Kernel- und UserSpace Debugging Techniken

Hagen Paul Pfeifer

„Debuggers don't remove Bugs, they only show them in Slow-Motion."

http://jauu.0xdef.net
hagen@jauu.net

19. November 2004

Fahrplan

1. Einführung
   - Grundlegende Debugging Überlegungen
   - kleines Handwerkzeugs
2. Userspace Debugging
   - cpp
   - gcc
   - gdb
   - ddd
3. Kernelspace Debugging
   - gdb und ddd
   - kgpd kernel patch
   - uml
4. Verschiedenes
   - x86 Hardware-Implementierung
   - Linux-Implementierung

Fehlersuche - Der Käfer im Heuhaufen

- Fehler eingrenzen (Nebenläufigkeit, Komplexität eliminieren (wenn möglich))
- Teilbereiche inspizieren
- gewöhnlich ein iterativer Prozeß

Fehlersuche - Der Käfer im Heuhaufen

- Fehler eingrenzen (Nebenläufigkeit, Komplexität eliminieren (wenn möglich))
- Teilbereiche inspizieren
- gewöhnlich ein iterativer Prozeß

Fehlersuche - Der Käfer im Heuhaufen

- Fehler eingrenzen (Nebenläufigkeit, Komplexität eliminieren (wenn möglich))
- Teilbereiche inspizieren
- gewöhnlich ein iterativer Prozeß

Virtuelle Werkzeugkasten

- assert(3)
  - Nützlich bei definierten Wertebereich (Vordedingungen)
  - `void assert(scalar expression);`
  - ruft abort(3) wenn `expression` falsch ist
  - als Markro definiert
  - NULL-Makro wenn NDEBUG definiert (`/usr/include/assert.h`)
- Preprocessor Direktiven a'la

```
# ifdef DEBUG
fprintf(stderr, "n: %d", x)
# endif
```

Virtuelle Werkzeugkasten

- assert(3)
  - Nützlich bei definierten Wertebereich (Vordedingungen)
  - `void assert(scalar expression);`
  - ruft abort(3) wenn `expression` falsch ist
  - als Markro definiert
  - NULL-Makro wenn NDEBUG definiert (`/usr/include/assert.h`)
- Preprocessor Direktiven a'la

```
# ifdef DEBUG
fprintf(stderr, "n: %d", x)
# endif
```

Virtuelle Werkzeugkasten

- assert(3)
  - Nützlich bei definierten Wertebereich (Vordedingungen)
  - `void assert(scalar expression);`
  - ruft abort(3) wenn `expression` falsch ist
  - als Markro definiert
  - NULL-Makro wenn NDEBUG definiert (`/usr/include/assert.h`)
- Preprocessor Direktiven a'la

```
# ifdef DEBUG
fprintf(stderr, "n: %d", x)
# endif
```

Virtuelle Werkzeugkasten

- strace(1)
  - Zeigt Systemaufrufe der getracten Applikation
  - Vorteilhaft bei Programmen welche ohne Debugging Symbole übersetzt wurden
  - Liefert Grundlagen für ein gesundes Programmverständniss
  - `strace -e trace=open,close,read,write` beschränkt Ausgabe

Virtuelle Werkzeugkasten

- strace(1)
  - Zeigt Systemaufrufe der getracten Applikation
  - Vorteilhaft bei Programmen welche ohne Debugging Symbole übersetzt wurden
  - Liefert Grundlagen für ein gesundes Programmverständniss
  - `strace -e trace=open,close,read,write` beschränkt Ausgabe

Virtuelle Werkzeugkasten

- ldd - zeigt Bibliothekenabhängigkeiten
- cat /proc/pid/*
  - liefert Speicheradressen aller gemappten Pages
  - argumente, umgebungsvariablen
  - file descriptoren
  - status (statm, status)
- readelf, objdump und nm (binutils)
  - offenbart das Laufzeitverhalten auf abstrakter Ebene
  - liefert Symbole, Segmente, Sectionen, Debuginfo, asm-output, ...

Virtuelle Werkzeugkasten

- ldd - zeigt Bibliothekenabhängigkeiten
- cat /proc/pid/*
  - liefert Speicheradressen aller gemappten Pages
  - argumente, umgebungsvariablen
  - file descriptoren
  - status (statm, status)
- readelf, objdump und nm (binutils)
  - offenbart das Laufzeitverhalten auf abstrakter Ebene
  - liefert Symbole, Segmente, Sectionen, Debuginfo, asm-output, ...

