Teaching operating systems: windows kernel projects

Conference Paper · March 2010

DOI: 10.1145/1734263.1734429 · Source: DBLP

3 authors, including:

Andreas Polze
Hasso Plattner Institute
182 PUBLICATIONS **1,176** CITATIONS
SEE PROFILE

Dave Probert
Microsoft
8 PUBLICATIONS **68** CITATIONS
SEE PROFILE

Teaching Operating Systems – Windows Kernel Projects

Alexander Schmidt
Hasso Plattner Institute at University of Potsdam
Operating Systems and Middleware
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
alexander.schmidt@hpi.uni-potsdam.de

Andreas Polze
Hasso Plattner Institute at University of Potsdam
Operating Systems and Middleware
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
andreas.polze@hpi.uni-potsdam.de

Dave Probert
Windows Kernel Architecture
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052, USA
dave.probert@microsoft.com

ABSTRACT

When studying operating systems, students need to understand **u**ser-mode system interfaces (U), they need to learn about tools to monitor and **m**easure OS behavior (M), and they finally should understand central implementation details of the OS **k**ernel (K). Following the *UMK approach*, even complex projects such as modifying the memory management inside the Windows kernel can be carried out in an undergraduate OS curriculum.

Here we concentrate on the kernel- and measurement part and present the Abstract Memory Management (AMM) project. AMM provides a framework for modifying the working set management in Windows while still hiding many implementation details of the kernel. AMM has been used in OS courses at U of Washington Bothell and HPI/U of Potsdam, Germany, with very good results.

The AMM lab – together with other labs – is based on the Windows Research Kernel (WRK) as available in source from Microsoft. These labs complement our previously developed Curriculum Resource Kit (CRK) and are available for download.

Categories and Subject Descriptors

D.4.0 [**Operating Systems**]: General—*Microsoft Windows NT*; D.4.2 [**Operating Systems**]: Storage Management—*Virtual Memory*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer Science Education*

General Terms

Algorithms, Experimentation, Human Factors

Keywords

Operating Systems, Windows Sources, Memory Management, Windows Research Kernel

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

1. INTRODUCTION

The roots of Windows reach back to the late 1980s. Back then, many interesting things were happening in the operating system design space – including SVR4, the Mach microkernel, innovations in networking and windowing systems, and many research projects on OS fundamentals. The desire to gain in-depth knowledge of these exciting developments motivated many CS students to study operating systems back then. With our OS projects we want help to re-spark interest in operating systems again.

Within this paper, we advocate a hands-on approach towards teaching (and learning) OS concepts. We present our experiences from teaching Windows-based OS courses during the last ten years. We suggest a three-phase scheme, where students first learn to master **u**ser-mode system interfaces (U) – often referred to as "system programming". Secondly, they need to master principles and tools to monitor and **m**easure OS behavior (M). And third, students should be presented with central implementation details of the OS **k**ernel (K). Following the *UMK Approach*, even complex projects such as the modifying the implementation of memory management inside the Windows kernel can be carried out in an undergraduate OS curriculum. Undertakings, such as the Abstract Memory Management (AMM) project, integrate well with our previously developed courseware – the Microsoft Windows Internals Curriculum Resource Kit (CRK) [6, 2].

Microsoft made the Windows kernel sources widely available to academia in 2006 [3], replacing the earlier limited distribution that was available only to select universities. Since then, we have expanded our earlier use of Windows in OS courses by developing a number of projects and labs that rely on modifying the Windows kernel. These projects focus on topics such as scheduling/dispatching, synchronization, and memory management. Within this paper, we present the Abstract Memory Management (AMM) experiment which is comprised of a U section, where students practice relevant system APIs (such as Windows API function `VirtualAllocEx()`), an M section, where we ask students to familiarize themselves with measurement techniques and tools (such as the Windows performance monitor – *perfmon.exe*), and a K section where students need to modify source code (e.g.; ntos/mm/wsmanage.c), compile, and run their own version of Windows. During the course, projects are assigned to groups of three students.

