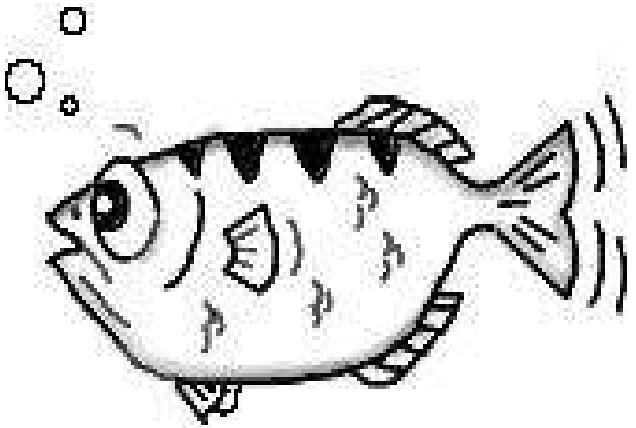
Umgang mit GDB

Referat im Rahmen des Proseminar *Unix-Tools*
an der Technischen Universität München
im Sommersemester 2005

Aleksandar Kanchev kanchev@in.tum.de
Maksym Marchenko marchenk@in.tum.de



Vortragsinhalt

0 GDB: The GNU Project Debugger

GDB: Vier Hauptbestimmungen

- **Ihr Programm starten und alle Bedingungen angeben, die das Programm beeinflüssen können**
- **Anhalten des Programms beim Auftretten von bestimmten Bedingungen**
- **Untersuchen, was geschah, als das Programm angehalten wurde**
- **Ihr Programm ändern, um die Folgen eines Fehlers korrigieren zu können und um weitere Fehler finden und beheben zu könnten**

1 GDB starten und beenden

1.1 GDB starten und beenden

- **$ *gdb***
  - Zum Starten einfach *gdb* in der Kommandozeile eingeben

- **$ *gdb program***
  - als erstes Argument – untersuchte Programm
- **$ *gdb program dump***
  - als zweites Argument kann man Speicherdump angeben
- **$ *gdb program 1234***
  - oder PID eines bereits laufenden Prozesses als zweites Argument
- **$ *gdb -help***
  - gibt mögliche Startparameter aus
- (gdb) ***quit***
  - gdb verlassen

1 GDB starten und beenden

1.2 Start – optionen

- **-- cd *dir***
  - Verzeichnis dir als aktuelle benutzen
- **-- *pid number***
  - Sich zum laufenden prozess mit PID *number* anschliesssen
- **-- tty *device***
  - GDB starten mit *device* als standartmäßiges I/O-Gerät
- **- *b bps***
  - Übertragungsgeschwindigkeit beim serial port einstellen (für remote debugging)
- **- *w***
  - benutzt GUI (falls vorhanden ist)

1 GDB starten und beenden

1.3 Shell-Befehle ausführen

- **shell *command string***
  - ruft Standard Shell auf und führt *command string* aus
- **make *make-args***
  - make mit argumenten kann ohne “shell” aufgerufen werden
  - gleichwertig mit *shell make make-args*
- **set logging on/off**
  - Protokollierung ein-/ausschalten
- **set logging file *file***
  - Protokolierung in *file* schreiben. Voreinstellung - gdb.txt
- **show logging**
  - Zeigt aktuelle Einstellungen der Protokolierung an

2 GDB-Befehle

2.1 Befehl-Vervollständigung und help

- **Befehl-vervollstandigung mit “Tab”**
  - Ausgehend von Anfangsbuchstaben, sonst gibt die Varianten aus
- **(gdb) help**
  - Gibt kurze Liste mit Kommandoklassen aus
- **(gdb) help klasse**
  - listet zu *klasse* gehörige Befehle aus
- **(gdb) help befehl**
  - kurze Hilfe zum *befehl*
- **(gdb) apropos *apr***
  - Suche nach *apr* in GDB-Befehlen und dessen Dokumentation
- **(gdb) complete *args***
  - gibt aus mögliche Beendungen von *args*

2 GDB-Befehle

2.2 info, set und show

- **info**
  - Zeigt programm-status an. Z.B. info registers ; info breakpoints ...