Virtuelle Werkzeugkasten

- ldd - zeigt Bibliothekenabhängigkeiten
- cat /proc/pid/*
  - liefert Speicheradressen aller gemappten Pages
  - argumente, umgebungsvariablen
  - file descriptoren
  - status (statm, status)
- readelf, objdump und nm (binutils)
  - offenbart das Laufzeitverhalten auf abstrakter Ebene
  - liefert Symbole, Segmente, Sectionen, Debuginfo, asm-output, ...

GNU - Preprocessor

- `gcc -E`
- Hilfreich bei Fehlern welcher durch Preprozessor verursacht wurden
  - Flags verivizieren (z.B. GNU autotools makros)
  - Makros kontrollieren
  - Sollwert inspizieren

GNU - Preprocessor

- `gcc -E`
- Hilfreich bei Fehlern welcher durch Preprozessor verursacht wurden
  - Flags verivizieren (z.B. GNU autotools makros)
  - Makros kontrollieren
  - Sollwert inspizieren

GNU - Compiler Collection

- `gcc -Wall`
- Warnt bei fragwürdigen Konstrukten
  - `-Wunused`
  - `-Wswitch-default`
  - `-Wreturn-type`
  - ...
- `-W` bei Entwicklungsphase von Vorteil
  - zusätzliche Warnungen bei zweifelhaften Konstrukten
  - teilweise schwer zu „maskieren“
- `-Werror` konvertiert Warnugen in Fehler
- Großeltern verwendeten lint, Funktionalität erfüllt heute `-Wall`

GNU - Compiler Collection

- `gcc -Wall`
- Warnt bei fragwürdigen Konstrukten
  - `-Wunused`
  - `-Wswitch-default`
  - `-Wreturn-type`
  - `...`
- `-W` bei Entwicklungsphase von Vorteil
  - zusätzliche Warnungen bei zweifelhaften Konstrukten
  - teilweise schwer zu „maskieren“
- `-Werror` konvertiert Warnugen in Fehler
- Großeltern verwendeten lint, Funktionalität erfüllt heute `-Wall`

GNU - Compiler Collection

- `gcc -Wall`
- Warnt bei fragwürdigen Konstrukten
  - `-Wunused`
  - `-Wswitch-default`
  - `-Wreturn-type`
  - `...`
- `-W` bei Entwicklungsphase von Vorteil
  - zusätzliche Warnungen bei zweifelhaften Konstrukten
  - teilweise schwer zu „maskieren“
- `-Werror` konvertiert Warnugen in Fehler
- Großeltern verwendeten lint, Funktionalität erfüllt heute `-Wall`

GNU - Compiler Collection

- `gcc -Wall`
- Warnt bei fragwürdigen Konstrukten
  - `-Wunused`
  - `-Wswitch-default`
  - `-Wreturn-type`
  - `...`
- `-W` bei Entwicklungsphase von Vorteil
  - zusätzliche Warnungen bei zweifelhaften Konstrukten
  - teilweise schwer zu „maskieren“
- `-Werror` konvertiert Warnugen in Fehler
- Großeltern verwendeten lint, Funktionalität erfüllt heute `-Wall`

GNU - Compiler Collection

- `gcc -Wall`
- Warnt bei fragwürdigen Konstrukten
  - `-Wunused`
  - `-Wswitch-default`
  - `-Wreturn-type`
  - `...`
- `-W` bei Entwicklungsphase von Vorteil
  - zusätzliche Warnungen bei zweifelhaften Konstrukten
  - teilweise schwer zu „maskieren“
- `-Werror` konvertiert Warnugen in Fehler
- Großeltern verwendeten lint, Funktionalität erfüllt heute `-Wall`

gdb - GNU Debugger

- GNU Projekt Debugger
- Entwicklung: arch, bugfixes, flat rewrites (Changelog)
- Richard Stallmann Author, Cygnus, jetzt viel Unterstützung von RedHat

gdb - GNU Debugger

- GNU Projekt Debugger
- Entwicklung: arch, bugfixes, flat rewrites (Changelog)
- Richard Stallmann Author, Cygnus, jetzt viel Unterstützung von RedHat

gdb - GNU Debugger

- GNU Projekt Debugger
- Entwicklung: arch, bugfixes, flat rewrites (Changelog)
- Richard Stallmann Author, Cygnus, jetzt viel Unterstützung von RedHat

gdb - GNU Debugger (Debugginformationen)

- -g generiert Ausgabe mit Debugging Informationen
- -g und -OX moeglich, aber nicht empfehlenswert
- -gX Debug level:
  1. minimale Debug Informationen (backtrace ja, lokale Variablen nein, ...)
  2. default level
  3. Preprozessor Direktiven
- eventuell gdwarf-2 zu -g3 erforderlich
- viel Debuginformationen: Verdoppelung bis Vervierfachung der Dateigrösse

gdb - GNU Debugger (Debugginformationen)

