 4.0*, 1996.

[3] M. Borriss and H. Härtig. Design and Implementation of a Real-Time ATM-based Protocol Server. In *19th IEEE Real-Time Systems Symposium (RTSS)*, Madrid, Spain, Dec. 1998.

[4] C.-J. Hamann. On the quantitative specification of jitter constrained periodic streams. In *MASCOTS*, Haifa, Israel, Jan. 1997.

[5] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter. The Performance of μ-Kernel-based Systems. In *16th ACM Symposium on Operating System Principles (SOSP)*, pages 66–77, Saint-Malo, France, Oct. 1997. Paper and slides available from URL: `http://os.inf.tu-dresden.de/L4/`.

[6] H. Härtig, M. Hohmuth, and J. Wolter. Taming Linux. In *Proceedings of the 5th Annual Australasian Conference on Parallel And Real-Time Systems (PART '98)*, Adelaide, Australia, Sept. 1998.

[7] M. B. Jones, P. J. Leach, R. P. Draves, and J. S. B. III. Modular Real-Time Resource Management in the Rialto Operating System. In *5th Workshop on Tot Topics in Operating Systems*, May 1995.

[8] M. B. Jones, D. Rosu, and M.-C. Rosu. Cpu Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *16th ACM Symposium on Operating System Principles (SOSP)*, pages 198–211, 1997.

[9] J. Liedtke, H. Härtig, and M. Hohmuth. OS-controlled Cache Predictability for Real-Time Systems. In *Third IEEE Real-time Technology and Applications Symposium (RTAS)*, pages 213–223, Montreal, Canada, June 1997.

[10] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1997.