Proposal of an External Processor Scheduling in Micro-Kernel based Operating Systems*

Winfried Kalfa+

TR-92-028

May, 1992

Until now, the management of resources was a task of the operating systems kernel. The applications running on the operating system were in general, similiar to each other. Thus the limited policy of the resource manager could satisfy the demands of applications. With the advent of computer systems capable handling new applications such as multi-media and of new operating systems based on micro-kernels and supporting object paradigm in a distributed environment, an external resource manager became important for both traditional operating systems like UNIX® with new applications and new object oriented and micro-kernel based operating systems. In this paper an approach to an external scheduling on the basis of the operating system BirliX is given. The proposal is based on a scheduler implemented in the user space. Problems of the implementation are described by means of the operating system BirliX as an example. Because the operating system is a distributed object-oriented operating system, our proposal deals with local and distributed managers. Coming from a system model of the BirliX, a resource model, and a process model, the scheduling model is developed.

---

* This research is supported by the Deutsche Forschungsgemeinschaft, and the International Computer Science Institute. The views and conclusions contained in this document are those of the author, and should not be interpreted as representing official policies, either expressed or implied, of any of the sponsoring organizations.

+ The Tenet Group, Computer Science Division, Department of EECS, University of California, Berkeley, and International Computer Science Institute. On sabbatical leave from Dresden University of Technology, Germany.

1. Introduction

The increased performance of computer hardware enables one to use computers for up to now not usable applications. For example, recent applications are large scale production control and multi-media systems. The latter is characterized by support of audio and video input/output. These continuous media have strong constraints. Traditional operating systems, such as UNIX, will still be used a long time because of the large number of software running on these operating systems. The current common purpose operating systems cannot satisfy these special constraints. The increased performance is not so large, that the scheduling of resources does not play a role. Figure 1 shows a situation in which applications are possible, but in which resources are scarce.

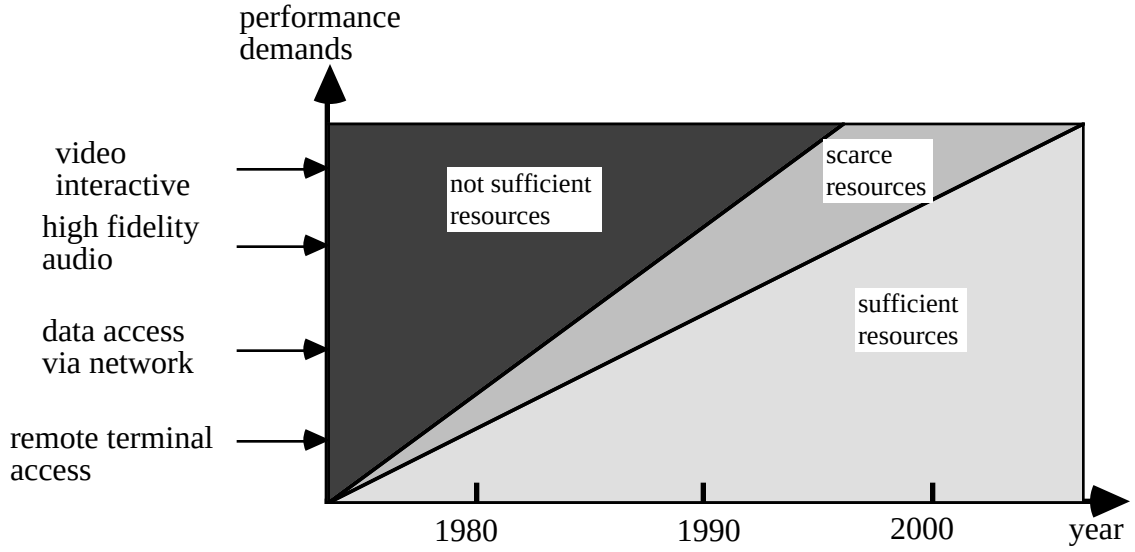

Figure 1: Performance requirements of applications/1/

That means the resource management needs a redesign. With regard to the existing software, the operating system interface must not change. This allows only a realization of the new resource management outside of the kernel in the user space with the additional advantage that there is no switching between kernel and user address space.

On the other side, the micro-kernel based operating systems pertain to state-of-art operating systems, such as Mach/2/, Amoeba/3/, LOCUS/4/, Chorus/5/, Clouds/6/, PEACE/7/, DIMOS/8/, and BirliX/9/. These operating systems are characterized by keywords such as "micro-kernel", "distributed", and "object-oriented". "Object-oriented" means there is a rough granularity in the user space. All of the relations in the user space concern objects. But the objects consist of resources and activities. The kernel manages the objects as well as the resources and activities.

The resource management is often very simple and does not meet the demands of the application, hence it seems to be a good idea to schedule all or a portion of the resources outside the kernel in the user mode. This paper deals with the scheduling of the processor in the operating system BirliX. The present approach can also be used for the modification of old fashioned operating systems, in principle.

