49. Internationales Wissenschaftliches Kolloquium
Technische Universität Ilmenau
27.-30. September 2004

B. Däne

Estimating Operating System Resource Occupation by Simulation

1. Introduction

Embedded systems frequently are limited in resources such as memory space and processing power. The limits come from technical (power consumption, space, weight) and economical factors. In the design process it is desirable to achieve information about resource demand as early as possible in order to optimize the design. Unfortunately there is no straight way to calculate such values. The results depend on many details, including the behavior of the controlled process and the real time operating system used. So some fundamental design decisions must be delayed or suboptimal results must be tolerated.

One solution is a design process that is primarily based on a model of the whole system, including control algorithms and behavioral models of the operating system and the controlled process. This paper will show how an operating system can be modeled in a general-purpose modeling environment and how quantitative data about resource occupation can be derived from simulation.

In the modeling process the general-purpose multi-domain modeling tool *MLDesigner* [1] of MLDesign Technologies Inc. has been used.

2. Modeling Real Time Operating Systems

The behavioral model of the real time operating system consists of a framework for modeling the individual tasks of the system, as shown in [2], and blocks that model kernel components. All these parts belong to *MLDesigner's* "DE" (discrete event) domain. But the tool also provides interfacing to model components that belong to other domains. So this model can be included into larger models that cover both the embedded system and the controlled process.

The individual tasks are primarily modeled by their timing behavior. For this purpose they are structured into atomic blocks with known time consumption. Other elements are control structures and kernel invocations. For more detailed description see [2] and [3].

3. Model Structure for Kernel Components

The model contains a number of blocks that model kernel components of the operating systems. Examples are schedulers, memory management, device management and message systems. While the task models have to be changed for each application, the kernel components remain unchanged for a given operating system with similar configuration parameters.

All kernel modules have the same principal structure. They contain a number of blocks with each dedicated to a certain function in this module. There are two views for each function: The “external” view, e.g. the interface to application tasks, and the “internal” view, e.g. the interface to other kernel components. Most functions are modeled by two blocks representing these different views to the same function.

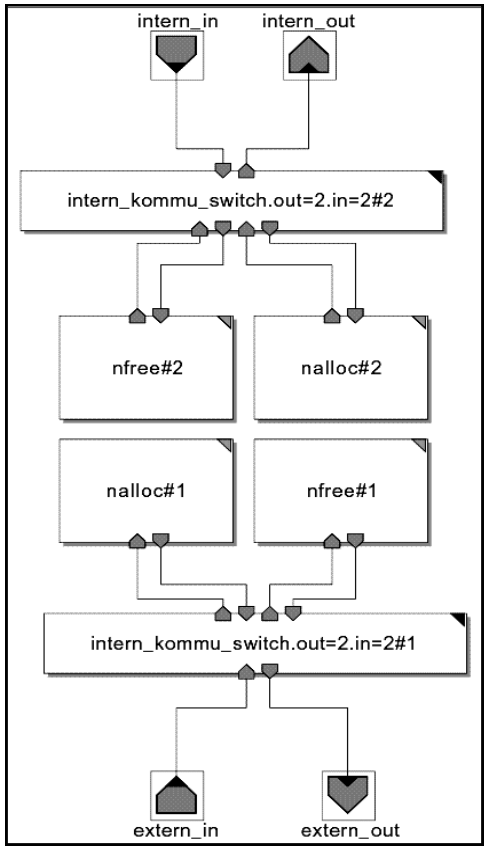

Fig. 1: Kernel module for memory management

The different functions of a kernel module are supervised by switching blocks. The invocation of functions is modeled by events that are attributed with a reference to the function called, parameters for the function call and a reference to the calling task or service. The switching blocks direct the events to the block that belongs to the function invoked and collect the completion events for returning them to the caller. There are distinct switching blocks for “external” and “internal” invocation.

Kernel functions in the model include task management for preemptive tasks and for rate-monotonic tasks, as well as device management and memory management. Fig. 1 as an example shows the top level structure of a kernel module for memory management. There are two functions only: allocation of a memory block (“nalloc”) and freeing a previously allocated block (“nfree”). As described above the “external” and “internal” views are modeled by distinct

