UNIVERSITETET I OSLO
DET MATEMATISK-NATURVITENSKAPELIGE FAKULTET

Dagens tema 1: Minnet

- Fast minne
  - Store og små indianere
  - «align»-ing
- Noen nyttige instruksjoner
  - Vektorer
  - Hva er adressen?

Dagens tema 2: Feilsøking

- gdb
- ddd
- Valgrind
- Egne testutskrifter

INF2270

Faste variable

Faste variable lever så lenge programmet kjører. De kan gis en initialverdi.

Det vanlige er å legge slike variable i .data-segmentet.†

I C:

```
int a, b;
static char c;  /* 'static´ betyr «private» for globale! */
long d = 5;

void f (void) {}
```

I assemblerkode:

```
        .globl  a, b, d, f
        .text
f:
        ret

        .data
a:      .long   0
b:      .long   0
c:      .byte   0
        .align  2
d:      .long   5
```

† Hva skjer om de ligger i .text? Noen OS setter skrivebeskyttelse på .text.

INF2270

«Alignment» (B&O'H-boken 3.10)

Hva om vi ber CPUen utføre

```
    movl    var,%eax
```

der adressen til var er 0x-------3?

Noen prosessorer klarer ikke slikt, men x86 gjør det selv om det tar mer tid. Enda verre er det ved skriving til minnet.

Brukeren kan angi at variable skal være *alignet*, dvs ikke krysse ordgrenser:

```
    .align n
```

Denne spesifikasjonen får assembleren til å legge inn 0 eller flere byte med ett eller annet inntil adressen er har n 0-bit sist.

INF2270

Byte-rekkefølgen (B&O'H-boken 2.1.4)

De fleste datamaskiner i dag er byte-maskiner der man adresserer hver enkelt byte. short, int og long trenger da 2–4 byte.

Anta at register **%EAX** inneholder 0x01234567.

Om resultatet av

```
    movl    %eax,0x100
```

blir

	0x100	0x101	0x102	0x103	
...	01	23	45	67	...

kalles maskinen **big-endian**.

Om det blir

	0x100	0x101	0x102	0x103	
...	67	45	23	01	...

kalles den **little-endian**.

INF2270

Vektorer

En vektor er et sammenhengende område i minnet der man kan *regne* seg frem til hvert elements adresse.

```
int a[4];
```

ligger slik i minnet:

	0x104	0x108	0x10C	0x110	
...	a[0]	a[1]	a[2]	a[3]	...

INF2270

Vektorer i x86-kode

Det finnes en egen adresseringmåte for å slå opp i en vektor:

20(**%eax**,**%ebx**,n)

som gir adressen

%eax + n×**%ebx** + 20

n må være 1, 2, 4 eller 8.

```
        .globl arrayadd
# Navn:         arrayadd.
# Synopsis:     Summerer verdiene i en vektor.
# C-signatur:   int arrayadd (int a[], int n).
# Registre:     %eax:   summen så langt
#               %ecx:   indeks til a (teller ned)
#               %edx:   adressen til a
arrayadd:
        pushl   %ebp            # Standard
        movl    %esp,%ebp       # funksjonsstart.

        movl    $0,%eax         # sum = 0.
        movl    12(%ebp),%ecx   # ix = n.
        movl    8(%ebp),%edx    # a.

a_loop: decl    %ecx            # while (--ix
        js      a_exit          #            >=0) {
        addl    (%edx,%ecx,4),%eax # sum += a[ix].
        jmp     a_loop          # }

a_exit: popl    %ebp            # return sum.
        ret                     #
```

INF2270

Instruksjonen lea

Instruksjonen lea («load affective address») fungerer som en mov men henter adressen i stedet for verdien.

```
eks1:   leal    var,%eax

eks2:   movl    index,%edx
        leal    array,%eax
        leal    (%eax,%edx,4),%ecx

        .data
var:    .long   12
array:  .fill   100
index:  .long   8
```

INF2270

Debuggere

En «debugger» er et meget nyttig feilsøkingsverktøy. Det kan

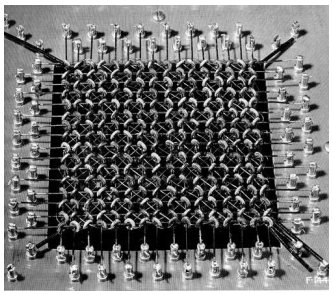
- analysere en program-dump,
- vise innholdet av variable,
- vise hvilke funksjoner som er kalt,
- kjøre programmet én og én linje, og
- kjøre til angitt stoppunkter.

Debuggeren gdb er laget for å brukes sammen med gcc. Den har et vindusgrensesnitt som heter ddd som kan brukes på Unix-maskiner.