In our approach the applications themselves estimate the scheduling strategy as far as the resources suffice. The approach includes three phases:

1. In the first phase the application negotiates with the operating system to determine whether sufficient resources are available and whether the restricted quality of service will be guaranteed. This phase is not time critical.

2. The second scheduling phase is time critical.

3. The third phase terminates the resource scheduling for the application. The resource can now be scheduled for another application.

The next chapter presents a system model of BirliX, Chapter 3 the process model, and Chapter 4 the resource model. Chapter 5 discusses our approach of the external processor management.

2. System model

All of the above quoted systems are similar. Therefore, the operating system BirliX is given as an example of that class of operating systems. On the other hand we will implement a prototype using our approach.

Objects in the user or application space of BirliX are instances of abstract data types. The implementation structure of each instance is a so-called ***team.*** The teams consist of activities (light-weight processes = threads), and resources. Applications in BirliX are sets of interacting teams distributed among several loosely coupled nodes. Teams are the smallest units of identification, distribution, and communication. A distributed naming and locating service identifies and locates the teams. Teams communicate by (remote) procedure calls between agents (the light-weight processes in the team). This communication represents the functionality of the applications. From the view point of teams, BirliX pretends to be a single large system providing all of the resources. The teams are implemented in a persistent memory, so that the status diagram in Figure 3 describes the status of a team. Teams exist forever opposite to processes in traditional operating systems which exist only during their run time. Teams can either be active or passive.

Passive teams have a persistent data representation (program code, descriptor tables, access tables, *and* data) in the persistent memory. They continue to exist as long as they are referenced, and they can only be destroyed by an explicit service. As far as teams are passive they need only the persistent memory as a resource. As teams become active they need more resources, a

processor, and ports for the communication, to run the processes in the team and to provide the functionality. The idea of BirliX does not consider such resources as files, devices, signals, or messages. The first two can be delivered by teams, the last two are reduced to the remote procedure call mechanism.