- -g generiert Ausgabe mit Debugging Informationen
- -g und -OX moeglich, aber nicht empfehlenswert
- -gX Debug level:
  1. minimale Debug Informationen (backtrace ja, lokale Variablen nein, ...)
  2. default level
  3. Preprozessor Direktiven
- eventuell gdwarf-2 zu -g3 erforderlich
- viel Debuginformationen: Verdoppelung bis Vervierfachung der Dateigrösse

gdb - GNU Debugger (Debugginformationen)

- -g generiert Ausgabe mit Debugging Informationen
- -g und -OX moeglich, aber nicht empfehlenswert
- -gX Debug level:
  1. minimale Debug Informationen (backtrace ja, lokale Variablen nein, ...)
  2. default level
  3. Preprozessor Direktiven
- eventuell gdwarf-2 zu -g3 erforderlich
- viel Debuginformationen: Verdoppelung bis Vervierfachung der Dateigrösse

gdb - GNU Debugger (Debugginformationen)

- `-g` generiert Ausgabe mit Debugging Informationen
- `-g` und `-OX` moeglich, aber nicht empfehlenswert
- `-gX` Debug level:
  1. minimale Debug Informationen (backtrace ja, lokale Variablen nein, ...)
  2. default level
  3. Preprozessor Direktiven
- eventuell `gdwarf-2` zu `-g3` erforderlich
- viel Debuginformationen: Verdoppelung bis Vervierfachung der Dateigrösse

gdb - GNU Debugger (Debugginformationen)

- `-g` generiert Ausgabe mit Debugging Informationen
- `-g` und `-OX` moeglich, aber nicht empfehlenswert
- `-gX` Debug level:
  1. minimale Debug Informationen (backtrace ja, lokale Variablen nein, ...)
  2. default level
  3. Preprozessor Direktiven
- eventuell `gdwarf-2` zu `-g3` erforderlich
- viel Debuginformationen: Verdoppelung bis Vervierfachung der Dateigrösse

gdb - Prologue

- `gdb -help` und `info` nimmt dich an die Hand
- `gdb programm` der übliche und üble Fall
- `gdb programm corefile` post-mortem Analyse
- `gdb program pid` jit debugging
- `gdb --command=FILE program` eventuell hilfreich

gdb - Prologue

- `gdb -help` und `info` nimmt dich an die Hand
- `gdb programm` der übliche und üble Fall
- `gdb programm corefile` post-mortem Analyse
- `gdb program pid` jit debugging
- `gdb --command=FILE program` eventuell hilfreich

gdb - Prologue

- `gdb -help` und `info` nimmt dich an die Hand
- `gdb programm` der übliche und üble Fall
- `gdb programm corefile` post-mortem Analyse
- `gdb program pid` jit debugging
- `gdb --command=FILE program` eventuell hilfreich

gdb - Prologue

- `gdb -help` und `info` nimmt dich an die Hand
- `gdb programm` der übliche und üble Fall
- `gdb programm corefile` post-mortem Analyse
- `gdb program pid` jit debugging
- `gdb --command=FILE program` eventuell hilfreich

gdb - Prologue

- `gdb -help` und `info` nimmt dich an die Hand
- `gdb programm` der übliche und üble Fall
- `gdb programm corefile` post-mortem Analyse
- `gdb program pid` jit debugging
- `gdb --command=FILE program` eventuell hilfreich

gdb - Bedienung (Args und Envs)

- `--args` als argument
- `set args` interactive
- `set env` setzt Umgebunsvariablen
- show args und show envd

gdb - Bedienung (Args und Envs)

- `--args` als argument
- `set args` interactive
- `set env` setzt Umgebunsvariablen
- show args und show envd

gdb - Bedienung (Args und Envs)

- `--args` als argument
- `set args` interactive
- `set env` setzt Umgebunsvariablen
- show args und show envd

gdb - Bedienung (Args und Envs)

- `--args` als argument
- `set args` interactive
- `set env` setzt Umgebunsvariablen
- show args und show envd

gdb - Programmlauf

- `run (run > output)`
- `attach id; detach`
- `continue` führt Programm weiter aus

gdb - Programmlauf

- `run` (`run > output`)
- `attach id`; `detach`
- `continue` führt Programm weiter aus

gdb - Programmlauf

- `run` (`run > output`)
- `attach id`; `detach`
- `continue` führt Programm weiter aus

gdb - Breakpoints

- Unterbricht Programmfluss an Stelle
- `break` ist das Kommando der Wahl
- Funktionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`
- `info breakpoints` liefert alle Breakpoints mit Info
- `delete n` löscht Breakpoint `n`
- `disable n` und `enable n` temporär