In the remainder of the paper, we first present an overview

about the projects we created for the WRK. Then, we present the kernel (K) and the measurement (M) part of the AMM project. (We have omitted the user-mode (U) part due to space limitations.) Instead we present feedback from students who took our course. Finally, we conclude the paper with an outlook on future UMK projects.

2. WRK PROJECTS

The Windows Research Kernel (WRK) is based on the x86/x64 source code of the Windows Server 2003 operating system kernel with Service Pack 1. Figure 1 gives a brief overview of the source code components that are shipped with the WRK. Gray, rectangular boxes denote modules that are available as source code, while the white rectangular boxes denote components that are either provided as static library, like the power management module, or not at all, like device drivers and all user mode components.

Our operating system projects – such as the AMM lab – are structured in such a way that they align well with the Curriculum Resource Kit (CRK) and the IEEE/ACM Computer Science Body of Knowledge (CSBOK) for operating systems. An introductory project helps students to familiarize themselves with the WRK build-process. This project just presents how to modify the kernel sources, rebuild the WRK, deploy the kernel on a test system and show some debugging output. This preliminary project is denoted as P1 in Figure 1. In physics, and other natural sciences, experiments are the fundamental approach to obtaining new insights into the subject material and to cement the knowledge gained from the classroom [1]. Experiments are used to test a theory against the physical world, to gain experimental proof, or to investigate natural phenomena in order to postulate a theory. We believe the former approach to be appropriate for teaching operating systems as well. While the lecture formulates some sort of model, or principle, experiments should be used to test this model against the physical world, in our case the behavior of the computer system.

All of our experiments are structured in the same way: we re-iterate important principles of the respective CRK or CSBOK-OS section, we propose a programming task that must be accomplished in user-mode (U) by facilitating existing programming interfaces (API) and in kernel-mode (K) by extending or modifying operating system source code. We provide a test framework that allows our students to measure (M) benefits and disadvantages of either implementation.

By using this two level approach, we believe we achieve two things: (1) increasing the students' system-programming abilities and (2) improving their understanding of OS design and implementation principles. In the following sections, we concentrate particularly on the kernel mode implementation of the AMM experiment, as we want to report on the challenges we dealt with when running these experiments. We also present the overall goal of the experiment and the respective CRK section of the experiment.

2.1 System Service Call Implementation

This project is designed as an experiment for CRK section OS2: Operating System Principles. System services, encapsulated in application programming interfaces (APIs), are the fundamental abstraction for operating systems to provide functionality to applications. Our experiment addresses the transition from user mode to kernel mode and vice versa. It discusses how parameters are passed from the user space into kernel space on Windows, and how system service calls are dispatched. Therefore, we ask our students to write a simple system call that can alter visibility properties of system resources like processes.

Listing 1: Sample implementation for unlinking a process (`ProcessId`) from the `FromList` and linking it to the `ToList`. Depending on how the function is called, `ToList` may be either the active list of processes or the internal list to keep track of invisible processes. The same holds for the `FromList` parameter.

```
NTSTATUS WrkRemoveAndAppendProcess(
  HANDLE ProcessId, PLIST_ENTRY FromList,
  PLIST_ENTRY ToList)
{
  NTSTATUS Status =
    STATUS_OBJECT_NAME_NOT_FOUND;
  PETHREAD CurrentThread =
    PsGetCurrentThread();
  PLIST_ENTRY LoopEntry;
  PEPROCESS Process;

  PspLockProcessList(CurrentThread);

  for (LoopEntry = FromList->Flink;
      LoopEntry != FromList;
      LoopEntry = LoopEntry->Flink)
  {
    Process = CONTAINING_RECORD(
      LoopEntry, EPROCESS,
      ActiveProcessLinks);

    if (Process->UniqueProcessId ==
        ProcessId)
    {
      Status = STATUS_SUCCESS;
      break;
    }

  }

  if (!NT_SUCCESS(Status))
  {
    PspUnlockProcessList(CurrentThread);
    return Status;
  }

  RemoveEntryList(LoopEntry);
  InsertHeadList(ToList, LoopEntry);
  PspUnlockProcessList(CurrentThread);
  return Status;
}
```