- **set**
  - Ausdrucksergebnis einer Umgebungsvariable zuzuweisen

- **show**
  - im gegensatz zu *info* zeigt GDB-Status an
  - z.B. show version

3 Programmausführung unter GDB

3.1 Kompilieren und ausführen

- ***- g***
  - Schlüssel beim kompilieren, der GDB-nutzung ermöglicht

- **(gdb) r (run)**
  - Programm ausführen

- **(gdb) run args**
  - startet Programm mit angegebenen Parametern

- **(gdb) set *args***
  - gibt die Argumente an (gilt ab nächstem Start)
  - show args – zeigt die Argumente an, mit dennen das Programm läuft

- **(gdb) cd *dir***
  - ändert Arbeitsverzeichnis zum *dir* um
  - pwd – zeigt aktuelle das Arbeitsverzeichnis

3 Programmausführung unter GDB

3.2 I/O-Umleitung und Debugging des laufenden Prozesses

- **info terminal**
  - zeigt Terminal-modi, die unsere Programm benutzt
- **run > *output-file***
  - Shell-like Output-Umleitung
- **tty /dev/ttyb**
  - Default-terminal einstellen
- **attach PID**
  - laufenden Prozess mit PID debuggen
- **continue**
  - angehaltenen Prozess fortsetzen
- **detach**
  - und loslassen vom GDB-control

3 Programmausführung unter GDB

3.3 Multiple-threads Debugging

- **Automatische Benachrichtigung über neuen thread**
- **Breackpoint für jeden thread einzeln einstellen**
- **info threads**
  - zeigt die Information über die im Programm vorhandene Threads an
- **thread *num***
  - schaltet auf thread *num* um
- **thread apply [threadno] [all] *args***
  - Befehl auf eine oder alle Threads anwenden

3 Programmausführung unter GDB

3.4 Programme mit vielen Prozessen

- **set follow-fork-mode *mode***
  - wobei *mode* ist
  - parent
  - child
  - ask

- **show follow-fork-mode**
  - zeigt die aktuelle Einstellung von follow-fork-mode an

4. Halten und Fortsetzen

4.1. Breakpoints, Watchpoints und Catchpoints

- **Breakpoint**
  - einfaches Anhalten
  - konditioneles Anhalten
  - Ziel als Funktion, absolute Addresse oder Quellcodezeile
  - *enable, disable, delete*
  - eindeutige Nummer

- **Watchpoint**
  - speziele Breakpoint
  - hält an, wenn einen booleschen Ausdruck wahr wird

- **Catchpoint**
  - speziele Breakpoint
  - hält an, wenn einen *C++ Exception* auftritt

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.1. Setzen von Breakpoints

- **break**
  - setzt Breakpoint auf die nächste Instruktion
  - um anzuhalten, wird verlangt, dass mindestens 1 Instruktion ausgeführt ist
- **break *function***
  - Breakpoint am Anfang der Zielfunktion *function*
- **break *linenum***
  - Breakpoint an Zeile *linenum* in der aktuellen Quellencodedatei
- **break *+/-offset***
  - Breakpoint an *+/-offset* Zeilen von der aktuellen Zeile
- **break **address***
  - Breakpoint an der Speicheraddresse *address*
- **break *filename:linenum***
- **break *filename:function***

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.1. Setzen von Breakpoints

- **break ... *if cond***
  - hält an, wenn der booleschen Ausdruck *cond* wahr ist
- **tbreak *args***
  - Breakpoint wird gelöscht beim ersten Erreichen
- **hbreak *args***
  - Hardware-Breakpoint
- **thbreak *args***
  - Kombination zwischen **tbreak** und **hbreak**
- **rbreak *regexp***
  - unbedingte Breakpoint für alle Funktionen, die von *regexp* gematcht sind

Beispiel:

(gdb) **break**
(gdb) **break main**
(gdb) **tbreak printf**
(gdb) **rbreak foo***

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.1. Setzen von Breakpoints

- **info *breakpoints***
  - gibt eine Tabelle mit allen Break-, Watch- und Catchpoints aus
  - Tabellenfelder:
    1) *Breakpointnummer*
    2) *Type:* Break-, Watch-, Catchpoint
    3) *Enabled* oder *Disabled*
    4) *Address:* Speicheraddresse oder <PENDING>
    5) *What:* wo die Breakpoint in der Quelldatei sich befindet

- **pending Breakpoints**
  - wenn das Ziel einer Breakpoint nicht übersetzt werden kann
  - wenn das Ziel übersetzt wird, wird das pending Breakpoint zu normaler
  - *set breakpoint pending (auto|on|off)*
  - *show breakpoint pending*

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.2. Setzen von Watchpoints

- wenn der Haltepunkt unbekannt ist

- **watch *expr***
  - Programm unterbricht, wenn *expr* nicht mehr wahr ist
- **rwatch *expr***
  - nur Hardware
  - es wird gehalten, wenn das Programm den Wert von *expr* ausliesst
- **awatch *expr***
  - nur Hardware
  - wenn *expr* gelesen oder verändert wird
- **info *watchpoints***
  - Alias von *info breakpoints*
- **set can-use-hw-watchpoints 0/1**
  - verbietet / erlaubt GDB Hardware-Watchpoints zu nutzen
- **show can-use-hw-watchpoints**
  - gibt den Wert der Variable aus