gdb - Breakpoints

- Unterbricht Programmfluss an Stelle
- `break` ist das Kommando der Wahl
- Funktionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`
- `info breakpoints` liefert alle Breakpoints mit Info
- `delete n` löscht Breakpoint n
- `disable n` und `enable n` temporär

gdb - Breakpoints

- Unterbricht Programmfluss an Stelle
- `break` ist das Kommando der Wahl
- Funktionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`
- `info breakpoints` liefert alle Breakpoints mit Info
- `delete n` löscht Breakpoint n
- `disable n` und `enable n` temporär

gdb - Breakpoints

- Unterbricht Programmfluss an Stelle
- `break` ist das Kommando der Wahl
- Funktionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`
- `info breakpoints` liefert alle Breakpoints mit Info
- `delete n` löscht Breakpoint n
- `disable n` und `enable n` temporär

gdb - Breakpoints

- Unterbricht Programmfluss an Stelle
- `break` ist das Kommando der Wahl
- Funktionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`
- `info breakpoints` liefert alle Breakpoints mit Info
- `delete n` löscht Breakpoint n
- `disable n` und `enable n` temporär

gdb - Breakpoints

- Unterbricht Programmfluss an Stelle
- `break` ist das Kommando der Wahl
- Funktionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`
- `info breakpoints` liefert alle Breakpoints mit Info
- `delete n` löscht Breakpoint n
- `disable n` und `enable n` temporär

gdb - Breakpoints (aufgebohrt)

- Breakpoint mit Abhängikeiten `break if ...`
- `hbreak` setzt Hardwarebreakpoints
- `tbreak` setzt Breakpoint und löscht ihn nach erreichen
- `rbreak` setzt Breakpoints via regulären Ausdruck (z.B. C++)
- Functionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`

gdb - Breakpoints (aufgebohrt)

- Breakpoint mit Abhängikeiten `break if ...`
- `hbreak` setzt Hardwarebreakpoints
- `tbreak` setzt Breakpoint und löscht ihn nach erreichen
- `rbreak` setzt Breakpoints via regulären Ausdruck (z.B. C++)
- Functionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`

gdb - Breakpoints (aufgebohrt)

- Breakpoint mit Abhängikeiten `break if ...`
- `hbreak` setzt Hardwarebreakpoints
- `tbreak` setzt Breakpoint und löscht ihn nach erreichen
- `rbreak` setzt Breakpoints via regulären Ausdruck (z.B. C++)
- Functionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`

gdb - Breakpoints (aufgebohrt)

- Breakpoint mit Abhängikeiten `break if ...`
- `hbreak` setzt Hardwarebreakpoints
- `tbreak` setzt Breakpoint und löscht ihn nach erreichen
- `rbreak` setzt Breakpoints via regulären Ausdruck (z.B. C++)
- Functionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`

gdb - Breakpoints (aufgebohrt)

- Breakpoint mit Abhängikeiten `break if ...`
- `hbreak` setzt Hardwarebreakpoints
- `tbreak` setzt Breakpoint und löscht ihn nach erreichen
- `rbreak` setzt Breakpoints via regulären Ausdruck (z.B. C++)
- Functionsname, Quellcode Zeile oder Adresse sind gute Argumente
  1. `break funktion`
  2. `break zeile`
  3. `break datei:zeile`
  4. `break datei:funktion`
  5. `break *adresse`

gdb - Watchpoints

- Unterbricht Programmfluss falls Wert sich ändert
- `watch expr` unterbricht Ausführung bei schreibenden Zugriff
- `rwatch expr` unterbricht bei lesenden Zugriff (arch)
- `awatch expr` bei lesenden und schreibenden Zugriff
- `delete` und `disable` löscht oder unterbricht Breakpoint

gdb - Watchpoints

- Unterbricht Programmfluss falls Wert sich ändert
- `watch expr` unterbricht Ausführung bei schreibenden Zugriff
- `rwatch expr` unterbricht bei lesenden Zugriff (arch)
- `awatch expr` bei lesenden und schreibenden Zugriff
- `delete` und `disable` löscht oder unterbricht Breakpoint

gdb - Watchpoints

- Unterbricht Programmfluss falls Wert sich ändert
- `watch expr` unterbricht Ausführung bei schreibenden Zugriff
- `rwatch expr` unterbricht bei lesenden Zugriff (arch)
- `awatch expr` bei lesenden und schreibenden Zugriff
- `delete` und `disable` löscht oder unterbricht Breakpoint

gdb - Watchpoints

- Unterbricht Programmfluss falls Wert sich ändert
- `watch expr` unterbricht Ausführung bei schreibenden Zugriff
- `rwatch expr` unterbricht bei lesenden Zugriff (arch)
- `awatch expr` bei lesenden und schreibenden Zugriff
- `delete` und `disable` löscht oder unterbricht Breakpoint

