Tölvutækni og Forritun Heimadæmi 8

brj46

October 2024

1

Við Fáum gefið fallið `mul_elem` og þýðingu þess í smalamáli:

```
long A[M][N];
long B[N][M];
long mul_elem( long i, long j )
{
return A[i][j] * B[j][i];
}
```

```
mul_elem:
leaq (%rdi,%rdi,2), %rax
leaq (%rsi,%rax,2), %rax
leaq (%rsi,%rsi,4), %rdx
leaq (%rdi,%rdx,2), %rdx
movq A(,%rax,8), %rax
imulq B(,%rdx,8), %rax
ret
```

Við viljum vita hver gilin á **M** og **N** eru:

Skoðum smalamálskóðann:

- $leaq(\%rdi, \%rdi, 2), \%rax$ Hér er i margfaldað með þremur og geymt í rax
- $leaq(\%rsi, \%rax, 2), \%rax$ Hér er rsi (sem er j) bætt við tvöfalt rax

  rax er þá $j + 6i$
- $leaq(\%rsi, \%rsi, 4), \%rdx$ Hér er rsi margfaldað með 5 og niðurstaðan geymd í rdx
- $leaq(\%rdi, \%rdx, 2), \%rdx$ Hér er rdi bætt við tvöfaldaða rdx

  rdx er þá $i + 10j$
- $movqA(, \%rax, 8), \%rax$ Hleður gildi frá A með rax
- $imulqB(, \%rdx, 8), \%rax$ Margfaldar rax með gildi úr B með rdx

Niðurstaða

$$M = 10$$
$$N = 6$$

2

a)

Gefið er fylkið short int a[6][10]. Ef upphafsvistfang þess er 100 (tugatala), hvert er þá vistfang staksins a[3][4]?

Við vitum að short int eru 2 bæti og fylkið er geymt í línuröð svo minnisskekkjan er

$$\text{Offset} = (i * \text{númerdálka} + j) * \text{stærð staks}$$

$$\text{Offset} = ((3 * 10) + 4) * 2$$

bætum síðan við upphafsvistfangi og fáum þá 168

Vistfang staksins a[3][4] = 168

b)

Gefin er færslan:

```
struct abcd {
    short int a[2];
    double b;
    char c;
    char *d;
};
```

Rissum upp mynd af henni út frá upprööunarkröfu (alignment requirements) x86- 64. Og svörum síðan hversu mörg bæti tekur færslan?

Skoðum gildin í kóðanum:

- short int: 2 bæti
- double: 8 bæti
- char: 1 bæti
- char * : 8 bæti

Skoðum nú hvar gildin vistast

- short int a[2] Stærðin er 2 stykki af 2 bætum = 4 bæti

  Geymd frá offset 0 til 3
- Fylling 4 bæti þar sem næst verður 8 bæta krafa
- double b; stærð 8

  Vistast frá offset 8 til 15
- char c; Stærð 1 bæti

  vistast á offset 16
- fylling 7 bæti þar sem char *d er 8 bæti
- char * d stærð 8 bæti

  vistast frá offset 24- 31

Sýnum þetta myndrænt:

Offset	Gagn
$0-1$	a[0]
$2-3$	a[1]
$4-7$	**fylling**
$8-15$	b
16	c
$17-23$	**fylling**
$24-31$	d

færslan tekur **32** bæti

3

Við ætlum að skoða hvernig kanarífuglagildið virkar með því að keyra forritið `bufdemo.c` með `gcc` og rekja keyrslu þess í `gdb`.

Þýðum forritið:

```
gcc -Og -g -fstack-protector -o bufdemo bufdemo.c
```

keyrum forritið með gdb

```
gdb bufdemo
```

setjum brotpunkt

```
(gdb) break echo
```

Keyrum síðan forritið:

```
(gdb) run
```

og þá lendum við á brotpunktinum echo setjum break á enda keyrslu echo og gerum continue þá fáum við

```
Type a string:
```

Ég setti strenginn abcdefghijklmnopqrstuvwxyz til að stuðla að yfirflæði

```
(gdb) disas echo
```

Til að skoða kanarífuglagildið í skrá `%rax`, notum við eftirfarandi skipun:

```
(gdb) p/x $rax
```

Þetta sýnir gildi kanarífuglsins í `%rax` í hexadecimal formi:

```
$1 = 0x5
```

Hættum í gdb til að endurtaka:

```
(gdb) quit
```

og fáum:

```
$1 = 0x5
```

ég átti í erfiðleikum með að fá eitthverja breytingu á gildum með sama streng en ef stun–implaði inn strenginn: abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 þá fékk ég segmentation fault og rax gildið var 0x3f.

