Moduł 4

Programowanie hybrydowe

Plan wykładu

- Definicja i zastosowanie programowania hybrydowego
- ABI i konwencja wołania
- Konwencja wołania MIPS
- Konwencja wołania Unix System V dla x86
- Konwencja wołania Windows x64 (AMD64)
- Konwencja wołania Unix System V dla x86-64

Programowanie hybrydowe

- Tworzenie programu, którego poszczególne części (moduły) są pisane w różnych językach programowania
- Zastosowania:
  - używanie gotowych modułów ze starszych programów w nowym oprogramowaniu, pisanym w innym języku
  - realizacja w języku niższego poziomu operacji niedostępnych lub nieefektywnych w języku wysokiego poziomu, np. języki 4. generacji łączone z C, C łączony z asemblerem
  - korzystanie z modułów bibliotecznych

Zasady programowania hybrydowego

- Moduł wywołujący (w tym program główny) określa wymagania na zachowanie modułów przezeń wywoływanych
  - procedury wywoływane nie mogą powodować zaburzeń w pracy procedur wywołujących
  - muszą one być zgodne z określonymi w danym środowisku regułami zachowania oprogramowania (tzw. ABI), w tym z konwencją wołania
- Zazwyczaj kod w języku wysokiego poziomu wywołuje kod pisany w języku podobnego lub niższego poziomu
  - nie ma przeciwwskazań, by było odwrotnie, ale konstrukcje takie są niezbyt uzasadnione

Konwencja wołania

- Określa sposób przekazywania argumentów, wywoływania procedur i odbierania wartości oraz zasady korzystania procedur z zasobów procesora i systemu
- Może być zdefiniowana przez:
  - projektantów procesora – dla wszystkich środowisk programowych dla danego procesora (np. MIPS)
  - projektantów systemu operacyjnego – dla twórców oprogramowania dla danego systemu (np. Unix/Linux dla x86, Linux i Windows dla AMD64)
  - projektantów kompilatora – dla programistów tworzących programy, których przynajmniej niektóre moduły są kompilowane danym kompilatorem (np. kompilatory dla systemu DOS)
- Im większy zasięg stosowalności konwencji, tym łatwiej jest tworzyć przenośne oprogramowanie hybrydowe
    - może być nawet różna dla różnych języków programowania w jednym środowisku!

Funkcje rejestrów w konwencji wołania

- Wskaźnik stosu
  - określa położenie wierzchołka stosu
- Wskaźnik ramki
  - zawiera adres, względem którego procedura adresuje swoją ramkę stosu
  - opcjonalny – niekiedy nie używany
- Rejestry argumentów
  - służą do przekazywania argumentów do wywoływanej procedury
  - argumenty nie mieszczące się w rejestrach lub o typach “nie pasujących” do rejestrów są przekazywane przez stos w pamięci
- Rejestry wartości
  - służą do zwracania skalarnych wartości funkcji
  - wartości strukturalne oraz typów przekraczających pojemność rejestrów są zwracane na stosie, w miejscu zaalokowanym przez procedurę wołającą

Grupy rejestrów w konwencji wołania

- Rejestry zachowywane (grupa "*saved*")
  - wartości muszą być zachowane przez procedurę
  - procedura albo nie używa ich wcale albo zachowuje na stosie, używa, a przed powrotem odtwarza ze stosu
  - służą do alokacji zmiennych lokalnych
  - do tej grupy należy wskaźnik ramki
- Rejestry tymczasowe (grupa "*temporary*")
  - mogą być dowolnie używane przez każdą procedurę
  - procedura wołana może niszczyć ich zawartość
  - zastosowanie:
    - tymczasowe wyniki pośrednie obliczeń i adresy
    - zmienne lokalne w procedurach nie wołających innych procedur
  - do tej grupy zwykle należą rejestry argumentów i wartości