INF2270

Programdumper
Når et program dør på grunn av en feil («aborterer»), prøver det ofte å skrive innholdet av hele prosessen på en fil slik at det kan analyseres siden.

```
> ls -l core*
-rw-------  1 dag 61440 2008-02-27 09:07 core.22577
```



(Dette kalles ofte en «core-dump» siden datamaskinene for 30–50 år siden hadde hurtiglager bygget opp av ringer med kjerne av feritt. I Unix heter denne filen derfor core.*.)

INF2270

For å bruke gdb/ddd må vi gjøre to ting:

- kompilere våre programmer med opsjonen -g, og
- angi at vi ønsker programdumper:

  ```
  ulimit -c unlimited
  ```

  hvis vi bruker bash. (Da må vi huske å fjerne programdumpfilene selv; de er noen ganger *store!*)

INF2270

Et program med feil
Hovedprogrammet; se også (B&O'H-boken 3.11):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern char *mystrcpy (char *til, char *fra);

char *s;

int main (void)
{
  mystrcpy(s, "Abc");
  printf("Teksten \"%s\" har %d tegn.", s, strlen(s));
  exit(0);
}
```

```
        .globl mystrcpy
# Navn:         mystrcpy.
# Synopsis:     Kopierer en tekst.
# C-signatur:   char *mystrcpy (char *til, char *fra)
# Registre:     AL  - tegn som flyttes
#               ECX - til (som økes)
#               EDX - fra (som økes)

mystrcpy:
        pushl   %ebp            # Standard
        movl    %esp,%ebp       # funksjonsstart.

        movl    8(%ebp),%ecx    # Hent til
        movl    12(%ebp),%edx   # og fra.
                                # do {
mys_l:  movb    (%edx),%al      #   AL = *fra
        incl    %edx            #             ++ .
        movb    %al,(%ecx)      #   til    = AL.
        incl    %ecx            #       ++
        cmpb    $0,%al          #          AL != 0
        jne     mys_l           # } while (       )

mys_x:  movl    8(%ebp),%eax    #        til.
        popl    %ebp            #
        ret                     # return
```

INF2270

Under kjøring går dette galt:

```
> gcc -m32 -g -o feil-strcpy feil-strcpy.c strcpy.s
> ./feil-strcpy
Segmentation fault (core dumped)
```

De viktigste spørsmålene da er:

❶ Hvor skjer feilen?

❷ Hva vet vi om situasjonen når feilen inntreffer?

Svarene finner vi ved å analysere programdumpene.

INF2270

Debuggeren gdb (B&O'H-boken 3.12)

Den enkleste debuggeren er gdb som finnes overalt.

```
> gdb feil-strcpy core.22577
GNU gdb Red Hat Linux (6.3.0.0-1.90rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, ...

warning: core file may not match specified executable file.
Core was generated by `./feil-strcpy'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/tls/libc.so.6...done.
Loaded symbols for /lib/tls/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  mys_l () at strcpy.s:18
18              movb    %al,(%ecx)      #   til     = AL.
(gdb)quit
```

Da vet vi *hvor* feilen oppsto.

Debuggeren ddd

Denne debuggeren (som egentlig bare er et grafisk grensesnitt mot gdb) finnes på Ifi men dessverre ikke på alle Linux-maskiner.

Programmet startes slik:

```
> ddd feil-strcpy &
```


Sjekke programdumpen

I File-menyen finner vi «Open core dump» og da ser vi *hvor* feilen oppsto:

Sjekke registrene

I Status-menyen finner vi «Registers»:

Et eksempel til
Hovedprogrammet:

```
#include <stdio.h>
#include <stdlib.h>

extern void swap (int *a, int *b);

int *pa, *pb;

int main (void)
{
  pa = malloc(sizeof(int));  pa = malloc(sizeof(int));
  *pa = 3;  *pb = 17;

  printf("*pa = %d, *pb = %d\n", *pa, *pb);
  swap (pa, pb);
  printf("*pa = %d, *pb = %d\n", *pa, *pb);
  return 0;
}
```

Assemblerfunksjonen:

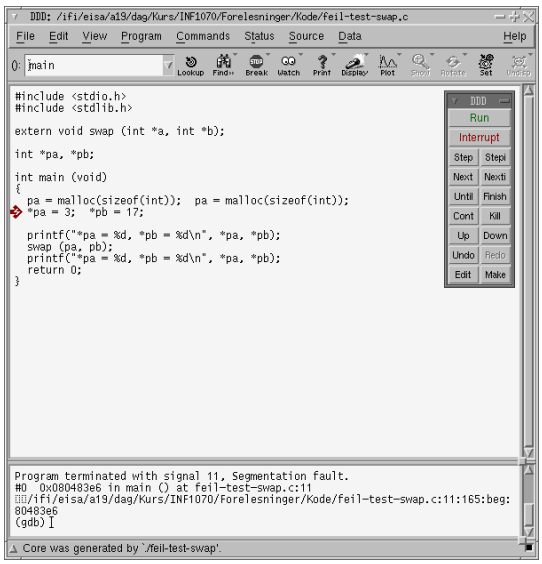
```
        .globl swap
# Navn:         swap.
# Synopsis:     Bytter om to variable.
# C-signatur:   void swap (int *a, int *b).