gdb - Watchpoints

- Unterbricht Programmfluss falls Wert sich ändert
- `watch expr` unterbricht Ausführung bei schreibenden Zugriff
- `rwatch expr` unterbricht bei lesenden Zugriff (arch)
- `awatch expr` bei lesenden und schreibenden Zugriff
- `delete` und `disable` löscht oder unterbricht Breakpoint

gdb - Stepping

- `step n` bis n nächsten Quellcodezeile
- `stepi n` n nächste Instruktionen
- `next n` bis n nächsten Quellcodezeile (aktueller Stack Frame)
- `nexti n` nächsten Instruktionen (aktueller Stack Frame)

gdb - Stepping

- `step n` bis n nächsten Quellcodezeile
- `stepi n` n nächste Instruktionen
- `next n` bis n nächsten Quellcodezeile (aktueller Stack Frame)
- `nexti n` nächsten Instruktionen (aktueller Stack Frame)

gdb - Stepping

- `step n` bis n nächsten Quellcodezeile
- `stepi n` n nächste Instruktionen
- `next n` bis n nächsten Quellcodezeile (aktueller Stack Frame)
- `nexti n` nächsten Instruktionen (aktueller Stack Frame)

gdb - Stepping

- `step n` bis n nächsten Quellcodezeile
- `stepi n` n nächste Instruktionen
- `next n` bis n nächsten Quellcodezeile (aktueller Stack Frame)
- `nexti` n nächsten Instruktionen (aktueller Stack Frame)

gdb - Stepping

- `step n` bis n nächsten Quellcodezeile
- `stepi n` n nächste Instruktionen
- `next n` bis n nächsten Quellcodezeile (aktueller Stack Frame)
- `nexti` n nächsten Instruktionen (aktueller Stack Frame)

gdb - Variablen, Speicherbereich

- `print /f n` zeigt Inhalt der Variable n
  - x == hex
  - d == digit
  - c == character
  - f == float
- `x/nfu adresse` zeigt Speicherbereich
- `display` kontinuierlicher print
- `bt` zeigt Aufrufstack
- `info locals` zeigt alle lokalen Variablen im aktuellen Frame
- `info variables` listet moduleglobale und programglobale Variablen

gdb - Variablen, Speicherbereich

- `print /f n` zeigt Inhalt der Variable n
  - x == hex
  - d == digit
  - c == character
  - f == float
- `x/nfu adresse` zeigt Speicherbereich
- `display` kontinuierlicher print
- `bt` zeigt Aufrufstack
- `info locals` zeigt alle lokalen Variablen im aktuellen Frame
- `info variables` listet moduleglobale und programglobale Variablen

gdb - Variablen, Speicherbereich

- `print /f n` zeigt Inhalt der Variable n
  - x == hex
  - d == digit
  - c == character
  - f == float
- `x/nfu adresse` zeigt Speicherbereich
- `display` kontinuierlicher print
- `bt` zeigt Aufrufstack
- `info locals` zeigt alle lokalen Variablen im aktuellen Frame
- `info variables` listet moduleglobale und programglobale Variablen

gdb - Variablen, Speicherbereich

- `print /f n` zeigt Inhalt der Variable n
  - x == hex
  - d == digit
  - c == character
  - f == float
- `x/nfu adresse` zeigt Speicherbereich
- `display` kontinuierlicher print
- `bt` zeigt Aufrufstack
- `info locals` zeigt alle lokalen Variablen im aktuellen Frame
- `info variables` listet moduleglobale und programglobale Variablen

gdb - Variablen, Speicherbereich

- `print /f n` zeigt Inhalt der Variable n
  - x == hex
  - d == digit
  - c == character
  - f == float
- `x/nfu adresse` zeigt Speicherbereich
- `display` kontinuierlicher print
- `bt` zeigt Aufrufstack
- `info locals` zeigt alle lokalen Variablen im aktuellen Frame
- `info variables` listet moduleglobale und programglobale Variablen

gdb - Variablen, Speicherbereich

- `print /f n` zeigt Inhalt der Variable n
  - x == hex
  - d == digit
  - c == character
  - f == float
- `x/nfu adresse` zeigt Speicherbereich
- `display` kontinuierlicher print
- `bt` zeigt Aufrufstack
- `info locals` zeigt alle lokalen Variablen im aktuellen Frame
- `info variables` listet moduleglobale und programglobale Variablen