4

Við fáum gefið fallið `addto_diag`

```
void addto_diag(long *A, long n, long val) {
long i;
for (i=0; i<n; i++)
A[i*n + i] += val;
}

addto_diag:
testq %rsi, %rsi
jle .L1
leaq 8(,%rsi,8), %rcx
movl $0, %eax
.L3:
addq %rdx, (%rdi)
addq $1, %rax
addq %rcx, %rdi
cmpq %rax, %rsi
jne .L3
.L1:
ret
```

a)

- **%rdi**: Bendir á upphafsstað fylkisins `A`. Fylkið er geymt sem einvítt í smalamálinu. Gistið bendir á hornalínustakið `A[i*n + i]` í hverri umferð lykkjunnar.
- **%rsi**: Inniheldur gildið `n`, sem er fjöldi lína og dálka í tvívíða fylkinu `A`. Þetta gildi stjórnar því hve margar umferðir lykkjan keyrir.
- **%rdx**: Inniheldur gildið `val`, sem er gildið sem á að leggja við hvert hornalínustak `A[i*n + i]`. Þetta gisti er notað í `addq` skipuninni til að bæta `val` við hornalínustökin.
- **%rax**: Notað sem teljari fyrir lykkjuna. Í upphafi er það núll og er aukið í hverri umferð lykkjunnar þar til það nær gildinu `n`. Það er borið saman við `%rsi` til að ákvarða hvenær lykkjan á að enda.
- **%rcx**: Inniheldur forskotið sem þarf til að færa bendilinn frá einu hornalínustaki til þess næsta. Þetta forskot er reiknað sem `(n * 8) + 8`, þar sem hvert stak er `long` (8 bæti) og forskotið milli hornalínustaka er því `n + 1` stök í einvíða fylkinu.

b

Skipunin `leaq 8(,%rsi,8), %rcx` reiknar forskotið milli hornalínustaka í fylkinu. Þar sem fylkið er með stök af gerðinni `long` (sem er 8 bæti), reiknar skipunin út forskotið `(n * 8) + 8`, sem er forskotið frá `A[i*n + i]` til `A[(i+1)*n + (i+1)]`.

c

Í C er tvívítt fylki geymt sem einvítt í minnið í línuröð. Smalamálskóðinn vinnur beint með þessa minnisuppsetningu og því þarf ekki að margfalda til að reikna út staðsetningu fyrir hvert `A[i*n + i]`. Forskotið á milli hornalínustaka er reiknað einu sinni í upphafi með skipuninni `leaq` og síðan er það geymt í `%rcx` og notað í hverri umferð lykkjunnar.

5

a)

Við þýðum skrána `afun.c` fyrst með `-Og` og síðan með `-O3` og skoðum hvort summan er skrifuð út í minni inni í lykkjunni í þessum útgáfum.

þýðum með Og-

Skoðum lykkjuna í smalamálskóðanum:

```
.L3:
    movslq  %eax, %rcx
    movq    (%rdi,%rcx,8), %rcx
    addq    %rcx, (%rdx)
    addl    $1, %eax
.L2:
    cmpl    %esi, %eax
    jl      .L3
```

Sjáum að `movq (%rdi,%rcx,8), %rcx` les a[i] úr minni og setur í rcx

Skipunin `addq %rcx, (%rdx)` bætir a[i] við *res í minninu.

Í hverri ítrun er verið að uppfæra *res í minni, sem þýðir að symman er skrifuð út í minni inni í lykkjuni.

Þýðum með -O3

Skoðum lykkjuna í smalamálskóðanum:

```
.L3:
    addq    (%rdi), %rax
    addq    $8, %rdi
    movq    %rax, (%rdx)
    cmpq    %rdi, %rcx
    jne     .L3
```

Við skoðum `addq (%rdi), %rax` það bætir *a við summuna rax

`movq %rax, (%rdx)` skrifar núverandi summu í *res **Niðurstaða fyrir a):**

Í báðum tilvikum er summan skrifuð út í minni inni í lykkjunni.

b)

Til að láta þýðandann safna summunni upp í gisti og sleppa minnisaðganginum, getum við breytt fallinu með því að nota staðværa breytu fyrir summuna.

Breytt fall:

```
void fun(long *a, int len, long *res) {
    int i;
    long sum = 0;

    for (i = 0; i < len; i++) {
        sum += a[i];
    }

    *res = sum;
}
```

þýðum með -Og

Skoðum lykkjuna í smalamálskóðanum:

```
fun:
.LFB0:
    .cfi_startproc
    endbr64
    movl $0, %ecx
    movl $0, %eax
    jmp .L2
.L3:
    movslq %eax, %r8
    addq (%rdi,%r8,8), %rcx
    addl $1, %eax
.L2:
    cmpl %esi, %eax
    jl .L3
    movq %rcx, (%rdx)
    ret
    .cfi_endproc
```

Með þessari breytingu hefur þýðandinn nú safnað summunni upp í gistinu rcx og sleppt minnisaðganginum að *res inni í lykkjunni