swap:   push    %ebp            # Standard
        movl    %esp,%ebp       # funksjonsstart

        movl    8(%ebp),%eax    # %eax = a.
        movl    12(%ebp),%ecx   # %ecx = b.

        push    (%eax)          # push *a.
        push    (%ecx)          # push *b.
        pop     (%eax)          # pop *a.
        pop     (%ecx)          # pop *b.

        pop     %ebp            # Standard retur
        ret
```

INF2270

Kjøringen:

```
> gcc -m32 -g -o feil-swap feil-swap.c swap.s
> ./feil-swap
Segmentation fault (core dumped)
> ddd feil-swap &
```

Etter «Open core dump» ser vi:



Ved å peke på navnene pa og pb ser vi at pa=0x9c06018 og pb=0x0. Dette bør fortelle oss hva som gikk galt.

INF2270

Minnelekkasje
Valgrind (http://valgrind.org/) er et ypperlig feilfinningsverktøy, spesielt for å finne minnelekkasjer.

Eksempel
Dette er et program som leser en fil, byger opp et binært søketre av ordene og skriver dem ut i sortert rekkefølge.

Vi tester dette på dagens melding /etc/motd:

OBSOBS: nøkler er endret 2008-03-07 til samme nøkkel for alle maskinene bak login.

Problemer med ssh-nøkler når du kobler deg til login.ifi.uio.no?

Se http://www.ifi.uio.no/it/ssh.html for hvordan unngå problemet fra din egendriftede maskin.

-drift@ifi.uio.no, 2008-02-25

INF2270


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node {
  char *navn;
  struct node *v, *h;
};

struct node topp = { "", NULL, NULL };;

void sett_inn (struct node *p, struct node *ny)
{
  if (strcmp(ny->navn,p->navn) < 0) {
    if (p->v) sett_inn(p->v,ny); else p->v = ny;
  } else {
    if (p->h) sett_inn(p->h,ny); else p->h = ny;
  }
}

void skriv_ut (struct node *p)
{
  if (p->v) skriv_ut(p->v);
  printf("\"%s\"\n", p->navn);
  if (p->h) skriv_ut(p->h);
}

void rydd_opp (struct node *p)
{
  if (p->v) rydd_opp(p->v);
  if (p->h) rydd_opp(p->h);
  free(p);
}

int main (int argc, char *argv[])
{
  FILE *f = fopen(argv[1], "r");
  char n[200];

  while (fscanf(f,"%s",n) != EOF) {
    struct node *nx = malloc(sizeof(struct node));
    nx->navn = strdup(n);  nx->v = nx->h = NULL;
    sett_inn(&topp, nx);
  }
  fclose(f);
  skriv_ut(&topp);  rydd_opp(&topp);

  return 0;
}
```

INF2270

Programmet ser ut til å fungere fint:

```
""
"-drift@ifi.uio.no,"
"2008-02-25"
"2008-03-07"
"OBSOBS;"
"Problemer"
"Se"
"alle"
"bak"
"deg"
"din"
"du"
"egendriftede"
"endret"
"er"
"for"
"for"
"fra"
"http://www.ifi.uio.no/it/ssh.html"
"hvordan"
"kobler"
"login."
"login.ifi.uio.no?"
"maskin."
"maskinene"
"med"
"når"
"nøkkel"
"nøkler"
"problemet"
"samme"
"ssh-nøkler"
"til"
"til"
"unngå"
```