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - Stützen

- `set x = 4` ändert Varaible x auf 4
- `commands n ...end` führt Kommandos bei Breakpoint aus
- gdb Variablen: `$pc`, `$sp`
- `list` zeigt Quellcode
- `disassemble` zeigt Maschinenbefehle für bestimmten Bereich
- `info registers` zeigt Register
- `info macro n` zeigt Macrodefinition an

gdb - fork(2) und Thread's

- `set follow-fork-mode` (child—parent—ask)
- breakpoint in child wirft SIGTRAP
- `info threads` listet laufende Threads auf
- `thread n` wechselt zu Thread n

gdb - fork(2) und Thread's

- `set follow-fork-mode` (child—parent—ask)
- breakpoint in child wirft SIGTRAP
- `info threads` listet laufende Threads auf
- `thread n` wechselt zu Thread n

gdb - fork(2) und Thread's

- `set follow-fork-mode` (child—parent—ask)
- breakpoint in child wirft SIGTRAP
- `info threads` listet laufende Threads auf
- `thread n` wechselt zu Thread n

gdb - fork(2) und Thread's

- `set follow-fork-mode` (child—parent—ask)
- breakpoint in child wirft SIGTRAP
- `info threads` listet laufende Threads auf
- `thread n` wechselt zu Thread n

ddd - data display debugger

- Front-End für gdb (und andere)
- Andreas Zeller and Dorothea Luetkehaus sind Autoren
- schöne grafische Visualisierung
- `ddd programmname` (da Front-End: nahezu identische Bedienung)

ddd - data display debugger

- Front-End für gdb (und andere)
- Andreas Zeller and Dorothea Luetkehaus sind Autoren
- schöne grafische Visualisierung
- `ddd programmname` (da Front-End: nahezu identische Bedienung)

ddd - data display debugger

- Front-End für gdb (und andere)
- Andreas Zeller and Dorothea Luetkehaus sind Autoren
- schöne grafische Visualisierung
- `ddd programmname` (da Front-End: nahezu identische Bedienung)

ddd - data display debugger

- Front-End für gdb (und andere)
- Andreas Zeller and Dorothea Luetkehaus sind Autoren
- schöne grafische Visualisierung
- `ddd programmname` (da Front-End: nahezu identische Bedienung)

Grundsätzliches

- lxr (linux cross reference)
- Magic SysReq (/usr/src/linux/Documentation/sysrq.txt)
- `printk(format, ...);` (/var/log/*; dmesg)
- ksymoops(8) um kernel oops zu analysieren
- `lkcd` für spezialisierte Ausaben (linux kernel crash dump)

Grundsätzliches

- lxr (linux cross reference)
- Magic SysReq (/usr/src/linux/Documentation/sysrq.txt)
- `printk(format, ...);` (/var/log/*; dmesg)
- ksymoops(8) um kernel oops zu analysieren
- `lkcd` für spezialisierte Ausaben (linux kernel crash dump)

Grundsätzliches

- lxr (linux cross reference)
- Magic SysReq (/usr/src/linux/Documentation/sysrq.txt)
- `printk(format, ...);` (/var/log/*; dmesg)
- ksymoops(8) um kernel oops zu analysieren
- `lkcd` für spezialisierte Ausaben (linux kernel crash dump)

Grundsätzliches

- lxr (linux cross reference)
- Magic SysReq (/usr/src/linux/Documentation/sysrq.txt)
- `printk(format, ...);` (/var/log/*; dmesg)
- ksymoops(8) um kernel oops zu analysieren
- `lkcd` für spezialisierte Ausaben (linux kernel crash dump)

Grundsätzliches

- lxr (linux cross reference)
- Magic SysReq (/usr/src/linux/Documentation/sysrq.txt)
- `printk(format, ...);` (/var/log/*; dmesg)
- ksymoops(8) um kernel oops zu analysieren
- `lkcd` für spezialisierte Ausaben (linux kernel crash dump)

gdb - ddd

- `gdb /usr/src/linux/vmlinux /proc/kcore`
- (mehr bei den praktischen Beispielen)
- Magic SysReq Tasten

gdb - ddd

- `gdb /usr/src/linux/vmlinux /proc/kcore`
- (mehr bei den praktischen Beispielen)
- Magic SysReq Tasten

kgdb - Übersicht

- kgdb Kernel Patch (http://kgdb.sourceforge.net)
- benötigt zwei Hosts
- gdb läuft auf master, kgdb auf target

kgdb - Übersicht

- kgdb Kernel Patch (http://kgdb.sourceforge.net)
- benötigt zwei Hosts
- gdb läuft auf master, kgdb auf target

kgdb - Übersicht

- kgdb Kernel Patch (http://kgdb.sourceforge.net)
- benötigt zwei Hosts
- gdb läuft auf master, kgdb auf target