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.3. Löschen von Breakpoints

- **clear**
  - löscht alle Breakpoints, die für die nächste Instruktion bestimmt sind
- **clear** ***[filename:]function***
  - löscht alle Breakpoints vom Anfang der Funktion *function*
- **clear** ***[breakpoints] [range]***
  - löscht alle Breakpoints
  - löscht die durch *breakpoints* und *range* bestimmte Breakpoints

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.4. Aktivieren und Deaktivieren von Breakpoints

- **Zustände einer Breakpoint**
  - *Enabled:* die Breakpoint hält das Programm an
  - *Disabled:* die Breakpoint beeinflußt das Programm nicht
  - *Enabled once:* die Breakpoint hält das Programm nur 1-mal an
  - *Enabled for deletion:* die Breakpoint wird nach dem ersten Halten gelöscht

- **enable *[breakpoints] [range]***
  - aktiviert alle Breakpoints
  - aktiviert die durch *breakpoints* und *range* bestimmte Breakpoints
- **disable *[breakpoints] [range]***
  - deaktiviert alle Breakpoints
  - deaktiviert die durch *breakpoints* und *range* bestimmte Breakpoints
- **enable *[breakpoints]* once *range* ...**
  - aktiviert die Breakpoints für 1-maliges Halten
- **enable *[breakpoints]* delete *range* ...**
  - aktiviert die Breakpoints für 1-maliges Halten, danach werden die gelöscht

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.5. Bedingte Breakpoints

- Breakpoint mit booleschen Ausdruck als Bedingung
- Bedingungen können Seiteneffekte haben, wie Aufruf von Funktionen
- im allg. sind die Bedingungen mit dem *if* Argument des *break* Kommandos definiert

➢ **condition *bnum expression***

- definiert eine Bedingung für Break-, Watch- oder Catchpoint *bnum*

➢ **condition *bnum***

- löscht die Bedingung von Breakpoint *bnum*

4.1. Breakpoints, Watchpoints und Catchpoints

4.1.6. Kommandolisten für Breakpoints

- **commands** ***[bnum]***
  ... command-list ...
  end

  - definiert eine Kommandoliste für Breakpoint *bnum*
  - leere Liste entfernt alle vorherige Kommandos
  - wenn *bnum* nicht gegeben ist, bezieht sich das *command* Kommando auf die am letzten gesetzte Breakpoint
  - es können die *continue* oder *step* Kommandos verwendet werden, um das Programm weiter laufen zu lassen
  - wenn das *silent* Kommando am Anfang der Liste steht, wird keine Mitteilung ausgegeben, dass die Breakpoint erreicht worden ist
  - Die *echo, output* und *printf* können verwendet werden, um formatierte Ausgabe anzugeben

4.2. Normales und schrittweises Ausführen und Fortfahren

- **continue** ***[ignore-count]***
  - das Programm fährt von der Addresse, wo es angehalten war, fort
  - *ignore-count* spezifiziert wieviel mal die Breakpoint übersprungen wird
- **step**
  - schrittweises Ausführen und Fortfahren
  - verfolgt Funktionen
- **step** ***count***
  - führt schrittweises Ausführen *count*-mal aus
- **next** ***[count]***
  - schrittweises Ausführen und Fortfahren
  - verfolgt Funktionenaufrufe nicht
- **set step-mode on**
  - das *step* Kommando haltet bei der ersten Instruktion von einer Funktion, für die es keine Debuginformation gibt
- **set step-mode off**
  - das *step* Kommando überspring Funktionen ohne Debuginformation
- **show step-mode**

4.2. Normales und schrittweises Ausführen und Fortfahren

- **finish**
  - das Programm fährt bis zum Verlassen der aktuellen Funktion fort
- **until**
  - verfolgt die Schleifen nur 1mal
- **until** ***location***
  - das Programm fährt bis zum Erreichen von *location* oder bis zum Verlassen der aktuellen Funktion fort
  - *location* ist wie das Argument von **break**
- **advance** ***location***
  - die gleiche Funktionalität wie until
  - verfolgt Rekursionen
- **stepi** ***[arg]***
  - führt eine Instruktion aus und stoppt
  - führt *arg* Instruktionen aus und stoppt
- **nexti** ***[arg]***
  - gleiche Funktionalität wie **stepi**, verfolgt aber keine Funktionen

5. Den Stack untersuchen

5.1. Stackframe

- **stack frame**
  - enthält Daten, die sich auf einem Funktionsaufruf beziehen
  - diese sind Funktionsargumente, lokale Variablen der Funktion und die Funktionsaddresse
  - jeder Stackframe ist als Feld von Bytes im Programm gespeichert
  - jeder Stackframe bekommt eine eindeutige Nummer in GDB zugeordnet
  - jeder Rekursivenaufruf generiert einen neuen Stackframe

