# GDB som debugger i C

Hugo Hammer og Håvard Berland

19. september 2003

## Introduksjon

*Gdb* er et program som er effektivt for å debugge kode i C. Programmet fungerer slik at du kan gå kontrollert gjennom koden din kodelinje for kodelinje.

En vanligere debuggingsteknikk som du sikkert har brukt i java og kanskje også i C, er å sette inn masse `printf` kommandoer for å se hvor langt man kommer i programmet sitt før noe går galt, og også for å se på variabler. Men man kan *ikke* alltid stole på at resultatet av slik printf-bruk er riktig (i kanskje 95% av tilfellene går det likevel bra). Derfor kan det være en fordel å lære seg en skikkelig debugger, som gdb.

Gdb kan man kjøre enten fra kommandolinja, eller fra emacs. Vi anbefaler bruk av emacs.

## Kompilering av kode

La oss si vi ønsker å debugge C-koden i fila `program.c` Vi ønsker å legge inn noe informasjon i den kjørbare fila slik at det går an å finne ut hvilket linjenummer som svarer til hvilke instruksjoner i den kjørbare fila. Dette gjør vi ved å legge til opsjonen `-g` til kompilatoren `gcc`:

```
lastebil:~:$ gcc -g -o program program.c
```

Det vil da bli generert en fil `program` som kan kjøres fra kommandolinja, men vi vil heller la gdb kjøre fila.

## Kjøring av gdb

Gdb vil vi helst starte fra emacs, da kan vi få emacs til å utnytte linjeinformasjonen i den kjørbare fila til å flytte oss direkte til linja med problemer på. For å prøve dette gjør du følgende

1. Start emacs, skriv

   ```
   lastebil:~:$ emacs program.c
   ```

   på kommandolinja.

2. Splitt emacs-vinduet i to ved å trykke `C-x 2` og bytt til det bufferet (emacs kaller sine indre vindu for buffere) som du vil kjøre gdb i (trykk `C-x o` for bytte buffer).

3. Tast så `M-x` (Esc + x). Det vil da stå `M-x` nederst i emacs-vinduet, og du skriver videre inn `gdb`. Alternativt kan du velge "Debugger..." fra "Tools"-menyen.

4. Det vil da stå i Emacs sin meldingslinje nederst

   ```
   Run gdb (like this): gdb
   ```

   og du taster inn `program` (eller hva navnet er på filen du ønsker å kjøre i gdb heter) og trykk *enter*.

Du vil da få opp omtrent noe slikt i emacs:

```
Current directory is ~/
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.16 (sparc-sun-solaris2.3),
Copyright 1996 Free Software Foundation, Inc...
(gdb)
```

Gdb er nå klar for å motta kommandoer fra deg om hva du vil gjøre. "Promptet" `(gdb)` betyr at gdb venter fra kommandoer fra deg. Et tips for senere er at det å bare trykke *enter* her vil gjøre at gdb kjører sist kjørte kommando. De fleste kommandoer i gdb har en kortversjon og en langversjon. Vi vil konsekvent skrive den lange versjonen, og den korte versjonen i parantes rett bak når den finnes. Mange av kommandoene kan du også taste i kildekodevinduet i emacs, via tastekombinasjonene du finner i menyen "Gud".

Skriver du `run (r)`, vil programmet dit bli kjørt helt til det dukker opp en feil eller til programmet er ferdig. Dette er sjelden ønskelig når vi skal debugge. Derfor er det lurt å legge inn såkalte *breakpoints* med kommandoen `break (b)` i koden på forhånd. Dette kan du gjøre ved å taste `C-x <space>` på linja du vil ha breakpoint på i kildekodebufferet, eller du kan gjøre det på gdb sin kommandolinje, skriv

```
(gdb) b 34
```

hvis du vil ha et breakpoint på linje 34 i hovedfila for programmet ditt (den fila med `main()`-funksjonen). Hvis du vil ha breakpoint på linje 29 i en annen fil (la oss kalle den `kode2.c`) enn den programmet er i, skriver du `(gdb) b kode2.c:29` eller finn fila med emacs og bruk `C-x <space>`

Et breakpoint er et sted hvor programmet vil stoppe når du har startet det med `r`. Når koden når et breakpoint vil emacs åpne fila med korresponderende kildekode hvis den ikke er åpnet allerede, og en liten `=>` viser hvor i koden programmet har kommet. Du har nå en rekke kommandoer tilgjengelig som du kan manøvrere deg videre i koden med på en kontollert måte. Desverre er det ingen mulighet for å gå bakover i koden igjen hvis du så noe interessant i koden mens programmet gikk forbi. Da må du bare kjøre programmet begynne på nytt.

Aktuelle kommandoer du nå kan skrive er (se også under emacs-menyen "Gud"):

`c` Programmet kjører til neste breakpoint i koden (*continue*), i emacs `C-c C-r`.

`s` Hvis du kommer til en kodelinje hvor det er kall til en annen funksjon kan du gå inn i koden til denne funksjonen ved å skrive `s`.

`p` Kan benyttes til å skrive ut verdien til en variabel. Skriv `(gdb) p variabelnavn`. Hvis man har en vektor med verdien av første element i vektoren under `vektor[0]`, og at det er totalt 10 elementer i vektoren. Verdien av disse 10 elementene kan man da se ved å skrive `(gdb) p vektor[0]@10`.

`up/down` I et program vil gjerne en funksjon kalle en funksjon som igjen kaller en annen funksjon osv. Det kan da være lett å miste oversikten på hvor vi er i funksjons-hierarkiet. Vi kan da benytte `up` og `down` for å få kontroll på dette. Skriver vi `(gdb) up` går vi til funksjonen som ligger rett over oss i hierarkiet og skriver vi `(gdb) down` går vi tilbake til der vi var. Merk at dette gjøres mens koden er stoppet på et breakpoint, og vi endrer ikke hvor langt vi har kjørt i koden. Dette er rett og slett en hjelp for å få oversikt på hvor vi er i koden.