kgdb - Übersicht

- `gdb /path/to/vmlinux`
- target remote udp:192.168.1.3:6443

kgdb - Übersicht

- `gdb /path/to/vmlinux`
- target remote udp:192.168.1.3:6443

uml - user mode linux

- linux im userspace ausführen
- `linux ddd gdb gdb-pid=<pid> ubd0=root_fs root=/dev/udb0`
- `att 1` - jetzt debuggbar mit gdb oder ddd (auch breakpoints :-)

uml - user mode linux

- linux im userspace ausführen
- `linux ddd gdb gdb-pid=<pid> ubd0=root_fs root=/dev/udb0`
- `att 1` - jetzt debuggbar mit gdb oder ddd (auch breakpoints :-)

uml - user mode linux

- linux im userspace ausführen
- `linux ddd gdb gdb-pid=<pid> ubd0=root_fs root=/dev/udb0`
- `att 1` - jetzt debuggbar mit gdb oder ddd (auch breakpoints :-)

x86 Implementierung - Debug Register

- 4 debug address register (DR0 - DR3)
- 1 status register, 1 controll register (DR6 - DR7)
- Ansteuerung nur im Ring 0

x86 Implementierung - Debug Register

- 4 debug address register (DR0 - DR3)
- 1 status register, 1 controll register (DR6 - DR7)
- Ansteuerung nur im Ring 0

x86 Implementierung - Debug Register

- 4 debug address register (DR0 - DR3)
- 1 status register, 1 controll register (DR6 - DR7)
- Ansteuerung nur im Ring 0

x86 Implementierung - Debug Address Register

- DR0 - DR3
- beeinhalten linerae Adreesen
- Eigenschaften in abhaengigkeit von DR7 (Controll Register)
- kooperiert mit MMU - virtuelle/physikalische Adressen

x86 Implementierung - Debug Address Register

- DR0 - DR3
- beeinhalten linerae Adreesen
- Eigenschaften in abhaengigkeit von DR7 (Controll Register)
- kooperiert mit MMU - virtuelle/physikalische Adressen

x86 Implementierung - Debug Address Register

- DR0 - DR3
- beeinhalten linerae Adreesen
- Eigenschaften in abhaengigkeit von DR7 (Controll Register)
- kooperiert mit MMU - virtuelle/physikalische Adressen

x86 Implementierung - Debug Address Register

- DR0 - DR3
- beeinhalten linerae Adreesen
- Eigenschaften in abhaengigkeit von DR7 (Controll Register)
- kooperiert mit MMU - virtuelle/physikalische Adressen

x86 Implementierung - Debug Controll Register (DR7)

- steuert separate Bediengung für DR0 - DR3
- Moeglichkeiten (R/W0 - R/W3)
  - 00 Ausfuehrungsbreakpoint
  - 01 Veränderungen (writes)
  - 10 undefiniert
  - 11 lesender oder schreibender Zugriff (read, write)
- Ueberwachungsbreite (LEN0 - LEN3)
  - 00 byte
  - 01 half-word
  - 10 undefiniert
  - 11 word

x86 Implementierung - Debug Controll Register (DR7)

- steuert separate Bediengung für DR0 - DR3
- Moeglichkeiten (R/W0 - R/W3)
  - 00 Ausfuehrungsbreakpoint
  - 01 Veränderungen (writes)
  - 10 undefiniert
  - 11 lesender oder schreibender Zugriff (read, write)
- Ueberwachungsbreite (LEN0 - LEN3)
  - 00 byte
  - 01 half-word
  - 10 undefiniert
  - 11 word

x86 Implementierung - Debug Controll Register (DR7)

- steuert separate Bediengung für DR0 - DR3
- Moeglichkeiten (R/W0 - R/W3)
  - 00 Ausfuehrungsbreakpoint
  - 01 Veränderungen (writes)
  - 10 undefiniert
  - 11 lesender oder schreibender Zugriff (read, write)
- Ueberwachungsbreite (LEN0 - LEN3)
  - 00 byte
  - 01 half-word
  - 10 undefiniert
  - 11 word

x86 Implementierung - Debug Status Register (DR6)