The basic idea in this particular project is to unlink a process's control structure, an `EPROCESS` block, from the OS maintained list of active processes. To restore the process's visibility, it is simply reattached to the process list. Listing 1 shows a sample implementation.

The test application we provide performs a process enumeration on the running system. The user mode implementation simply hides a process from that enumeration and is implemented as a library. Now, when executing the kernel mode implementation, two things should become clear: (1) the implementation difference between normal library calls and system service calls and (2) system service calls usually cost more time than simple function calls. This experiment also forms the basis for several other experiments, as system service calls are the simplest way to extend kernel functionality.

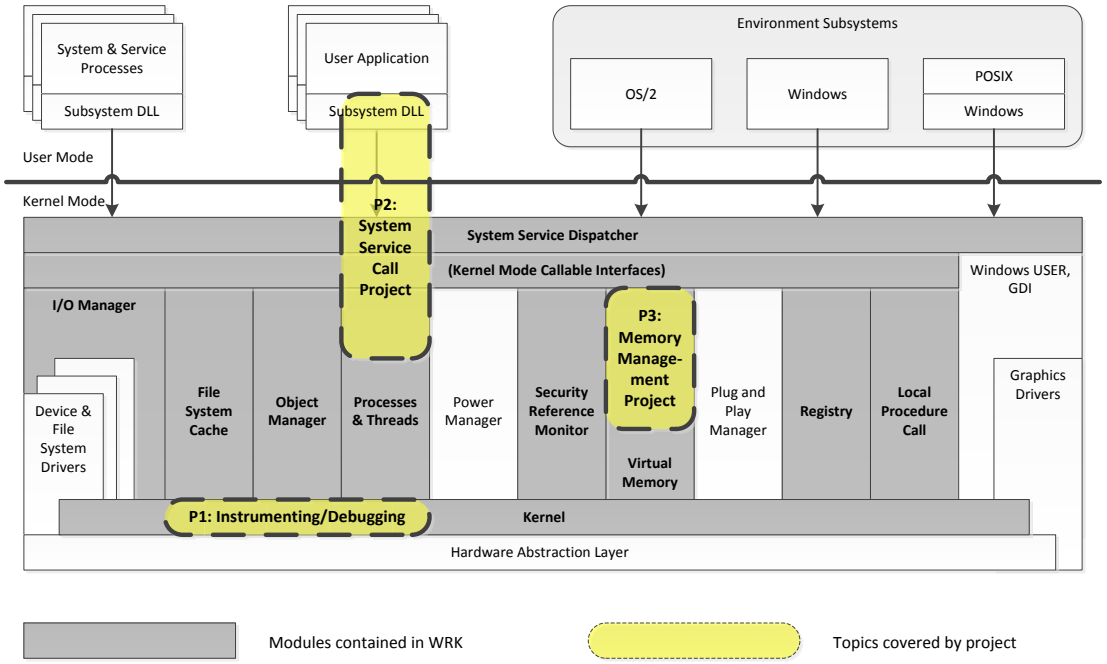

Figure 1: Contents of the WRK. Shaded rectangles denote modules for which sources are available through WRK. White shaded boxes denote modules that are shipped as static libraries. Dashed boxes denote those modules that are covered by the projects presented herein.

2.2 Page Replacement

Besides CPU scheduling, memory management is the most fundamental responsibility of an operating system. Changes within memory management algorithms may have a huge impact on the overall performance of the OS and its applications. Within this experiment we only focus on a small subproblem of memory management: page replacement strategies for working sets.

