Übung II
Windows Research Kernel

Michael Schöbel

Betriebssystemarchitektur I – WS 2007

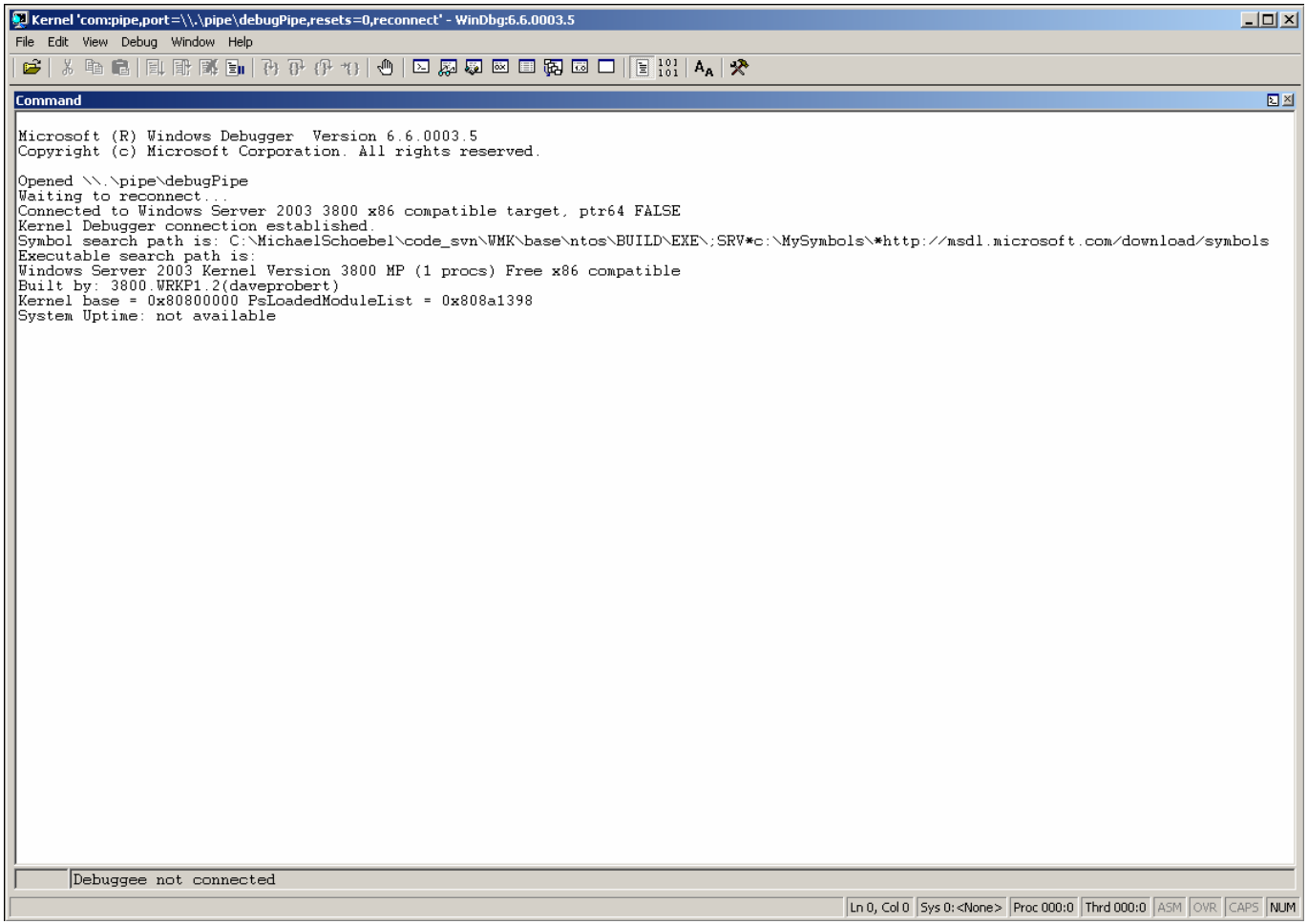
15. November 2007

Agenda

- **Debugging mit Windbg**
- **Windowskern Instrumentierung**
- **Systemaufrufe**
- **(MY)NTDLL.DLL**
- **Hinweise zum Übungsblatt**

Debugging mit Windbg
Setup

- **Konfiguration → siehe Übung vom 01.11.**



Debugging mit Windbg
Symbol-Dateien (.PDB)

- **PDB = Program Data Base**
  - Informationen für Debugger
    - Funktionsnamen
    - Zeilennummern
    - …
  - Tool: **`dia2dump`**
    - **`Microsoft Visual Studio .NET 2003\Visual Studio SDKs\DIA SDK\Sample\`**
    - **`Microsoft Visual Studio 8\DIA SDK\Samples\`**
  - Symbolserver von Microsoft
    - Enthält PDB-Dateien für Windows-Programme und DLLs
    - Symbol Pfad in Windbg konfigurieren
      `srv*x:\localsymbols*http://msdl.microsoft.com/download/symbols`

Debugging mit Windbg
Kommandos

- **Symbole**
  - **`ln <address>`**
    Zeigt Symbol zur gegebenen Adresse
  - **`X <module>!<symbolname>`**
    Suchen/anzeigen von Symbolen
- **Daten**
  - **`dt <symbol> [<address>]`**
    Anzeigen von Informationen über Datentyp
  - **`dd <address> [L<length>]`**
    Anzeigen des Speicherinhalts
  - **`uf <symbol>`**
    Dissassemblieren einer Funktion