- Dient für Abfrage welche Bediengung eintrat

x86 Implementierung - Interupt 3

- nützlich bei vielen Breakpoints
- nur Breakpoints (keine Watchpoints, . . .), logisch!
- Watchpoints die nicht in Hardware gegossen sind: single step, memcmp $\Rightarrow$ turbo taste aus!
- DR6 kann genutzt werden um Bediengungen abzufragen
- Bonbon: `maint show-debug-regs` zeigt DR0 - DR7

x86 Implementierung - Interupt 3

- nützlich bei vielen Breakpoints
- nur Breakpoints (keine Watchpoints, . . .), logisch!
- Watchpoints die nicht in Hardware gegossen sind: single step, memcmp $\Rightarrow$ turbo taste aus!
- DR6 kann genutzt werden um Bediengungen abzufragen
- Bonbon: `maint show-debug-regs` zeigt DR0 - DR7

x86 Implementierung - Interupt 3

- nützlich bei vielen Breakpoints
- nur Breakpoints (keine Watchpoints, . . .), logisch!
- Watchpoints die nicht in Hardware gegossen sind: single step, memcmp $\Rightarrow$ turbo taste aus!
- DR6 kann genutzt werden um Bediengungen abzufragen
- Bonbon: `maint show-debug-regs` zeigt DR0 - DR7

x86 Implementierung - Interupt 3

- nützlich bei vielen Breakpoints
- nur Breakpoints (keine Watchpoints, . . .), logisch!
- Watchpoints die nicht in Hardware gegossen sind: single step, memcmp $\Rightarrow$ turbo taste aus!
- DR6 kann genutzt werden um Bediengungen abzufragen
- Bonbon: `maint show-debug-regs` zeigt DR0 - DR7

Linux Implementierung (UserSpace)

- ptrace(2) liefert Schnittstelle für Debugging
- long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
- Request:
  - PTRACE_TRACEME
  - PTRACE_PEEKTEXT
  - PTRACE_GETREGS
  - PTRACE_SYSCALL

Linux Implementierung (UserSpace)

- ptrace(2) liefert Schnittstelle für Debugging
- long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
- Request:
  - PTRACE_TRACEME
  - PTRACE_PEEKTEXT
  - PTRACE_GETREGS
  - PTRACE_SYSCALL

Linux Implementierung (UserSpace)

- ptrace(2) liefert Schnittstelle für Debugging
- long ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
- Request:
  - PTRACE_TRACEME
  - PTRACE_PEEKTEXT
  - PTRACE_GETREGS
  - PTRACE_SYSCALL

Linux Implementierung (KernelSpace)

- Schnittstelle: ptrace(2)
- `kernel/ptrace.c` Implementierung
- Init und „ current“ nicht „tracebar“ `if (task->pid <= 1) goto bad; if (task == current) goto bad;`
- Prozess nur einmal „tracebar“

Linux Implementierung (KernelSpace)

- Schnittstelle: ptrace(2)
- `kernel/ptrace.c` Implementierung
- Init und „ current“ nicht „tracebar“ `if (task->pid <= 1) goto bad; if (task == current) goto bad;`
- Prozess nur einmal „tracebar“

Linux Implementierung (KernelSpace)

- Schnittstelle: ptrace(2)
- `kernel/ptrace.c` Implementierung
- Init und „ current“ nicht „tracebar“ `if (task->pid <= 1) goto bad; if (task == current) goto bad;`
- Prozess nur einmal „tracebar“

Linux Implementierung (KernelSpace)

- Schnittstelle: ptrace(2)
- `kernel/ptrace.c` Implementierung
- Init und „ current“ nicht „tracebar“ `if (task->pid <= 1) goto bad; if (task == current) goto bad;`
- Prozess nur einmal „tracebar“

Für den Urlaub

- Manual Pages
  man {ptrace,gdb,gcc,readelf,nm}
- Linux Kernel Documentation
  kgdb, oops-tracing, . . .
  . . . use the source, luke!

Richard M. Stallman and Roland H. Pesch
Using GDB: A Guide to the GNU Source-Level Debugger

John Gillmore
GDB Internals

Für den Urlaub

- Manual Pages
  man {ptrace,gdb,gcc,readelf,nm}
- Linux Kernel Documentation
  kgdb, oops-tracing, . . .
  . . . use the source, luke!

Richard M. Stallman and Roland H. Pesch
Using GDB: A Guide to the GNU Source-Level Debugger

John Gillmore
GDB Internals

Für den Urlaub

- Manual Pages
  man {ptrace,gdb,gcc,readelf,nm}
- Linux Kernel Documentation
  kgdb, oops-tracing, . . .
  . . . use the source, luke!

Richard M. Stallman and Roland H. Pesch
Using GDB: A Guide to the GNU Source-Level Debugger

John Gillmore
GDB Internals

Für den Urlaub

- Manual Pages
  man {ptrace,gdb,gcc,readelf,nm}
- Linux Kernel Documentation
  kgdb, oops-tracing, . . .
  . . . use the source, luke!

Richard M. Stallman and Roland H. Pesch
Using GDB: A Guide to the GNU Source-Level Debugger

John Gillmore
GDB Internals

This is the end – the end my only friend.