Bootloader

Programowanie Niskopoziomowe

Szymon Telega

Plan prezentacji

- Inline assembly w języku C
- Kompilacja do postaci bootloadera
- QEMU

Inline assembly w języku C

- Motywacja - kiedy się przydaje
- Zalety i wady takiego podejścia
- Kompilatory
- Składnia

Do czego się przydaje

- Dostęp do wywołań systemowych i przerwań
- Dostęp do instrukcji procesora
- Dostarczenie odpowiednich dyrektyw do asemblera lub linkera
- Optymalizacja

Co nam utrudnia

- Utrudnienie analizy kodu przez kompilator
- Konserwacja oprogramowania
- Problem z przeniesieniem na inną platformę sprzętową

Zalety względem języka asemblera

- Prostsza składnia
- Łatwiejsza implementacja bardziej złożonych programów
- Łatwiejsze przeniesienie na inną platformę

Wady względem języka asemblera

- Wydajność programu
- Problem kompilacji

Kompilatory

- Microsoft Visual C++
- GCC

Microsoft Visual C++

- Intel syntax
- Głównie Windows i DOS

GCC

- AT&T syntax
- Głównie Unix

Porównanie składni AT&T i Intel - prefiksy

	Intel	AT&T
Rejestry	Brak eax	% %eax
Stałe	Brak 16	$ $16
Liczby szesnastkowe	0 , h sufiks 0ffh 80h	$0x $0xff $0x80
Liczby binarne	Brak, b jako sufiks 101b	Brak

Porównanie składni AT&T i Intel – kolejność wyrażeń

Intel:

instr <dest> <src>

AT&T:

instr <src> <dest>

Porównanie składni AT&T i Intel – rozmiar parametru

	Intel	AT&T
byte – 8bit	mov al, 4	movb $4, %al
word - 16bit	add ax, 2	addw $2, %ax
double word/long - 32bit	sub eax, ebx	subl %ebx, %eax

Składnia inline assembly w GCC

Słowo kluczowe asm/__asm__

```
asm(„movb $'H' , %al\n”);

__asm__(„movb $'H' , %al\n”);
```

Składnia inline assembly w GCC

Wiele instrukcji w jednej funkcji:

```
asm( ”movl $10, %eax;”
     „movl $20, %ebx;”
     „subl %ebx, %eax;”
   );
```

Składnia inline assembly w GCC

Rozszerzona wersja inline assembly

```
asm ( "assembly code"
        : „=r”(arg1), „=r”(arg2)      // output operands
        : „r”(arg2), „r”(arg3), ...   // input operands
        : „param1”, „param2”, ...     //list of clobbered registers
);
```

Przykład

Składnia inline assembly w GCC

Tabela rejestrów:

znak	Rejestry		
a	%eax	%ax	%al
b	%ebx	%bx	%bl
c	%ecx	%cx	%cl
d	%edx	%dx	%dl
S	%esi	%si	
D	%edi	%di	
r	dowolny		

Przykład

Składnia inline assembly w GCC

Clobbered registers list

- Użyte rejestry – przekazane w postaci stringów
- „cc” - kod asemblerowy zmienia rejestr flag
- „memory” - mówi kompilatorowi, że kod nadapisuje/czyta rejestry spoza listy input/output registers

Składnia inline assembly w GCC

Słowo kluczowe *volatile – ochrona przed błędną optymalizacją*

volatile jest domyślne przy podstawowej wersji asm()

```
asm volatile(„...”);

__asm__ __volatile__(„...”);
```

Składnia inline assembly w GCC

Słowo kluczowe inline – *ustawia rozmiar wyrażenia asm na najmniejszy możliwy*

```
asm inline(„...”);

__asm__ __inline__(„...”);
```

Składnia inline assembly w GCC

ASM labels

```
int name asm("param") = 5;

void foo()
    { asm("mov param, %eax"); }
```

Składnia inline assembly w GCC

ASM labels – to samo, ale bez specyfikacji aliasu

```
int name = 5;

void foo()
    { asm("mov name, %eax"); }
```

Kompilacja do postaci bootloadera

- Najważniejsze informacje o bootloaderach
- Język asemblera
- Inline assembly w C

Najważniejsze informacje dotyczące bootloaderów

- Rozmiar programu – 512 bajtów
- Boot signature - 2 ostatnie bajty mają wartość odpowiednio 0x55 i 0xAA
- Miejsce programu w pamięci – 0x7c0:0x0000 (0x0000:0x7c00)
- Real Mode – 16 bitowy tryb pracy

Bootloader w języku assemblera

```
; boot.asm
hang:
    jmp hang

        times 510-($-$$) db 0;
    db 0x55
    db 0xAA
```

przykład

Bootloader w C

- 16 bit, 32 bit, a 64 bit w Real Mode - .code16gcc
- Kompilacja i linkowanie GCC
- Biblioteka standardowa

Bootloader w C

- Kompilacja do pliku objektowego
- Linkowanie
- Przekształcenie do bliku binarnego
- Uruchomienie programu przy pomocy QEMU

Kompilacja

```
gcc -c -m16 -Os -ffreestanding -Wall -Werror -o boot.o
boot.c
```

- -Os – optymalizacje pod względem długości kodu wynikowego
- -ffreestanding – zabrania używania biblioteki standardowej
- -m16 – tworzenie 16–bitowego kodu wynikowego (.code16gcc)

Linkowanie

ld -m elf_i386 -static -nostdlib -Tlinker.ld --nmagic -o boot.elf boot.o

- -static – zabrania linkowania bibliotek współdzielonych
- -nostdlib – nie pozwala używać standardowych funkcji startupowych
- --nmagic - pozwala linkerowi na wygenerowanie kodu bez _start_SECTION i _stop_SECTION
- -m elf_i386 – określenie skryptu linkera (32-bit i386 binaries)

Linker

- Określenie miejsca rozpoczęcia programu
- Dopisanie boot signature

linker.ld

Przekształcanie do pliku binarnego

```
objcopy -O binary boot.elf boot.bin
```

- objcopy – pozwala na konwertowanie plików
- -O binary – konwertuje plik na plik binarny

QEMU

- Informacje ogólne
- Tryby pracy
- Funkcje

Informacje ogólne

- Obsługa architektur x86, x86-64 i wielu innych
- Możliwość uruchomienia na wielu platformach
- Możliwość zapisywania i wznawiania stanu maszyny
- Możliwość uruchomienia kilku systemów operacyjnych na jednej maszynie
- Szybkość
- Brak obsługi mniej popularnych platform sprzętowych
- Stosunkowo trudna obsługa

Tryby pracy

- Tryb systemu – emulowany jest cały system, łącznie z urządzeniami peryferyjnymi.
- Tryb użytkownika – może uruchamiać procesy systemu Linux skompilowane na innym typie procesora niż bieżący

Emulowane urządzenia

- CD/DVD-ROM
- Stacja dyskietek
- PC speaker
- Karta graficzna/sieciowa
- I wiele innych

Uruchamianie

Qemu-system-i386 -no-fd-bootchk -fda floppy.img

- qemu-system-i386/qemu-system-x86-64 – uruchomienie emulatora w odpowiedniej architekturze
- -fda file – flaga mówiąca, że *file* będzie obrazem dyskietki
- -no-fd-bootchk – flaga mówiąca BIOSowi, żeby nie sprawdzał *boot signature* dyskietki

Przykład końcowy

Przykład

Skrypt

Bibliografia

- https://wiki.osdev.org
- https://gcc.gnu.org
- https://en.wikipedia.org
- https://pl.wikipedia.org
- https://www.codeproject.com
- https://stackoverflow.com