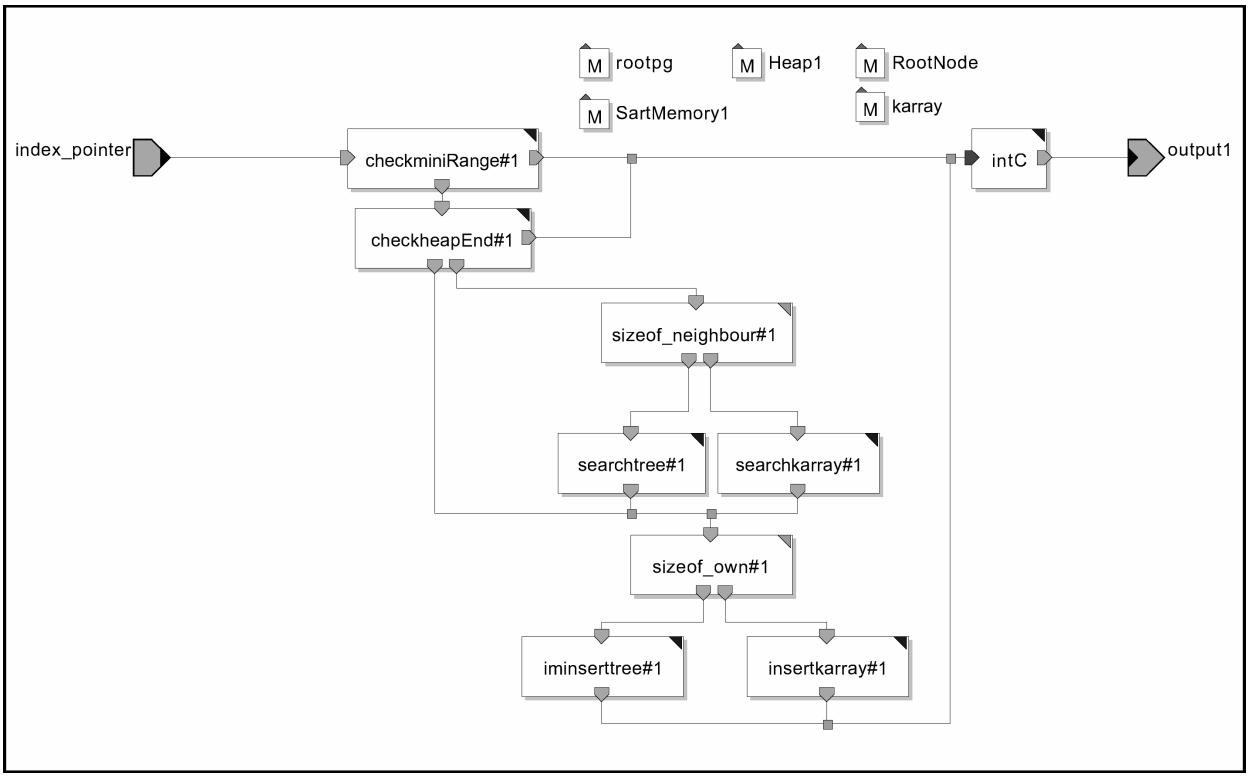

Fig. 2: Module “nfree”

blocks. Their links are hidden by using global data objects and global events available in the modeling tool.

Fig. 2 shows the internal structure of the block “nfree”. It is invoked by receiving events containing index pointers of the memory blocks. Some primitives handle the data structures that store the state of the memory management system. If a neighbor is already free, the free memory blocks will be merged. The data structures will be updated: an array (“karray”) holding information about actually allocated memory blocks, and a tree structure that points to free memory blocks.

The model allows the simulation of various memory allocation strategies in order to evaluate typical effects such as memory fragmentation.

4. Extracting Results from Simulation

All model components are equipped with attributes representing their resource occupation. Usage of processing time is modeled by atomic blocks with known time consumption. Effects of kernel functions to the scheduler’s behavior are modeled as part of the “internal” views of these functions. Other resource information is managed by the kernel modules dedicated to these functions. For instance the kernel module for memory management keeps track of the usage of

system memory and gets relevant information from other kernel modules via the “internal” interface.

During simulation all information is collected and interactively displayed by textual and graphical dialog elements of the modeling tool. As an example fig. 3 shows a snapshot of the log window for memory occupation during simulation. It lists block indices and sizes as well as properties, access counts and queue status. Similar displays exist for the list of free memory blocks (showing actual fragmentation) and for the other kernel functions such as task management and device management.

```
0.1534 index|size|lni-rni|counts|queue..
0.1534 0: -1: 1..1: 0:
0.1534 1: 100: 2..-1: 0:
0.1534 2: 18: -1..3: 0:
0.1534 3: 19: -1..4: 0:
0.1534 4: 21: -1..5: 0:
0.1534 5: 23: -1..6: 0:
0.1534 6: 32: 7..8: 0:
0.1534 7: 24: -1..10: 1: 109;
0.1534 8: 40: 9..11: 1: 133;
0.1534 9: 35: -1..12: 0:
0.1534 10: 27: -1..-1: 1: 283;
0.1534 11: 52: 13..-1: 1: 346;
0.1534 12: 36: -1..-1: 1: 310;
0.1534 13: 48: -1..-1: 1: 173;
```

