Deluppgift 7 Felsökning med debugger

Saker som kan vara bra att känna till

Observera att du behöver standardversionen 3.4.6 av gcc. Om du i tidigare kurs installerat en nyare eller äldre version måste du återställa till standardversionen:

```
module list
  1) unix/local
  2) unix/std
  ...
  9) prog/gcc/4
 10) unix/sfw
module rm prog/gcc/4
module add prog/gcc
```

Notera även att debuggern inte klarar hantera brytpunkter i implementationsfiler vars namn innehåller mellanslag. Byt ut mellanslag mot understrykningstecken (_). En gammal programmerare använder bara engelska bokstäver, siffror, vanligt bindestreck och understrykningstecken i filnamn.

Tutorial

Programmet `debug.c` finns bland de givna filerna på hemsidan. Programmet innehåller dock ett allvarligt fel du skall felsöka. *Felsökning kommer bli en stor del av Pintos-laborationerna.* Börja med att kompilera programmet:

```
gcc -Wall -Wextra -std=c99 -pedantic -g debug.c
```

Starta sedan programmet med hjälp av debuggern `gdb`. Det finns en grafisk front-end `ddd`, men effektivast är att använda text-mode `gdb`:

```
gdb a.out
```

Debuggern startar, läser in programmet `a.out` och presenterar en kommandoprompt `(gdb)`. Skriv `run` (istället för `a.out`) för att starta programmet. (Om main förväntar sig argument på kommandoraden kan de anges “som vanligt” efter `run`.) Programmet kommer nu att krascha eftersom det innehåller ett fel. Skriv `bt` eller `backtrace` för att se hela kedjan av funktionsanrop som ledde fram till krashen. Med större program är denna information ovärderlig, men i detta fall finns bara main. Vill du nu undersöka värdet på olika variabler kan du använda kommandot display. Några exempel:

```
(gdb) display *bufi
1: *bufi = 0x0
(gdb) display bufi
2: bufi = (char **) 0xffbfe25c
(gdb) display bufend
3: bufend = (char **) 0xffbfe25c
(gdb) display *(bufend-1)
4: *(bufend - 1) = 0xffbfe268 "sihtgubed"
```

Som du ser kan man skriva komplicerade uttryck (C-kod) för att undersöka innehållet av pekare. Skriv nu kommandot `break 12` för att stoppa programmet på rad 12 vid nästa körning. Starta om programmet genom att skriva `run` och bekräfta. Programmet stannar vid rad 12, och visar alla displayuttryck. Skriv `undisplay` för att inte visa uttrycken igen. För att fortsätta program-körningen kan du nu skriva `next`, `nexti`, `step`, `stepi` eller `continue`. Om du bara trycker enter upprepas föregående kommando. Använd `help` för att se skillnaden på de olika stegningarna (`help next`).

Uppgift

Rita och följ exekveringen med alla arrayer och pekare på papper. Vad är det meningen programmet skall göra? Försök använda debuggern för att verifiera sådant du är osäker på. Det är viktigt att förstå allt som händer i minnet. När du hittat felet, rätta det och prova. Det räcker att byta ordning på två rader för att rätta felet.

```
int main()
{
  char str[] = "sihtgubed";
  char *stri = &str[8];
  char *buf[9];
  char **bufi, **bufend;
  bufi = buf;
  bufend = &buf[9];

  while (bufi != bufend){
    *bufi = stri;
    bufi++;
    stri--;
  }

  while (bufi != buf){
    *(*bufi) -= 32;
    bufi--;
  }

  while (bufi != bufend){
    printf("%c", **bufi);
    bufi++;
  }
}
```

För att använda debuggern med Pintos krävs några ytterligare steg för att komma igång. Eftersom Pintos exekverar i en emulator (`qemu`) måste en anslutning upprättas mellan emulatorn och debuggern. Detta återkommer vi till när Pintos är installerat.