A process' working set is comprised of all pages in memory that are accessible to the process without incurring a page fault. Therefore we start with designing a suitable data structure to keep track of what pages are currently mapped into physical memory for a particular process – the *working set*. We then proceed to problems associated with that: given a fixed working set size, *i.e.*, the number of pages that can be mapped into physical memory remains constant, replacing pages becomes necessary once the working set is full.

There are several well-known approaches to dealing with that issue, like Second Chance or Aging [8, 9, 10]. However, each page replacement strategy has its advantages and disadvantages, which will be highlighted by our test framework. Also, implementing the working set management as well as a page replacement strategy, in our opinion, helps students to reinforce their understanding and knowledge of virtual memory, working sets, efficient data structures, and finally how all of those may affect the operating system performance.

To accomplish those goals, we had to overcome several challenges encountered during our preparation phase:

- To fully understand the memory management system in general and the page replacement implementation of the WRK in particular is, although inspiring, a difficult and time-consuming task for undergraduate students.

- Making mistakes is just human nature. Doing so in memory management source code, however, complicates matters, as debugging might not be an option. And even with the chance of debugging, finding the cause for an error is still a sophisticated task given the complexity of the virtual memory management system in Windows.

- It may be beneficial to preserve the original implementation, *i.e.*, allowing the students' implementation and the original WRK implementation to co-exist at the same time for the purposes of side-by-side comparison.

Because of these issues, we decided to build an abstraction layer on top of the WRK virtual memory management system, which we henceforth denote as *abstract memory management* (AMM) system and which is basically a framework that provides a simple interface to the WRK Memory Manager. This interface allows students to remove pages from physical memory, to query usage statistics (*e.g.*, whether the page has recently been accessed or modified), and to transparently enable abstract memory management only for specific processes and specific address ranges. In order to modify the memory management system, students have to implement a set of simple function callbacks that are invoked

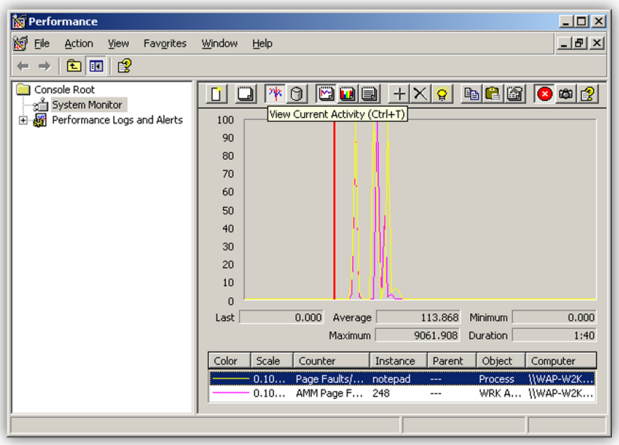

Figure 2: A screenshot showing the WRK page fault performance counter and the AMM page fault performance counter for comparison.

by the AMM system as appropriate, *e.g.*, when a page fault occurs.

The framework is provided as a Visual Studio solution file, mainly for convenience. We decided to use this integrated development environment (IDE) as it allows us to point out those files that need to be modified. IDEs typically provide syntax highlighting, and usually provide an auto-completion feature that assists in handling function names and structure definitions. Additionally, we provide two measurement programs that must be executed on the WRK to test the students' AMM implementation. The results of these measurements are used for evaluating the experiment.

In order to master the AMM lab, students are asked to implement two different page replacement strategies. While our students are free to choose an algorithm, we suggest implementing the FIFO algorithm and the Second Chance algorithm. The measurement part of the experiment then consists of executing test applications, once for each page replacement strategy.