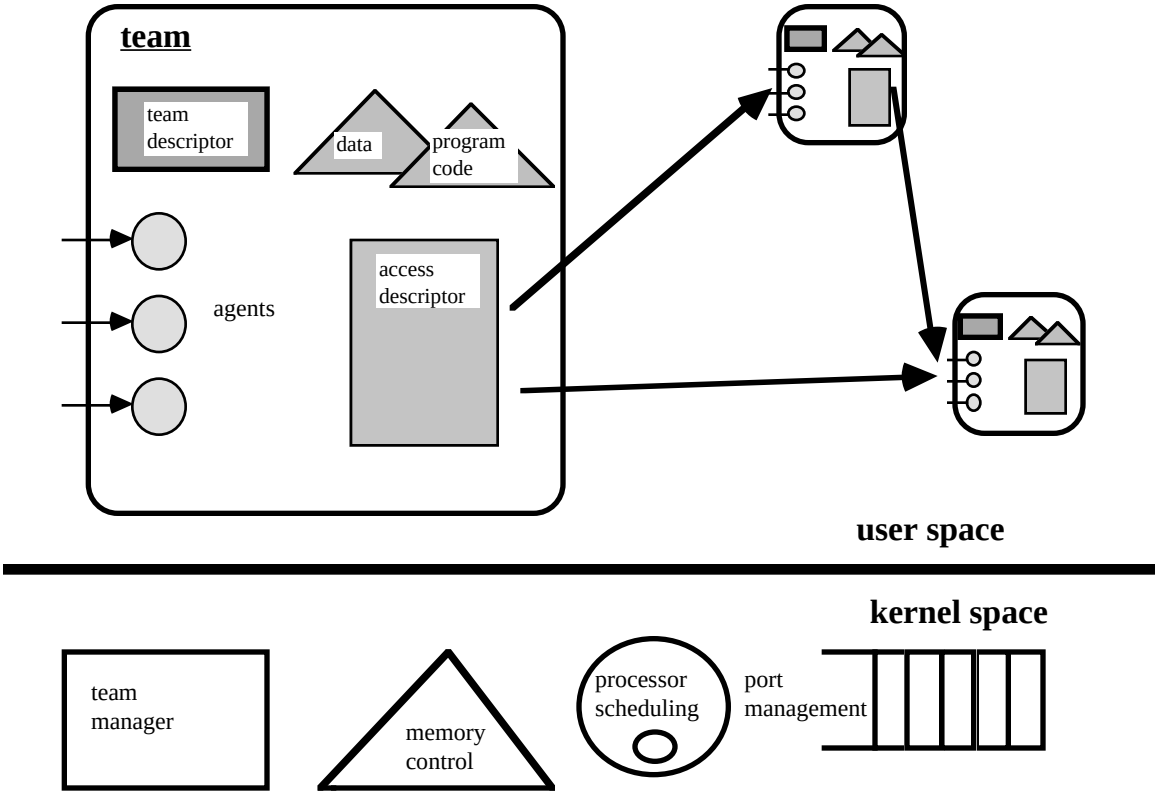

Figure 2: Objects in BirliX

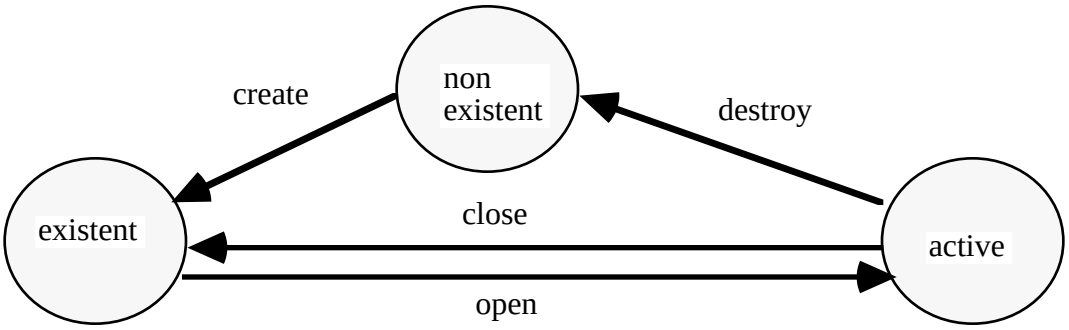

Figure 3: State diagram of teams

Thus we get the system model in Figure 4 for the goal of this paper. The teams need the resources

- processor
- memory
- message passing system.

This paper deals with the resource ***processor***; the other resources and their possible interaction remain for further works. The teams require the resource processor including special quality parameters. The resource management has to provide the resource with the required quality or reject the resource demand.

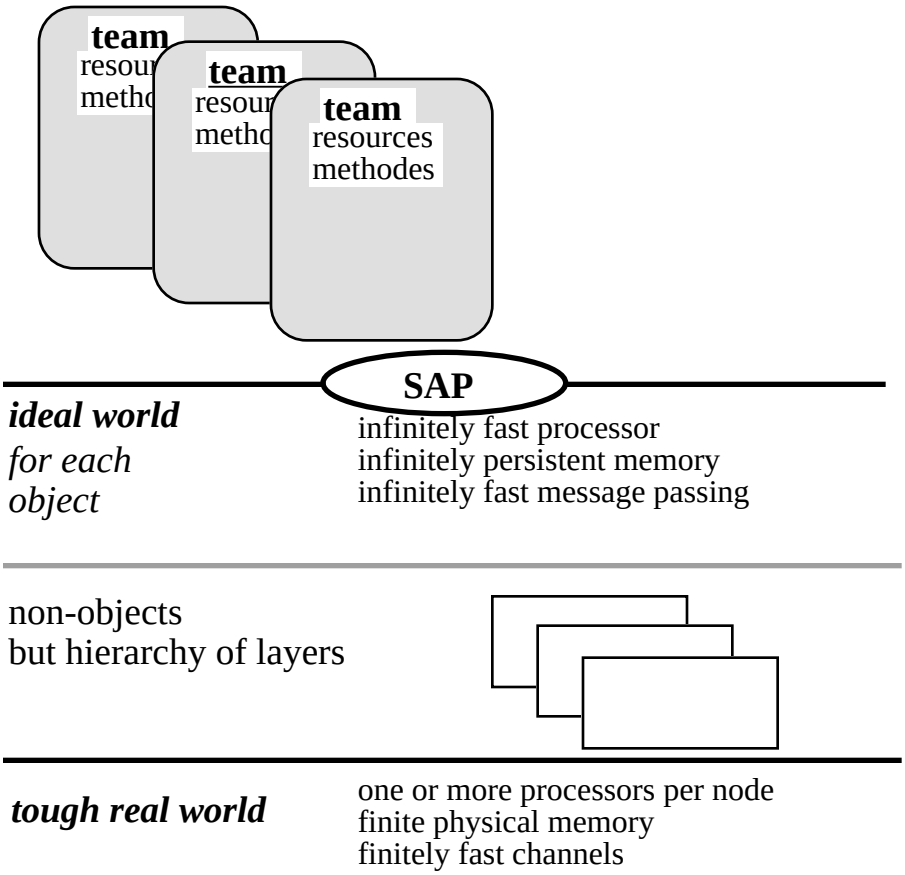

Figure 4: System model

3. Resource model

Let us assume the team activity as given. We will discuss it in the next chapter. If activities have ***all*** of the needed resources, they can continue. Activities and resources must belong to the same layer in a system. Activities in the application layer of traditional operating systems know

files, but no tracks on a storage medium. In contrast, an interrupt activity in the operating system kernel knows the signals of a controller, but no files. A resource is considered as a dupel

resource ::= (identifier, value).