Prolog procedury

- Wyrównanie stosu (opcjonalne)
- Zachowanie wskaźnika ramki (jeśli potrzebne)
- Alokacja miejsca na rejestry argumentów (zależnie od konwencji wołania)
- Składowanie na stosie rejestrów argumentów (w razie potrzeby)
- Ustanowienie nowej wartości wskaźnika ramki
- Alokacja miejsca na zmienne lokalne
- Składowanie rejestrów zachowywanych (jeśli potrzebne)

Epilog procedury

- Odtworzenie rejestrów zachowywanych
- Dealokacja zmiennych lokalnych
- Odtworzenie wskaźnika ramki procedury wołającej
- (W razie potrzeby) dealokacja innych danych na stosie (np. lokalnych kopii argumentów)
- Powrót według śladu

Konwencja wołania MIPS

- Zdefiniowana przez projektantów procesora, obowiązuje we wszystkich systemach operacyjnych
- Nadaje nazwy mnemoniczne rejestrom procesora
  - nazwy odzwierciedlają rolę rejestru w konwencji wołania
- Część argumentów przekazywana przez rejestry
- Do 9 danych lokalnych w rejestrach – grupa *saved*
- MIPS nie ma sprzętowej obsługi stosu
  - konwencja definiuje stos i zasady jego użycia

Konwencja wołania MIPS - rejestry

Numer	Symbol	Funkcja
$0	$zero	odczyt – stałe 0, zapis – wartość tracona
$1	$at	używany przez asembler do rozwijania metainstrukcji
$2, $3	$v0, $v1	rejestry wartości funkcji
$4..$7	$a0..$a3	rejestry argumentów funkcji
$8..$15	$t0..$t7	rejestry tymczasowe, nie zachowywane
$16..$23	$s0..$s7	rejestry dla zmiennych lokalnych, zachowywane
$24, $25	$t8, $t9	rejestry tymczasowe, nie zachowywane
$26, $27	$k0, $k1	do użytku systemu operacyjnego – ulotne
$28	$gp	wskaźnik danych statycznych
$29	$sp	wskaźnik stosu
$30	$fp lub $s8	wskaźnik ramki lub rejestr zachowywany
$31	$ra	ślad powrotu

MIPS – użycie rejestrów

- rejestr at
  - używany przez asembler do rozwijania metainstrukcji asemblera, jako rejestr tymczasowy przechowujący wyniki pośrednie
- rejestry k
  - używane przez system operacyjny do obsługi wyjątków. Dla programu użytkowego są one ulotne – ich zawartość może w każdej chwili zostać zamazana
- rejestr gp
  - przy starcie aplikacji inicjowany wartością równą adresowi początkowemu sekcji danych statycznych + 32K
  - umożliwia szybkie adresowanie 64 KB danych statycznych rzy użyciu przemieszczeń w zakresie -32K..32K
- rejestr fp
  - zwykle nie używany przez kompilatory jako wskaźnik ramki, może być użyty jako 9-ty rejestr zachowywany

Konwencja wołania Unix System V dla x86

- Zdefiniowana przez projektantów systemu Unix System V
- Opisana w dokumencie “Unix System V Application Binary Interface Intel386™ Architecture Processor Supplement”, dostępnym w www.sco.com
- Używana w systemach rodziny Unix i Linux
- Zgodna w podstawowych aspektach z konwencją wołania używaną przez kompilatory języka C w systemie Windows (32-bitowym)
  - umożliwia to częściową przenośność modułów pisanych w asemblerze i języku C w postaci źródłowej i pośredniej

