# Architektura komputerów

## Wykład 10

Jan Kazimirski

# Programowanie w assemblerze x86 c.d.

# Funkcje systemowe

- System operacyjny UNIX udostępnia programom usługi za pomocą funkcji systemowych (syscall).
- Liczba funkcji systemowych jest bardzo duża (1000+), jednak zwykle używany jest niewielki podzbiór.
- Systemy POSIX-owe udostępniają zbliżoną funkcjonalność i interfejs API, chociaż szczegóły mogą się różnić.

# Funkcje systemowe c.d.

- Sposób wywołania funkcji systemowej:
  - Poprzez wywołanie funkcji z biblioteki C
    - Pozwala zachować kompatybilność w przypadku zmiany numeru funkcji lub formatu wywołania.
  - Bezpośrednie wywołanie poprzez przerwanie programowe 0x80
    - Efektywne – omija kod C
    - Numer funkcji przekazywany jest w rejestrze EAX

# Funkcje systemowe c.d.

- Przekazywanie argumentów do funkcji systemowej:
  - Poprzez stos – metoda „Uniksowa”
    - Argumenty umieszczane są na stosie
  - Poprzez rejestry – stosowana w Linuksie
    - Szybkie – nie korzysta ze stosu
    - Argumenty przekazywane są w rejestrach EBX, ECX, EDX, ESI, EDI, EBP
    - Większa liczba argumentów przekazywana jest w postaci struktury (adres jako argument)

# Funkcja systemowa sys_exit

- Zakończenie wykonywanego aktualnie procesu
- Numer funkcji: 1
- Argumenty:
  - EBX: kod powrotu z programu
- Wartość zwracana: brak

# Dokumentacja funkcji systemowych

- Lista nazw funkcji systemowych
  *man 2 syscalls*
- Numery funkcji: /usr/include/asm-x86/unistd.h
- Dokumentacja poszczególnych funkcji:
  - Większość funkcji systemowych ma odpowiadające im funkcje w bibliotece standardowej libc
  - Dokumentacja funkcji bibliotecznych libc dostępna jest jako strony manuala systemowego (man)

# Dokumentacja funkcji systemowych c.d.

- Przykład: funkcja **sys_exit**
- Dokumentacja: *man 2 exit*
- Wywołanie: **void _exit(int status)**
  - *Funkcja ma jeden argument (EBX)*
  - *Funkcja nie zwraca wartości*

# [1] Program pole kwadratu

```
;; PROGRAM EX01
;; Calculate the area of a square
segment .data
side   db 8          ; side length

segment .text
        global  _start

_start:
    mov al,[side]    ; load side length to al
    mul al           ; calculate square
    mov bx,ax        ; result to bx

    mov eax,1        ; syscall code: sys_exit
    int 0x80         ; do syscall
```

Kompilacja i wykonanie:

```
nasm -f elf64 -g ex01.asm
ld ex01.o
./a.out
echo $?
64              <- wynik
```

# Podstawowe wejście i wyjście

- Aby utworzyć jakikolwiek kompletny nietrywialny program trzeba mieć możliwość wprowadzania danych z klawiatury i wyprowadzania wyników na ekran.
- Można w tym celu zastosować funkcje systemowe działające na strumieniach, oraz standardowe strumienie stdin (wejście) i stdout (wyjście).

# Funkcja systemowa: sys_write

- Zapis do pliku o podanym deskryptorze
- Numer: 4
- Składnia: ssize_t write(int fd, const void *buf, size_t count);
- Uwagi:
  - Deskryptor o numerze 1 to standardowy strumień wyjściowy.

# Funkcja systemowa: sys_read

- Odczyt z pliku o podanym deskryptorze
- Numer: 3
- Składnia: ssize_t read(int fd, void *buf, size_t count);
- Uwagi:
  - Deskryptor o numerze 0 to standardowy strumień wejściowy.