Because computers carry out operations in discrete moments we mention ***time*** as a period of more than one such moment.

There are re-usable resources (processor, memory) and consumable resources (signals, messages) with regard to time. Re-usable resources have unique identifiers and variable values. The values of re-usable resources are time independent, but variable (processor register, memory) or a function of time (transmission lines, buses). Consumable resources are produced and consumed, the identifier as well as the value.

Any resource able to change values of itself or another resource is called a processor. These processors are driven by a "hard-wired" control (clock) or a programmable control (CPU, I/O-controller) and operate independently of each other. The processors, like some other resources (transmission line, printer) need be used exclusively, because the simultaneous usage by more than one activity could cause irregularities. Thus, there are in computer systems many activities simultaneously, but at one time one processor belongs only to one activity. Processors can only be multiplexed in time, i.e. in one time the processors belongs to one activity, in another time to another activity. Other ressources can be used by more activities at the same time, either the activity reads only values (counter of a clock) or the resource is divisible in more parts assigned to distinct activities.

Resources are characterized by a lot of features:

- address space of own variables or the variables of other resources: processor
- value space (grain) : memory
- transformation (values per sec) : processor
- transmission (values per sec) : transmission line
- exchange of values of own variables (values per sec) : processor
- instruction set : processor

We mention that the resources are of a different kind in a computer system. A good way to describe this is with a hierarchical layer model /10/.