The first test application aims at understanding the implemented approach. We give our students a sequence of page accesses and a number of working set entries, for which they have to first manually calculate when which page resides in what working set entry. For the sake of brevity, we assume here that the working set entry position reflects the position in memory. Having calculated the working set for each discrete access time, our students need to verify that their implementation adheres to the scheme and if not they must find the reason for the discrepancy.

The second test program aims at direct competition between the original page replacement strategy and the AMM implementation. The test program therefore adds some Windows performance counters that retrieve the page fault rate of both the WRK memory management and the AMM. These values are displayed using the Windows performance monitor (perfmon.exe, see Figure 2) to analyze the page fault rate over time. Again, students have to discuss why one implementation performs differently than the other. This experiment is executed under supervision of a tutor and discussed with the lab supervisor right afterwards.

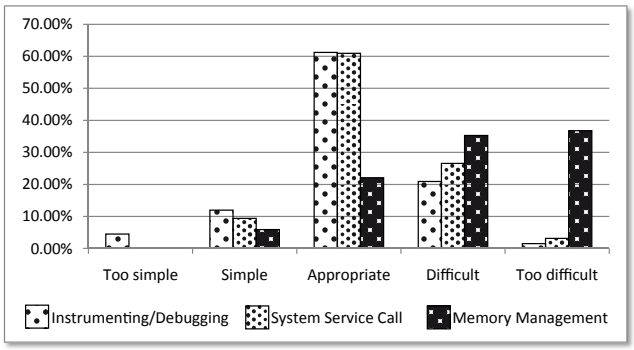

Figure 3: Level of difficulty as perceived by our students.

3. EVALUATION

After the semester, we have conducted a survey on our students to get some feedback on the exercises. In total 69 students participated in the survey. We have asked for feedback in two categories. First, we asked questions about the course in general, in the second part, we wanted to get feedback about the particular exercises that dealt with the Windows Research Kernel.

3.1 General Course Feedback

Within this set of questions, we wanted to know, how the students liked the idea of using the WRK (instead of a purely instructional OS [5, 6]), whether it was easier to do exercises by providing a fully functional IDE, and whether the exercises were understandable. 79% of our students liked using the WRK, and more satisfyingly, 80% considered the WRK as a helpful resource in comprehending OS principles. The survey results also demonstrated that students are well aware that there is no black and white when it comes to operating systems: 56% were interested in also using different operating systems, *e.g.*, MacOS, or UNIX (which they know we cover in our advanced OS course). When it comes to the WRK, providing a fully functional IDE helps lowering the bar for student OS experiments: 67% of our students could work more easily with the WRK when being equipped with a Visual Studio-like build environment. Please note that the WRK is shipped just with a makefile build environment that works perfectly well, if you are used to command line enviroments.

3.2 Exercise Related Feedback

In the second part of our feedback questionnaire we wanted to know more about specific projects. The graphs shown here aggregate the answers specific to our three WRK projects: The simple and introductory debugging task, the system service call task, and the memory management task that was introduced in the previous section.

The first thing we wanted to know was how difficult it was, from our students perspective, to solve a particular task. Figure 3 shows the summary. Given the normal distribution of skills in the audience of a course, *i.e.*, few students have exceptional skills, and few students have skills below average, the Figure shows that our first two experiments were appropriate for the majority of the course. That is,

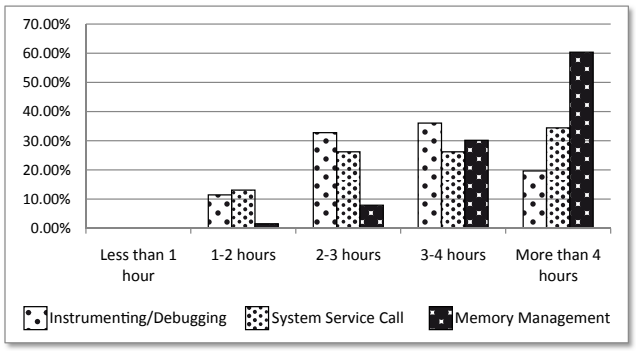

Figure 4: Average time spent to accomplish particular exercise.

the perceived difficulty level reflects the Gaussian distribution of student abilities.