# [2] Program echo

```
;; PROGRAM EX02
;; copy stdin to stdout
segment .bss
buff   resb  120    ; line buffer
segment .text
  global _start
_start:
  mov ecx,buff      ; set buffer address
loop:
  mov eax,3         ; sys_read
  mov ebx,0         ; stdin
  mov edx,120       ; set buffer size
  int 0x80          ; syscall - read line
  mov edx,eax       ; set actual line size
  mov eax,4         ; sys_write
  mov ebx,1         ; stdout
  int 0x80          ; syscall - write line
  jmp loop          ; start again
```

# [2] Program echo c.d.

```
;; PROGRAM EX02
;; copy stdin to stdout
segment .bss
buff   resb  120     ; line buffer
segment .text
  global _start
_start:
  mov ecx,buff       ; set buffer address
loop:
  mov eax,3          ; sys_read
  mov ebx,0          ; stdin
  mov edx,120        ; set buffer size
  int 0x80           ; syscall - read line
  mov edx,eax        ; set actual line size
  mov eax,4          ; sys_write
  mov ebx,1          ; stdout
  int 0x80           ; syscall - write line
  jmp loop           ; start again
```

Funkcja systemowa sys_read czyta z pliku o podanym deskryptorze (tu: stdin) linię (odczyt buforowany) i umieszcza ją w buforze.
Funkcja zwraca liczbę wczytanych znaków.

# [2] Program echo c.d.

```
;; PROGRAM EX02
;; copy stdin to stdout
segment .bss
buff   resb  120    ; line buffer
segment .text
  global _start
_start:
  mov ecx,buff      ; set buffer address
loop:
  mov eax,3         ; sys_read
  mov ebx,0         ; stdin
  mov edx,120       ; set buffer size
  int 0x80          ; syscall - read line
  mov edx,eax       ; set actual line size
  mov eax,4         ; sys_write
  mov ebx,1         ; stdout
  int 0x80          ; syscall - write line
  jmp loop          ; start again
```

Funkcja systemowa sys_write zapisuje do pliku o podanym deskryptorze (tu: stdout) określoną liczbę znaków z bufora.

# Wczytanie liczby całkowitej

- Nietrywialny program wymaga zwykle wprowadzania danych w postaci liczbowej.
- Założenia upraszczające
  - prosty format wejściowy: dwucyfrowa liczba w zakresie 00 .. 99.
  - brak kontroli błędów

# Wczytanie liczby całkowitej c.d.

- **Warunki początkowe:** W buforze wskazywanym przez ecx znajdują się dwie cyfry kodowane jako znaki ASCII
- **Algorytm**
  1. Załaduj 1-szą cyfrę z [ecx] do al
  2. Odejmij 48 od al (konwersja ASCII na int)
  3. Pomnóż ax przez 10 (przesunięcie dziesiętne w lewo)
  4. Załaduj 2-gą cyfrę z [ecx+1] do bl
  5. Odejmij 48 od bl (konwersja ASCII na int)
  6. Dodaj bx do ax.
- **Warunki końcowe**: W eax znajduje się zdekodowana liczba całkowita

# Wczytanie liczby całkowitej c.d.

- **Kod:**

```
        sub eax,eax         ; set eax=0
        mov al,[ecx]        ; read 1st digit
        sub al,48           ; convert from ASCII to int
        imul ax,10          ; shift left (decimal)
        sub ebx,ebx         ; set ebx=0
        mov bl,[ecx+1]      ; read 2nd digit
        sub bl,48           ; convert from ASCII to int
        add ax,bx           ; combine values of digits
```

# Wczytanie liczby całkowitej c.d.

- W przypadku większych programów często poszczególne zadania implementuje się jako osobne funkcje (podprogramy)
- W assemblerze jest to szczególnie istotne ze względu na złożoność kodu.
- Do wyodrębnienia fragmentu kodu jako podprogramu można użyć instrukcji CALL i RET.