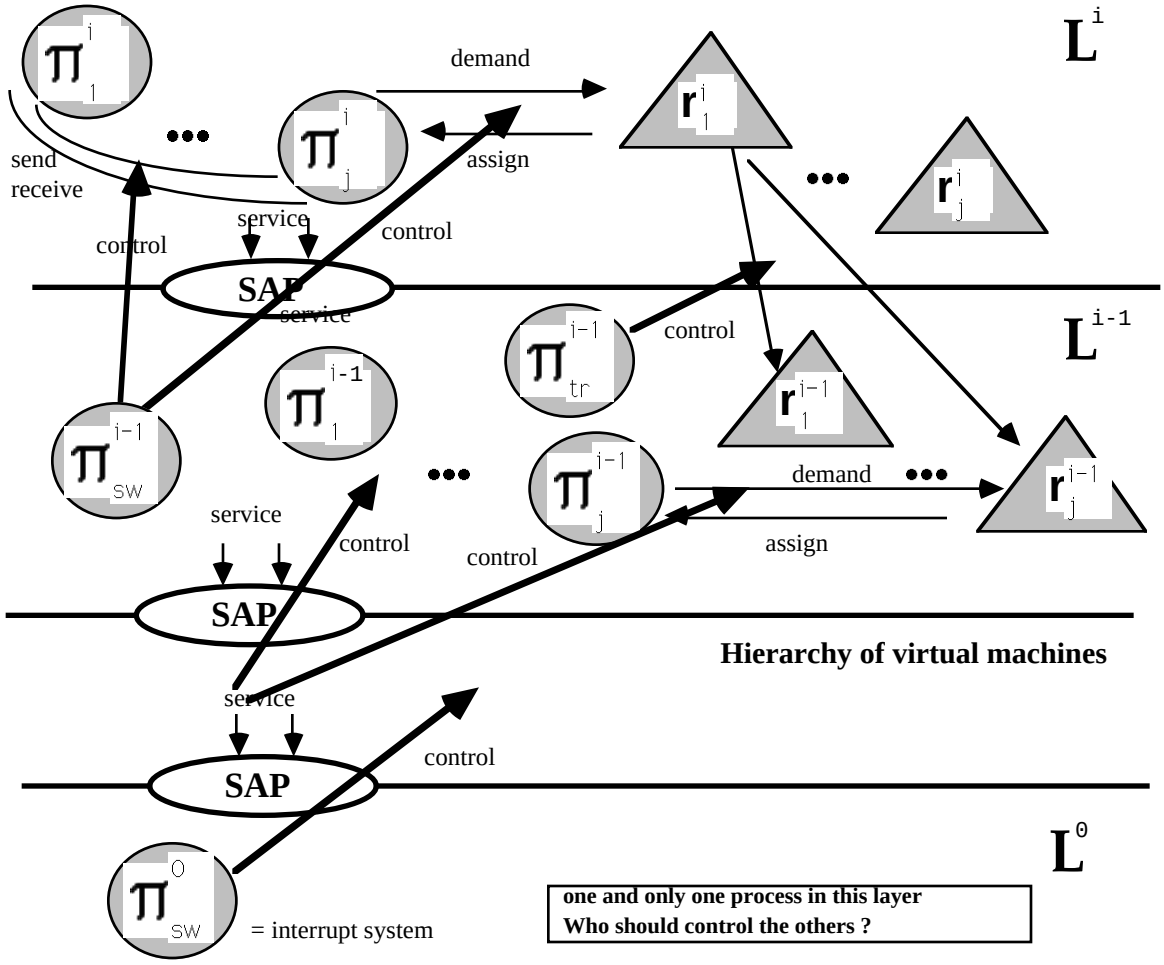

Figure 5: Hierarchical layer model of activities and resources

For example the resource file on the highest layer is transformed down on tables (e.g. i_nodes in UNIX) and external storage, the kinds of resources in different layers are different. In another example only the attributes of the resource are changed. The processor of the layer S^{i-1} with the transformation rate 10 mips is divided in 10 virtual processors in the layer S^{i} with the rate 1 mips to each virtual processor. An alternative approach consists in a 1:1 transformation from S^{i-1} to S^{i} and a scheduling of the only resource to the activities. We will discuss both approaches in Chapter 4.

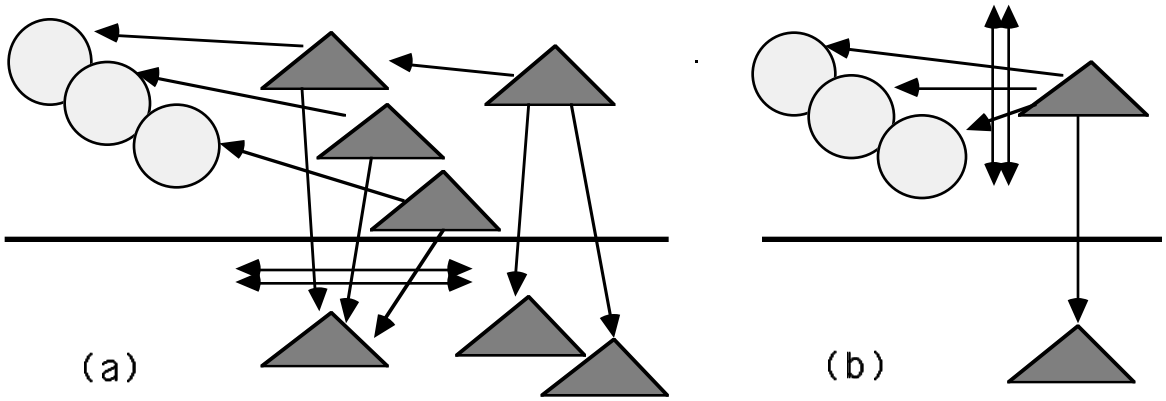

Figure 6: (a) Transformation, (b) scheduling of the resource processor

Because an activity must have all needed resources, there are interferences between the assignments of resources. In this paper we will not take this into account. We will deal only with the processor characterized by the following attributes:

- fixed identifier
- re-usable
- exclusively usable
- multiplexing in time
- m units of interchangeable processors
- further special attributes.

4. Process model

Activities in the teams of BirliX are the so-called natives and agents. Natives only need the processor, memory for program code and data, and special hardware resources dedicated to the native. Agents use as resources additional access tables and ports. All activities of a team are running in the same address on one and only one processor. In this manner they are light-weight processes, so that we will refer to them as processes. The syncronization of all teams and natives is realized by the team itself, e.g. by a monitor.

We consider a process as a sequence of actions. Each process has at any one time a vector of resource demands R^D and a vector of resource assignments R^A. If

$$R^D \leq R^A$$

is true, the process is running. We presume the process has all other resources without the processor. The interdependence of demands of more than one resource is subject to further research in the future, especially the interdependence between processor scheduling and pagefault handling. Another subject is to include parallel running processes in one team. But we will consider more than one processor in a node, and the kernel processes as well.

The input/output processes need the CPU and an input/output processor. In fact there are two communicating processes (Figure 7). One process needs the CPU processor, and the other the input/output processor. The communication happens by means of a shared memory, e.g., registers in the controller or in the CPU address space mapped memory on the controller. The interrupt system or a polling procedure manages the protocol between both processes. Thus each process needs only one processor.