Windowskern Instrumentierung Definition

- **Instrumentierung**
  - Hilfsmittel zur (Laufzeit-)Analyse von Programmen
  - Nachverfolgung von Programmschritten
  - Zeitmessung, Zählen von Ereignissen, … → Profiling
  - Einfügen von Instrumentierungscode
  - Beeinflussung der Programmabarbeitung
  - Gegensatz: „sampling“ des Programmzustandes

Windowskern Instrumentierung DbgPrint

- **Verwendung der Debug-Konsole**
  - Ausgabe von Informationen mit **`DbgPrint`**
  - http://msdn2.microsoft.com/en-us/library/ms792790.aspx
  - Formatierung analog zu **`printf`**
  - Beispiel:
    - **`DbgPrint("%p\n", Thread->StartAddress);`**

Windowskern Instrumentierung Übungsaufgabe

- **Prozess- und Threaderzeugung**
  - Prozess-/Thread-ID, Funktionsadressen
  - Instrumentierung von vier Kernel-Funktionen

- **Fragen**
  - Welche Prozesse werden zuerst erzeugt?
  - Welche Funktionen führen die ersten fünf gestarteten Threads aus?
  - Welcher Treiber erzeugt fortlaufend Threads?

Systemaufrufe
Trennung von User- und Kernel-Mode

- **Kernel-Mode**
  - Ausführung privilegierter/kritischer Operationen
- **User-Mode**
  - Getrennte Adressräume, nicht privilegierte Operationen
- **Systemaufruf → Wechsel vom User- zum Kernel-Mode**
  - Kontrollierter Funktionsaufruf
  - Kontrollierte Erweiterung der Ausführungs-Privilegien
  - Menge der Systemaufrufe = Schnittstelle zum Kernel

Systemaufrufe
Aufruf von Kernel-Funktionen

- **Prinzipieller Ablauf**
  - Systemaufruf Tabelle → Adressierung über Systemaufrufnummer
  - Auslösung eines (Software-)Interrupts (*trap*)
  - Wechsel in Kernel-Mode
  - Kopieren der Parameter
  - Sprung zur Systemaufruf-Implementierung (`Nt*`-Funktion)
  - Testen der Parameter
  - Ausführen der Kernelfunktion
  - Rücksprung in den Usermode

Systemaufrufe
Implementierung

- **Parameter**
  - Systemaufrufnummer → EAX Register
  - Systemaufrufparameter → EDX Register

- **Aufruf**
  - Implementierung durch Interrupt (Window **`2Eh`**, Linux **`80h`**)
  - Verwendung von CPU spezifischer Operation (**`syscall`**, **`sysenter`**)
  - Vorgabe durch Betriebssystem

(MY)NTDLL.DLL
Direkter Aufruf des Windowskerns (I)

- **Windows Systemaufrufe**
  - Undokumentierte Kernel-Schnittstelle
    http://www.metasploit.com/users/opcode/syscalls.html
  - Notwendige Informationen
    - Systemaufrufnummer
    - Erwartete Parameter
  - WRK Quelltext → nach Implementierung suchen (`Nt*`-Funktion)
  - Schnittstellenparameter sind Bestandteil des SDK

(MY)NTDLL.DLL
Direkter Aufruf des Windowskerns (II)

- Beispiel:
  - Systemaufrufnummer
    - XP SP2: 0x00AE
    - EE SP1: 0x00B6
  - Funktion:

```
NtQuerySystemTime(
         OUT PLARGE_INTEGER CurrentTime
);
```

```
MYDLL_API QuerySystemTime_2E(
         OUT PLARGE_INTEGER CurrentTime) {

  __asm {
      mov eax, 0x00AE                // XP
//    mov eax, 0x00B6                // EE

      lea edx, dword ptr [esp+4]
      int 0x2E

      ret
  }
}
```

(MY)NTDLL.DLL
Übungsaufgabe

- **Implementierung der Funktion `SetThreadPriorityExplicit`**
  - Direktes setzen der Thread Priorität auf einen bestimmten Wert
  - Programmrahmen auf der Vorlesungswebseite
- **Nachweis mit Hilfe des Performance Monitors (`perfmon`)**
  - Thread → Aktuelle Priorität darstellen
- **Kein(!) neuer Systemaufruf; vorhandene Schnittstelle**
  - → **`NtSetInformationThread`**
  - Übungsblatt enthält die notwendigen Schritte

Hinweise zum Übungsblatt

- Windows Research Kernel für Aufgabe 2.4 erforderlich,
  Aufgabe 2.5 kann auch mit „normalem“ Windows bearbeitet werden
  (benötigt allerdings Windows SDK)

- VMWare Image ist auf der Vorlesungswebseite verfügbar
  https://pao.dcl.hpi.uni-potsdam.de/

  Windows Server 2003 EE Key erforderlich → Maniac

- Windows Research Kernel spezifische Fragen bitte an
  wrk@hpi.uni-potsdam.de senden

Ausblick

- **Erweiterung vorhandener Systemaufrufe**
- **Implementierung neuer Systemaufrufe**