Konwencja wołania x86 - rejestry

Rejestr	Zastosowanie	Zachowywany
EAX	rejestr wartości	nie
ECX	rejestr roboczy	nie
EDX	rejestr wartości (jeśli dłuższy niż 32 bity)	nie
EBX	rejestr roboczy lub zmienna lokalna	tak
ESI	rejestr roboczy lub zmienna lokalna	tak
EDI	rejestr roboczy lub zmienna lokalna	tak
ESP	wskaźnik stosu	(tak)
EBP	wskaźnik ramki ew. roboczy/zmienna lokalna	(tak)

x86 - prolog i epilog procedury

```
myproc:
; prolog - stały dla wszystkich procedur
        push    ebp         ; zapamiętanie wskaźnika ramki procedury wołającej
        mov     ebp, esp ; ustanowienie własnego wskaźnika ramki
        sub     esp, (ROZMIAR_DANYCH_LOKALNYCH + 3) & ~3   ; alokacja danych lokaln

; zapamiętanie rejestrów zachowywanych (o ile są używane)
        push    ebx
        push    esi
        push    edi

; ciało procedury
        ; ...

; odtworzenie rejestrów, które były zapamiętane
        pop     edi
        pop     esi
        pop     ebx

; epilog - stały dla wszystkich procedur
        mov     esp, ebp ; dealokacja danych lokalnych
        pop     ebp         ; odtworzenie wskaźnika ramki procedury wołającej
        ret                 ; powrót
```

x86 - przykład

```
; odpowiednik funkcji strlen z biblioteki standardowej C
; deklaracja na poziomie C: unsigned mystrlen(char *s)
        section  .text
        global   mystrlen
mystrlen:
; prolog
        push     ebp      ; zapamiętanie wskaźnika ramki procedury wołającej
        mov      ebp, esp ; ustanowienie własnego wskaźnika ramki
; procedura nie alokuje danych lokalnych na stosie

; ciało procedury
        mov      eax, [ebp+8]     ; argument – wskaźnik na łańcuch
lop1:
        mov      dl, [eax]        ; kolejny bajt łańcucha
        inc      eax              ; inkrementacja adresu
        test     dl, dl           ; test czy bajt = 0
        jnz      lop1             ; nie – następny bajt
        dec      eax              ; cofnięcie wskaźnika o 1
        sub      eax, [ebp+8]     ; odjęcie adresu początku łańcucha
        ; wartość funkcji w EAX
; epilog
        ;mov     esp, ebp ; dealokacja danych lokalnych - zbędna
        pop      ebp      ; odtworzenie wskaźnika ramki procedury wołającej
        ret               ; powrót
```

Konwencja wołania Windows x64 dla AMD64

- Zdefiniowana przez Microsoft, obowiązuje w systemach rodziny Windows dla procesorów o modelu programowym zgodnym z architekturą AMD64
- Różna od konwencji wołania używanej w systemach Unix/Linux dla tych samych procesorów (!!!)
- Podobna do konwencji dla procesorów RISC
  - używa rejestrów do przekazywania argumentów
- Wymusza wyrównanie stosu do 16 lub 32 bajtów
  - ma to związek z wymaganiami jednostki wektorowej na wyrównanie danych

Konwencja wołania Windows x64 - rejestry

- 16 rejestrów uniwersalnych, w tym
  - RSP – wskaźnik stosu
  - RBP – wskaźnik ramki
- 16 rejestrów zmiennopozycyjnych i wektorowych jednostki SSE/AVX
- Oprogramowanie nie korzysta z dostępnej w procesorach starszej jednostki zmiennopozycyjnej i wektorowej x87/MMX

Konwencja wołania Windows x64 - rejestry

Rejestr	Zastosowanie	Zachowywane
RAX	rejestr wartości	nie
RCX	pierwszy argument	nie
RDX	drugi argument	nie
R8	trzeci argument	nie
R9	czwarty argument	nie
R10, R11	rejestry tymczasowe	nie
R12..R15	rejestry tymczasowe lub zmienne lokalne	tak
RDI, RSI, RBX	rejestry tymczasowe lub zmienne lokalne	tak
RBP	wskaźnik ramki ew. roboczy/zmienna lokalna	(tak)
RSP	wskaźnik stosu	(tak)
XMM0	pierwszy argument zmiennopozycyjny	nie
XMM1	drugi argument zmiennopozycyjny	nie
XMM2	trzeci argument zmiennopozycyjny	nie
XMM3	czwarty argument zmiennopozycyjny	nie
XMM4, XMM5	rejestry tymczasowe	nie
XMM6..XMM15	rejestry tymczasowe lub zmienne lokalne	tak