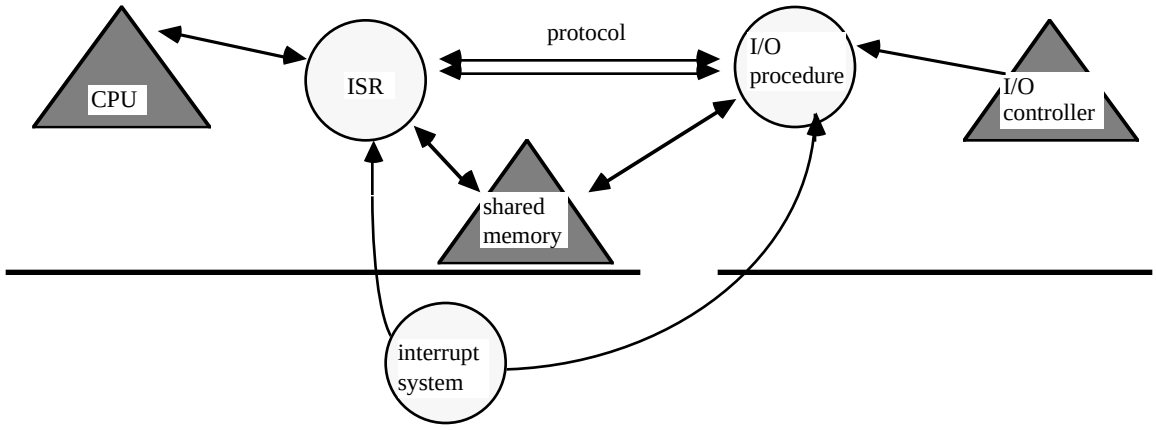

Figure 7: Separation of input/output processes in two processes
(ISR: interrupt service routine)

In order to decide when and how long a processor is to assign to a process, the scheduler needs some information provided by the processes. The laxity for the scheduler can only arise from the processes themselves, i.e., the applications.

Let us follow the notification in /11/ in the description of time parameters illustrated in Figure 8.

At the arrival time (creation of a new process or demand of a running process for a new scheduling strategy) a process tells the scheduler about its existence and all information of the future demands on processors. The more information and the earlier the process provides the information, the better the scheduler can make a schedule. Because our system is an open system, in which new processes are created and existing processes ended, we exclude making a complete schedule before the first process runs.

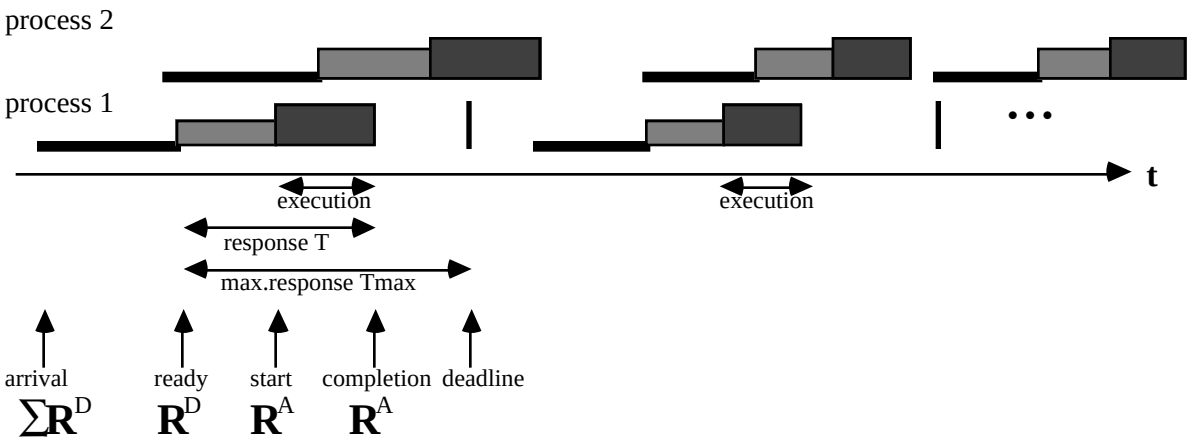

Figure 8: Time parameters of a process

With regards to the completion time we classify the processes (better the applications) according with /10/ an /12/ in

		priority
real time processes	$T \leq T_{max}$	1
work ahead processes	$T \leq T_{max} \mid T_{max} - t > T$	2
interactive processes	$T \leq T_{max}$	3
background processes	$T \leq \infty$	4

These unequations determine the goal of scheduling from site of the processes. Real time processes have to supply deterministic information, but it is hard for the process to predict in detail when, how long, and with what deadline they need the processor. In a large number of applications the same process recurs with a previously known period. Work ahead processes have a deadline, but are not critical to the time point **t** . Interactive processes supply only probabilistic information, and it is forbidden to background processes to put demands on the completion time.

5. Scheduling model

The demands of application for more flexibility led to operating systems with a microkernel and the object paradigm. New functions are simple introduced by creation of a new object class. But such systems, especially if they are distributed, are open systems with a permanent interaction with the environment. Contrary to technical systems, in with the goals are known, a priori, natural systems react on unexpected events with a change of goals and a reorganization. Therefore distributed object-oriented operating systems should not have a rigid resource

management, but should be able to react upon new objects with new goals and in connection with them with a new resource management.