However, the memory management project was different. More than 60% indicated that the exercise was difficult or, even worse, too difficult.

Figure 4 reflects those observations on a timely basis. We asked our students about how long they spend for solving a particular task. As we give out exercises on a bi-weekly basis, we considered it appropriate for students to spend up to 4 hours on these assignments. Again, for the first two tasks, the majority of the course was well in the estimated time range. The memory management project, however, being the most challenging one, took most of our students more than 4 hours of time.

Although being the toughest of all projects, informal question and answer sessions showed us that our students received the most benefit out from this project as they had to relate theoretical knowledge with implementation details as well as coding skills. Mastering the exercise was therefore a big success and huge gain in their abilities as software engineers. However, we want to address the issue by refining our documentation and also the provided framework in order to reduce the fraction of students who may not have a positive experience with the project.

4. CONCLUSION

When studying operating systems, students need to understand user-mode system interfaces (U), they need to learn about tools to monitor and measure OS behavior (M), and they finally should understand central implementation details of the OS kernel (K). Prior to the Windows Research Kernel (WRK), OS experiments in the Windows domain were limited to the system programming level [4].

Using the Abstract Memory Management (AMM) project as an example, we have briefly discussed our approach to teaching operating system concepts based on Windows and the WRK in particular. The authors have used the WRK in undergraduate courses taught at Hasso Plattner Institute, University of Potsdam in Germany, University of Washington Bothell, and Blekinge Institute of Technology in Ronneby, Sweden. Both WRK and the accompanying Curriculum Resource Kit (CRK) are available for faculty to download through Microsoft's Faculty Connection website or by contacting Microsoft at Compsci@Microsoft.com.

The AMM project is part of a forthcoming book on experimenting with the Windows kernel. Additional experiments will focus on topics such as concurrency and synchronization, the Windows object manager, thread scheduling and dispatching, and the input/output-system. The authors maintain a WRK-related blog that provides help and additional information for using the Windows kernel as a teaching vehicle [7].

Operating systems are exciting, complex software systems. However, today most users do not even distinguish the computer and its operating system. Students often do not see the need to understand the internals of the OS they use. With our OS projects, based on the operating system most students are likely to use upon graduation, we are attempting to re-spark interest in operating systems again. Student feedback has been very promising, but still much pedagogical work remains.

Acknowledgements

We would like to thank Bernhard Rabe and Michael Schöbel for their valuable feedback and fruitful discussions.

5. REFERENCES

[1] A. Franklin. Experiment in physics. In *Stanford Encyclopedia of Philosophy*. Stanford University, January 2009.

[2] Microsoft. Windows Operating System Internals Curriculum Resource Kit. http://www.microsoft.com/resources/sharedsource/windowsacademic/curriculumresourcekit.mspx.

[3] Microsoft. Windows Research Kernel. http://www.microsoft.com/resources/sharedsource/Licensing/researchkernel.mspx, 2006.

[4] G. J. Nutt. *Operating System Projects Using Windows NT*. Addison Wesley Publishing Company, January 1999.

[5] B. Pfaff, A. Romano, and G. Back. The pintos instructional operating system kernel. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 453–457, New York, NY, USA, 2009. ACM.

[6] A. Polze and D. Probert. Teaching operating systems: the Windows case. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 298–302, New York, NY, USA, 2006. ACM Press.

[7] M. Schöbel and A. Schmidt. Windows research kernel @ hpi. http://www.dcl.hpi.uni-potsdam.de/research/WRK, May 2007.

[8] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley, 8th edition, July 2008.

[9] W. Stallings. *Operating Systems: Internals and Design Principles*. Prentice Hall, 6th edition, April 2008.

[10] A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 3rd edition, 2007.