Mer er alt bra? Vi spør Valgrind:

```
> gcc -g -O0 -o navn navn.c && valgrind --leak-check=yes navn /etc/motd
==8381== Memcheck, a memory error detector.
==8381== Using valgrind-3.2.1, a dynamic binary instrumentation framework.
==8381== For more details, rerun with: -v
==8381==
==8381== Invalid free() / delete / delete[]
==8381==    at 0x4A0541E: free (vg_replace_malloc.c:233)
==8381==    by 0x4007BB: rydd_opp (navn.c:32)
==8381==    by 0x40087D: main (navn.c:46)
==8381==  Address 0x600D00 is not stack'd, malloc'd or (recently) free'd
==8381==
==8381== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 4 from 1)
==8381== malloc/free: in use at exit: 272 bytes in 34 blocks.
==8381== malloc/free: 69 allocs, 36 frees, 1,656 bytes allocated.
==8381== For counts of detected errors, rerun with: -v
==8381== searching for pointers to 34 not-freed blocks.
==8381== checked 64,104 bytes.
==8381==
==8381== 272 bytes in 34 blocks are definitely lost in loss record 1 of 1
==8381==    at 0x4A05809: malloc (vg_replace_malloc.c:149)
==8381==    by 0x32BB875EB1: strdup (in /lib64/libc-2.5.so)
==8381==    by 0x40080D: main (navn.c:42)
==8381==
==8381== LEAK SUMMARY:
==8381==    definitely lost: 272 bytes in 34 blocks.
==8381==      possibly lost: 0 bytes in 0 blocks.
==8381==    still reachable: 0 bytes in 0 blocks.
==8381==         suppressed: 0 bytes in 0 blocks.
==8381== Reachable blocks (those to which a pointer was found) are not shown.
==8381== To see them, rerun with: --show-reachable=yes
```


Egne utskrifter
De beste feilmeldingene får vi ved å lage dem selv.

- Regn med at programmet ditt vil inneholde feil!
- Programmér feilutskrifter du kan slå av og på.
- Husk at du kan kalle C-funksjoner (dine egne og standardfunksjoner som printf) fra assemblerkode.

  (Husk bare at disse kan ødelegge **%EAX**, **%ECX** og **%EDX**.)

~inf2270/programmer/dumpreg.s anbefales:

```
        .globl  dumpreg
# Navn:         dumpreg
# Synopsis:     Skriver ut alle registrene.
# C-signatur:   void dumpreg (void).
# Registre:     Ødelegger _ingen_ registre!

dumpreg:
        pushl   %ebp            # Standard
        movl    %esp,%ebp       # funksjonsstart
        pushf                   # Gjem også flaggene,
        pushl   %eax            # EAX,
        pushl   %ecx            # ECX og EDX (siden
        pushl   %edx            # 'printf´ kan
                                # ødelegge dem)
        pushl   %edx            # Legg EDX,
        pushl   %ecx            # ECX,
        pushl   %ebx            # EBX og
        pushl   %eax            # EAX på stakken.
        movl    4(%ebp),%eax    # Legg PC (returadr)
        pushl   %eax            # på stakken.
        leal    form1,%eax      # Legg adr til form1
        pushl   %eax            # på stakken.
        call    printf          # Kall 'printf´.
        popl    %eax            #
        popl    %eax            # Rydd
        popl    %eax            # opp
        popl    %eax            # på
        popl    %eax            # stakken.
        popl    %eax            #
```

```
        pushl   %edi            # Legg EDI
        pushl   %esi            # og ESI på stakken.
        movl    0(%ebp),%eax    # Hent riktig EBP
        pushl   %eax            # og legg på stakken.
        movl    %ebp,%eax       # Riktig ESP er
        subl    $4,%eax         # EBP-4;  legg
        pushl   %eax            # den på stakken.
        lea     form2,%eax      # Legg adr til form2
        pushl   %eax            # på stakken.
        call    printf          # Kall 'printf'.
        popl    %eax            #
        popl    %eax            # Rydd
        popl    %eax            # opp
        popl    %eax            # på
        popl    %eax            # stakken.

        popl    %edx            # Hent tilbake EDX,
        popl    %ecx            # ECX,
        popl    %eax            # EAX,
        popf                    # og flaggene.
        popl    %ebp            #
        ret                     # Retur.

        .data

form1:  .asciz  "Dump: PC=%08x EAX=%08x EBX=%08x ECX=%08x EDX=%08x\n"
form2:  .asciz  "                ESP=%08x EBP=%08x ESI=%08x EDI=%08x\n"
```

INF2270

Eksempel på bruk:

```
#include <stdio.h>

extern void dumpreg (void);

void f (void)
{
  dumpreg();
}

int main (void)
{
  dumpreg();
  f();
  dumpreg();
  return 0;
}
```

```
Dump: PC=080483a7 EAX=ffc2b464 EBX=00c57ff4 ECX=ffc2b3e0 EDX=00000001
                  ESP=ffc2b3b4 EBP=ffc2b3c8 ESI=00b18ca0 EDI=00000000
Dump: PC=0804838f EAX=ffc2b464 EBX=00c57ff4 ECX=ffc2b3e0 EDX=00000001
                  ESP=ffc2b3a4 EBP=ffc2b3b8 ESI=00b18ca0 EDI=00000000
Dump: PC=080483b1 EAX=ffc2b464 EBX=00c57ff4 ECX=ffc2b3e0 EDX=00000001
                  ESP=ffc2b3b4 EBP=ffc2b3c8 ESI=00b18ca0 EDI=00000000
```

INF2270