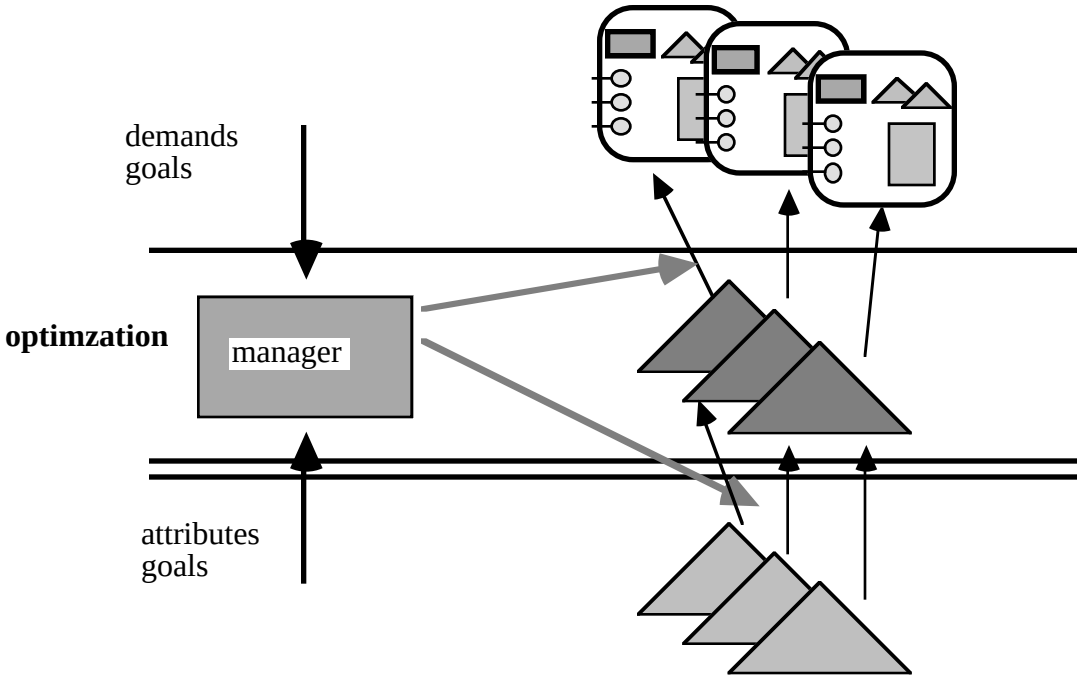

Figure 9: Scheduling model

The objects require resources with certain attributes. Their objective of optimization is to meet all of their requirements. On the other side the physical layer provides resources with the goal of maximum throughput. Thus the application of the system needs a manager to agree with both contrary goals. The manager is split in two parts, one transforms the physical resources to virtual resources, and the other part controls the assignment of the virtual resources to the processes. The management needs information, so there are two phases:

1. notification of later use of resources
2. scheduling

In traditional operating systems the scheduler follows rigid given algorithms. Perhaps some parameter, e.g., the priority, or the wanted class, e.g., "real time" or "background", can be specified, but not more.

Before a process will be running the resource manager has to know the scheduler. In the first negotiation phase, the process, calling the creation of the new process, requires a special virtual processor. Otherwise a running process wants to exchange the scheduling strategy. Thus the process negotiates with the processor manager. The required virtual processor is represented by a scheduling team. Before using a scheduler it must exist in the system (Figure 3). After its

opening the schedule team negotiates the opportunities of later scheduling with the calling process. The result is "yes", if the processor capacity is sufficient, or "no" in the opposite case.

In the second phase, the scheduling phase, the assignment of the processor to the process is managed, whenever an event appears. Such events are "soft" from another process or "hard" events from another processor. Especially, deadline scheduling or time slice algorithms need a clock processor. Input/output processors provide events as well.