Fig. 3: Example log window

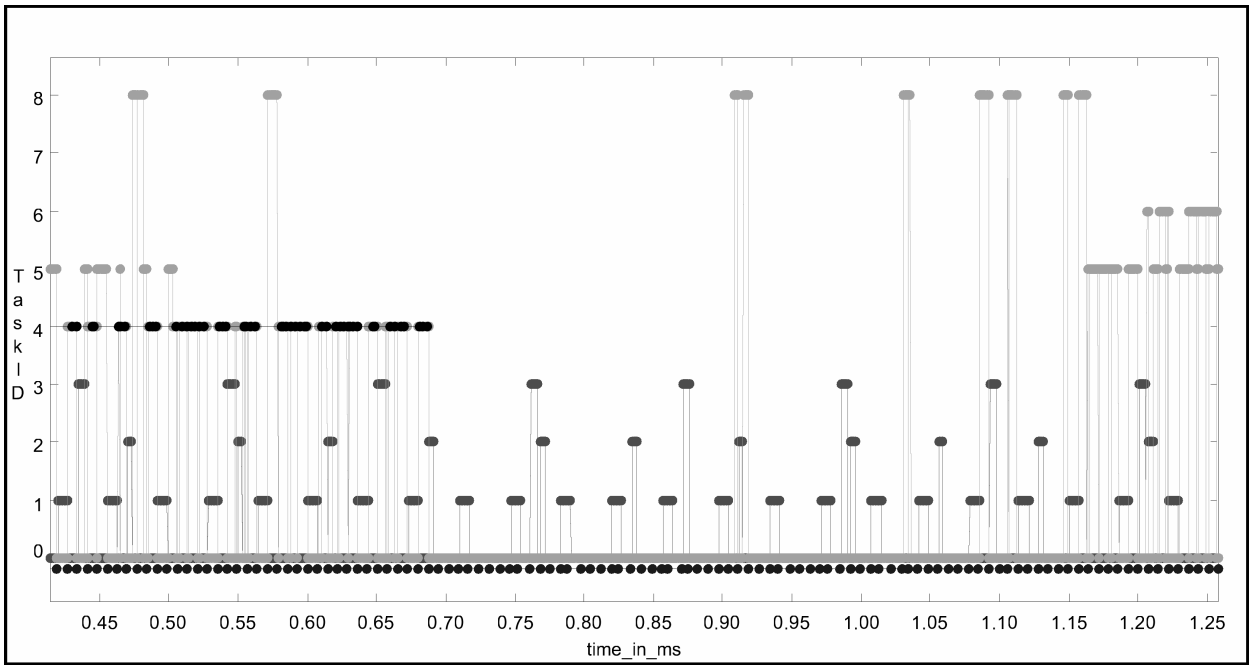

Fig. 4: Example task switch diagram

Other evaluation is available graphically. As an example fig. 4 shows a task switch diagram (zoomed part). It lists active task’s identificator (“TaskID”) versus model time. Task activity is shown by dots and bars. TaskID 0 denotes the Idle Task. The operating system’s tick period was 7.2 µs in model time. The ticks are shown by the lowermost row of dots. *MLDesigner* simulation took about 30 s simulation time per millisecond model time (on a PC with an 800 MHz Pentium III processor).

5. Conclusion

The paper shows how the behavior of operating systems can be included into models that are used when designing embedded systems. It has been shown how information about resource occupation can be derived, enabling optimization of the system early in the design process. Next work will be refinement of the modeling method, investigation of methods for implementing software directly from the model [4] and integration of all methods into more complex design processes.

6. Acknowledgements

This work is supported by the German Research Foundation (DFG) under SFB 622.

http://www.mldesigner.com/

References

[1] V. Zerbe, U. Freund, and H. Salzwedel, Mission Level Design of Control Systems, Proc. SCI/ISAS'99 Multiconference on Systemics, Cybernetics, Informatics, Orlando, USA, 1999, vol. 7, 237-243.

[2] B. Däne, W. Fengler, and F. Berger, Modeling and Simulation of Operating System Behavior, Proc. MSO 2003, IASTED International Conference on Modeling, Simulation and Optimization, Banff, Canada, 2003, 78-81.

[3] K. Rimbach, Model Based Analysis and Evaluation of Real Time Operating Systems (Ilmenau, Germany: Diploma Thesis 2002-12-02/054/IN97/2231, Ilmenau Technical University, 2002).

[4] B. Däne and W. Fengler, Implementing Mixed Discrete-Continuous Models into Realtime Environments, Proc. MIC 2004, IASTED International Conference on Modelling, Identification, and Control, Grindelwald, Switzerland, 2004.

Author

Dr. Bernd Däne
TU Ilmenau, P.O. Box 100565
98684 Ilmenau, Germany
Phone: +49-3677-69-1433
Fax: +49-3677-69-1614
E-mail: bernd.daene@tu-ilmenau.de