Konwencja wołania x86-64 ABI

- Konwencja opisana w dokumencie *System V Application Binary Interface AMD64 Architecture Processor Supplement* dostępnym w www.x86-64.org
- adresy 64-bitowe
- obiekty na stosie – min. 64 bity (również 128 i 256 – dane wektorowe) – wyrównane naturalnie
- część argumentów przekazywana przez rejestry
- stos wyrównany do granicy 16 bajtów (wkrótce do 32 bajtów)
  - wymaganie wynika z rozmiaru danych jednostki SSE (a wkrótce AVX), które muszą być wyrównane naturalnie

Konwencja wołania x64-64 ABI - rejestry

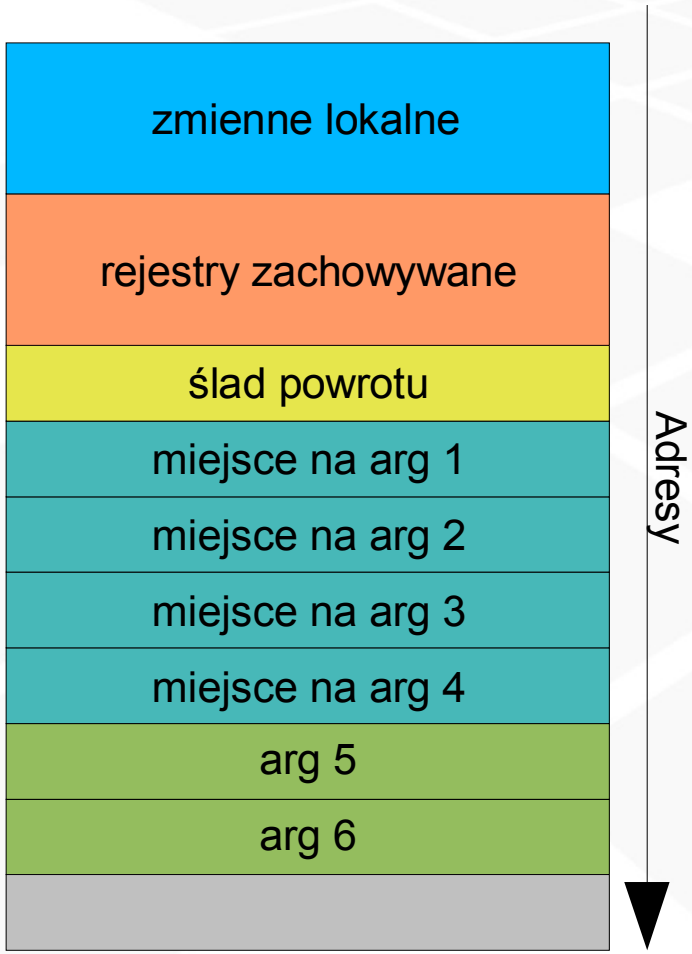
Rejestr	Zastosowanie	Zachowywane
RAX	rejestr wartości	nie
RDI, RSI	pierwszy i drugi argument	nie
RDX	trzeci argument, drugi rejestr wartości	nie
RCX	czwarty argument	nie
R8, R9	piąty i szóstyargument	nie
R10, R11	rejestry tymczasowe	nie
R12..R15	rejestry tymczasowe lub zmienne lokalne	tak
RBX	rejestry tymczasowe lub zmienne lokalne	tak
RBP	wskaźnik ramki ew. roboczy/zmienna lokalna	(tak)
RSP	wskaźnik stosu	(tak)
XMM0	pierwszy argument zmiennopozycyjny	nie
XMM1	drugi argument zmiennopozycyjny	nie
XMM2	trzeci argument zmiennopozycyjny	nie
XMM3	czwarty argument zmiennopozycyjny	nie
XMM4, XMM5	rejestry tymczasowe	nie
XMM6..XMM15	rejestry tymczasowe lub zmienne lokalne	tak