- **call stack**
  - enthält alle **stack frame**s
- **frame pointer register**
  - enthält die Speicheraddresse des ersten Bytes vom Stackframe
- **innermost frame**
  - der Stackframe der aktuellen Funktion
  - dieser Stackframe hat die Nummer 0

5. Den Stack untersuchen

5.1. Stackframe

- **frame** ***[args]***
  - erlaubt den Umzug vom aktuellen Stackframe zu einem anderen
  - *args* ist die Nummer vom Zielframe
  - ohne Parameter, gibt das Kommando Informationen über den aktuellen Stackframe aus

- **select-frame**
  - gleiche Funktionalität wie das *frame* Kommando
  - gibt keine Informationen über den aktuellen Stackframe aus

5. Den Stack untersuchen

5.2. Backtraces

- **backtrace** ***[-[n]]***
  - gibt eine Liste von Stackframes, die den Funktionsaufruf verfolgen
  - gibt nur die ersten *n* Stackframes aus
  - gibt nur die letzten *n Stackframes* aus
- **set backtrace limit** ***n***
  - limitiert die Anzahl der Backtraces, die Angezeigt werden
  - *n = 0* bedeutet unbegrenzt
- **show backtrace limit**
- **set backtrace past-main [on]**
  - Backtraces werden nach dem Eingangspunkt ausgegeben
- **set backtrace past-main off**
  - keine Backtraces werden nach dem Eingangspunkt ausgegeben
- **show backtrace past-main**
  - gibt an, ob Backtraces nach dem Eingangspunkt ausgegeben werden

5. Den Stack untersuchen

5.3. Wechseln zwischen Stackframes

- **frame *n***
  - wechseln zum *n*-ten Stackframe
- **frame *addr***
  - wechseln zum Stackframe, der sich auf Speicherdresse *addr* befindet
  - nützlich wenn die Stackframes durch einen Bug beschädigt worden sind
- **up *n***
  - wechseln vom aktuellen zum *n*-ten Stackframe
  - wenn *n* positiv ist, ist die Richtung dem inneren Stackframe entgegen
- **down *n***
  - wechseln vom aktuellen zum *n*-ten Stackframe
  - wenn *n* positiv ist, ist die Richtung dem äußeren Stackframe entgegen
- **up-silently *n***
  down-silently *n*
  - *up*- und *down*-Varianten ohne Ausgabe

5. Den Stack untersuchen

5.4. Informationen über Stackframe

- **frame**
  - kurzgefasste Information über den aktuellen Stackframe
- **info frame** ***[n | addr]***
  - gibt detaillierte Information über den Stackframe:
    - die Speicheraddresse vom Stackframe
    - die Speicheraddresse vom nächsten Stackframe unten
    - die Speicheraddresse vom nächsten Stackframe oben
    - die Quellsprache zum Stackframe
    - die Speicheraddresse von den lokalen Variablen des Stackframes
    - Gespeicherte Register
- **info args**
  - gibt zeilenweise eine Liste aller Arugemente des aktuellen Stackframes
- **info locals**
  - gibt zeilenweise eine Liste aller lokalen Variablen des aktuellen Stackframes

6. Untersuchen von Quellcodedateien

6.1. Quellcode ansehen

- **list** ***linenum***
  - gibt die Quellcodezeilen neben Zeile *linenum* aus
- **list** ***function***
  - gibt die Quellcodezeilen neben dem Anfang der Funktion *function* aus
- **list** ***[+offset]***
  - gibt die nachfolgenden Quellcodezeilen aus
- **list** ***-[offset]***
  - gibt die vorigen Quellcodezeilen aus
- **set listsize** ***count***
  - setzt die Anzahl der Quellcodezeilen, die vom *list* Kommando angezeigt werden
- **show listsize**
  - gibt die Anzahl der Quellcodezeilen aus, die vom *list* Kommando angezeigt werden

7. Untersuchen von Daten

- **print** ***[/f] expr***
  - *expr* ist ein Quellcodeausdruck
  - */f* spezifiziert auf welcher Art der Wert von *expr* ausgegeben wird

7.1. Untersuchen von Speicher

- **x** ***/nfu addr***
  - *n* spezifiziert die Anzahl der Speicherblöcke, die angezeigt werden
  - *f* spezifiziert den Format der Speicherblöcke:
    - *'s'* für String
    - *'i'* für Maschineninstruktion
    - *'x'* für Hexadezimal
  - *u* spezifiziert die Größe:
    - *'b'* Bytes
    - *'h'* Halbwort (2 x b)
    - *'w'* Wort (4 x b)
    - *'g'* Doublewort (8 x b)

Beispiel:

Quellcode:

```
int *array = (int *)malloc(len*sizeof(int))
```

```
(gdb) print  *array@len
/*                  (type[])value                    */
(gdb) print/x  (short[2])0x12345678
$1 = { 0x1234, 0x5678 }
(gdb) x/3uh  0x54320
```