6. Implementation

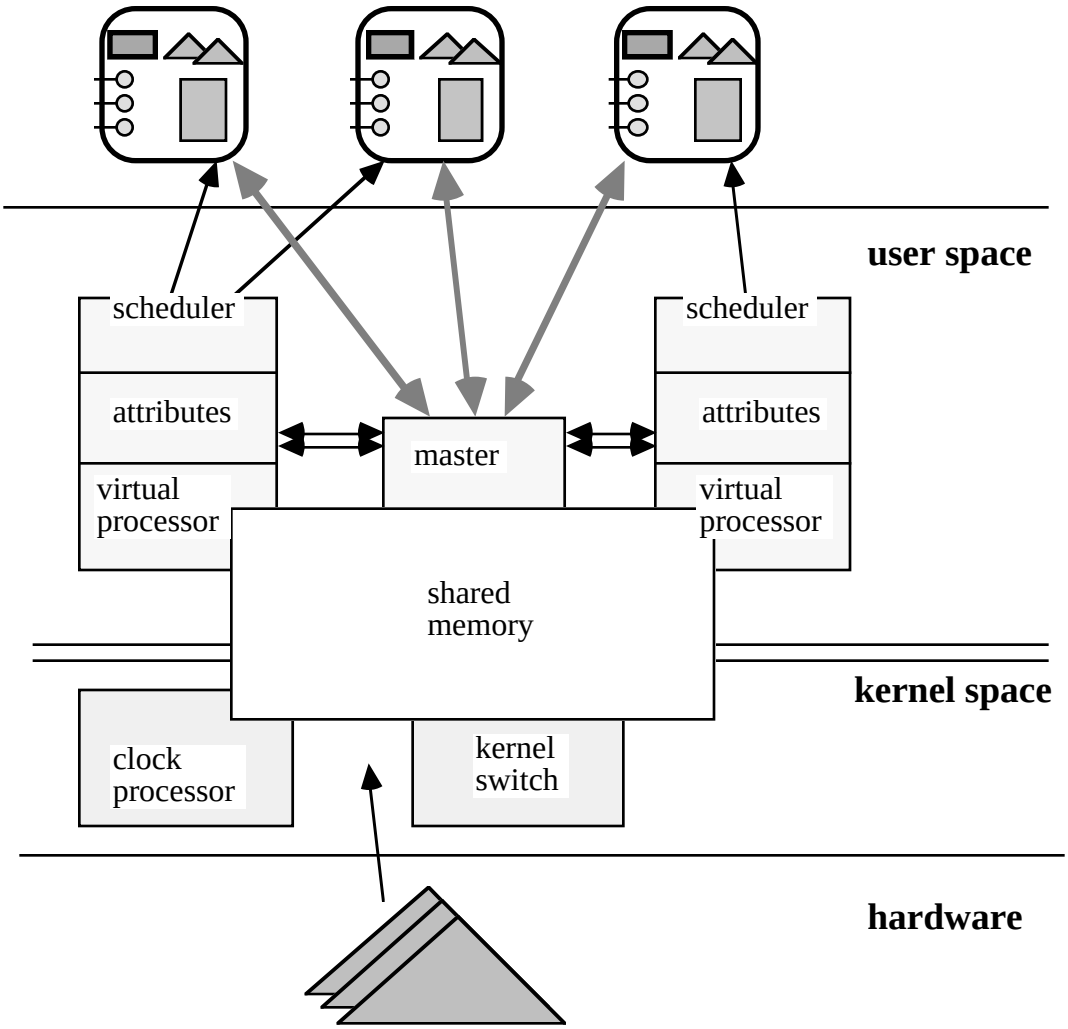

Figure 10: Implementation of the scheduler in BirliX

The schedulers are implemented as teams. Because the change of the address space is only possible in the kernel model the switching of it remains in the kernel. The information about all

scheduled processes is retained in a shared memory of all schedulers, master scheduler and kernel switch. The master scheduler negotiates with the processor requiring processes in the first phase. Specifically, it creates or opens a new necessary scheduler.

With that framework of scheduling we will have a testbed for

- implementation of distinct scheduling strategies
- measurements of the strategies
- classification of applications
- investigations of centralized and distributed scheduling
- investigations of interferences between scheduling distinct resources.

7. References

/1/ D.P.Anderson and R.G.Herrtwich: Resource Management for Digital Audio and Video. IEEE Workshop on Real-Time Operating Systems and Software, Charlottesville, May 1990.

/2/ M.Young, et al.: The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System. ACM Operating System Review, 21, 5.

/3/ S.T.Mullender, et al.: Amoeba: a Distributed Operating System for the 1990's. Computer 23,5.

/4/ G.Popek, B.J.Walker; The LOCUS Distributed System Architecture. MIT Press, 1985.

/5/ M.Rozier, et al.: Chorus Distributed Operating System. Computing Systems J. 1,4.

/6/ J.M.B.Auban: The Architecture of Ra: A Kernel for Clouds. Georgia Institute of Technology, Techn. Rep. GIT-ICS-88/25.

/7/ R.Berg, et al.: The PEASE Family of Operating Systems. GMD First Techn.Rep. 1991.

/8/ F.Krause, U.Schneider: Gestaltung des Kerns des verteilten Betriebssystems DIMOS. TU Dresden Proc. of the Workshop "Entwicklungstendenzen von Rechnernetzen" 1991.

/9/ H.Härtig, et al.: Architecture of the BirliX Operating System. Techn. Rep. GMD Birlinghoven, 1991.

/10/ W.Kalfa: Betriebssysteme. Berlin, Akademie-Verlag, 1990.

/11/ R.G.Herrtwich: An Introduction to Real-Time Scheduling. ICSI Berkeley Techn. Rep. TR-90-035 1990.

/12/ D.P.Anderson, R.Govindan: Scheduling and IPC Mechanisms for Continuous Media. Proc. of the 13th ACM Symposium on Operating Systems Principle 1991.

/13/ D.P.Anderson: Meta-Scheduling for Distributed Continuous Media. UC Berkeley EECS Dept. Techn.Rep. UCB/CSD/90/599 1990.