Konwencja wołania x64-64 ABI – ramka stosu

- Pierwszy argument przekazywany przez stos wyrównany (czyli ostatni wkładany na stos) wyrównany do 16(32) bajtów
  - argumenty zajmują kolejne lokacje stosu, więc o wyrównanie trzeba zadbać przed umieszczeniem na stosie pierwszego argumentu
  - Instrukcja CALL jest wykonywana przy takim wyrównaniu stosu
- Po zapamiętaniu na stosie wskaźnika ramki wierzchołek stosu jest wyrównany do 16 bajtów
- Procedura-liść może korzystać z obszaru 128 poniżej wartości RSP
  - nie ma obowiązku jawnej alokacji miejsca na stosie poprzez przesunięcie wskaźnika stosu.

E-STUDIA INFORMATYCZNE

Ramka stosu – Windows, AMD64



- pierwsze cztery argumenty skalarne w rejestrach
  - na stosie procedura wołająca rezerwuje miejsce na ich przechowanie przez procedurę wołaną
- przed CALL stos wyrównany do 16 bajtów
- zmienne lokalne alokowane po zachowaniu rejestrów
- wskaźnik ramki wskazuje początek (najmniejszy adres) zmiennych lokalnych

Kompilacja programu hybrydowego – x86, Linux

- Pliki źródłowe:
  - glowny.c – moduł w języku C
  - drugi.s – moduł w asemblerze x86
- Kompilacja modułu w C

```
cc -m32 -c glowny.c
```

  - opcja -m32 niezbędna do kompilacji w trybie 32-bitowym przy pracy w systemie 64-bitowym
  - Tworzy plik `glowny.o`
- Asemblacja – asembler NASM

```
nasm -f elf32 drugi.s
```

  - Tworzy plik `drugi.o`
- Konsolidacja

```
cc -m32 -o glowny glowny.o drugi.o
```

Kompilacja programu hybrydowego – x86-64, Linux

- Pliki źródłowe:
  - glowny.c – moduł w języku C
  - drugi.s – moduł w asemblerze x86
- Kompilacja modułu w C
  ```
  cc -c glowny.c
  ```
  - Tworzy plik `glowny.o`
- Asemblacja – asembler NASM
  ```
  nasm -f elf64 drugi.s
  ```
  - Tworzy plik `drugi.o`
- Konsolidacja
  ```
  cc -o glowny glowny.o drugi.o
  ```

Kompilacja programu hybrydowego – x86, Windows, kompilator MSC

- Pliki źródłowe:
  - `glowny.c` – moduł w języku C
  - `drugi.asm` – moduł w asemblerze x86
- Polecenia cl wydawane z konsoli, po wcześniejszym ustawieniu środowiska (Visual Studio Tools ->x86 Command Line)
- Kompilacja modułu w C
  ```
  cl -c glowny.c
  ```
- Asemblacja
  - asembler NASM (z konsoli nasm-shell)
    ```
    nasm -f win32 drugi.asm
    ```
  - asembler MASM
    ```
    ml -c -coff drugi.asm
    ```
- Konsolidacja
  ```
  cl glowny.obj drugi.obj
  ```

Asemblerowa funkcja main w środowisku języka C

- Funkcja main może być napisana w asemblerze (jako eksperyment – użyteczność takiego rozwiązania jest znikoma)
- Może ona korzystać z funkcji biblioteki standardowej C
- Funkcja musi być zgodna z konwencją wołania, gdyż jest wołana zgodnie z tą konwencją i może sama wywoływać inne funkcje napisane w C

main() w asemblerze - x86

```
; asemblacja:           nasm -f elf32 cmain32.s
; konsolidacja:         cc -m32 -o cmain32 cmain32.o

        section .data
msg:    db '32-bit assembly main',10, 0

        section .text
        global main     ;
        extern puts
main:
        push    msg
        call    puts
        add     esp, 4
        xor     eax, eax
        ret
```

main() w asemblerze – x86-64, Linux

```
; asemblacja:           nasm -f elf64 cmain64.s
; konsolidacja:         cc -o cmain64 cmain64.o

        section .data
msg:    db '64-bit assembly main',10, 0

        section .text
        global main    ;
        extern puts
main:
        mov     rdi, msg
        call    puts
        xor     eax, eax
        ret
```

X64, Linux - prolog i epilog procedury

Przykład – x86-64, Linux

```
; odpowiednik funkcji strlen z biblioteki standardowej C
; deklaracja na poziomie C: unsigned long mystrlen(char *s)
        section  .text
        global   mystrlen
mystrlen:
; prolog - pusty; argument w rejestrze RDI
; procedura nie alokuje danych lokalnych na stosie

; ciało procedury
        mov     rax, rdi          ; argument - wskaźnik na łańcuch
lop1:
        mov     dl, [rax]         ; kolejny bajt łańcucha
        inc     rax               ; inkrementacja adresu
        test    dl, dl            ; test czy bajt = 0
        jnz     lop1              ; nie - następny bajt
        dec     rax               ; cofnięcie wskaźnika o 1
        sub     rax, rdi ; odjęcie adresu początku łańcucha
        ; wartość funkcji w RAX
; epilog
        ret                 ; powrót
```

ARM - rejestry

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (SP)
r14 (LR)
r15 (PC)

CPSR

- 16 rejestrów r0..15
  - r0..12 – całkowicie uniwersalne
  - Ograniczona dostępność r8..r12 dla instrukcji Thumb
  - r13 (SP) – wskaźnik stosu
  - r14 (LR) – rejestr śladu
  - r15 (PC) – licznik instrukcji
- rejestr stanu CPSR
  - zawiera bity znaczników N, Z, C, V oraz informacje systemowe

Konwencja wołania ARM

- Zdefiniowana dla architektury ARM – niezależna od środowiska programowego.
- Pierwsze 4 argumenty przekazywane przez rejestry r0..3
- r0..1 służą również do zwracania wartości
- r4..r11 muszą być zachowane przez procedurę
- skok zostawia ślad w rejestrze LR
- podczas przekazywania sterowania stos wyrównany do 8

ARM – rejestry w konwencji wołania

Rejestr	Nazwa	Zastosowanie	Zachowywane
r0	a1	pierwszy argument, wartość funkcji	nie
r1	a2	drugi argument, wartość funkcji	nie
r2	a3	trzeci argument, wartość funkcji	nie
r3	a4	czwarty argument, wartość funkcji	nie
r4..8	v1..5	tymczasowe lub zmienne lokalne	tak
r9	v6/SB/TR	j.w. ew. baza statyczna	tak
r10, r11	v7, v8	tymczasowe lub zmienne lokalne	tak
r12	IP	rejestr tymczasowy	nie
r13	SP	wskaźnik stosu	(tak)
r14	LR	rejestr śladu	(tak)
r15	PC	licznik instrukcji	(tak)

ARM – wybrane instrukcje

- LDR
  - Do ładowania długich stałych używa się instrukcji LDR z adresowaniem względem PC – stałe są umieszczane w sekcji kodu, za najbliższą instrukcją skoku bezwarunkowego (np. za powrotem z procedury).
- PUSH
  - Z powodu konieczności zachowania wyrównania stosu zwykle używa się PUSH z parzystą liczbą argumentów.
- POP
  - Jeśli jednym z argumentów jest PC, instrukcja ta może służyć jako powrót z procedury.

ARM – prolog i epilog