# Procedura read_number c.d.

```
read_number:
     mov eax,3          ; sys_read
     mov ebx,0          ; stdin
     mov edx,3          ; set buffer size
     int 0x80           ; syscall - read line
     sub eax,eax        ; set eax=0
     mov al,[ecx]       ; read 1st digit
     sub al,48          ; convert from ASCII to int
     imul ax,10         ; shift left (decimal)
     sub ebx,ebx        ; set bx=0
     mov bl,[ecx+1]     ; read 2nd digit
     sub bl,48          ; convert from ASCII to int
     add ax,bx          ; combine values of digits
     ret                ; return to the caller
```

# read_number – program główny

```
segment .bss
buff   resb  10    ; line buffer

segment .text
  global _start

_start:
        mov ecx,buff      ; set buffer address
        call read_number ; read number

; Return decoded number
        mov ebx,eax       ; store number in ebx
        mov eax,1         ; syscall code: sys_exit
        int 0x80          ; do syscall
```

# Zmienne lokalne

- Procedura read_number zależy od zewnętrznego bufora w pamięci dostarczonego przez wołający kod.
- Problemy:
  - Programista korzystający z funkcji może zapomnieć o przydzieleniu miejsca na bufor.
  - Bufor na dane alokowany jest w sposób statyczny (globalnie) a używany tylko w czasie używania funkcji

# Zmienne lokalne c.d.

- Procedura read_number powinna korzystać z bufora allokowanego lokalnie tylko na czas działania procedury
- Rozwiązania:
  - Przydział stron pamięci przez system operacyjny – sys_mmap - **kosztowne i kłopotliwe!**
  - Wykorzystanie stosu

# read_number – wersja 2

```
read_number:
    push ebp            ; save original ebp
    mov ebp,esp         ; save stack pointer
    sub esp,16          ; move stack pointer - make "gap"
    lea ecx,[ebp-16]    ; set buffer to stack "gap"
    mov eax,3           ; sys_read
    mov ebx,0           ; stdin
    mov edx,10          ; set buffer size
    int 0x80            ; syscall - read line
    sub eax,eax         ; set eax=0
    mov al,[ecx]        ; read 1st digit
    sub al,48           ; convert from ASCII to int
    imul ax,10          ; shift left (decimal)
    sub ebx,ebx         ; set bx=0
    mov bl,[ecx+1]      ; read 2nd digit
    sub bl,48           ; convert from ASCII to int
    add ax,bx           ; combine values of digits
    mov esp,ebp         ; get original stack pointer
    pop ebp             ; restore original ebp
    ret                 ; return to the caller
```

# Wyświetlenie liczby całkowitej

- Podprogram wyświetlający liczbę całkowitą na ekran
- Założenia
  - Podprogram korzysta z lokalnego bufora alokowanego na stosie
- Uproszczenia
  - Zakres wyświetlanych liczb: 00 ..99
  - Brak kontroli błędów

# Wyświetlenie liczby całkowitej c.d.

- **Warunki początkowe:** Rejestr ax zawiera liczbę z przedziału 0 .. 99 do wyświetlenia
- **Działanie**: Na ekranie pojawia się liczba z rejestru ax
- **Algorytm**
  1. Wykonaj dzielenie ax przez 10. Wynik to liczba dziesiątek a reszta to liczba jedności
  2. Dodaj do wyniku i reszty 48 (konwersja na ASCII)
  3. Umieść skonwertowany wynik na pierwszej pozycji bufora
  4. Umieść skonwertowaną resztą na drugiej pozycji bufora
  5. Umieść znak nowej linii na trzeciej pozycji bufora
  6. Wywołaj funkcję systemową sys_write
- **Warunki końcowe**: Brak

# Wyświetlenie liczby całkowitej c.d.

```
write_number:
    push ebp                ; save original ebp
    mov ebp,esp             ; save stack pointer
    sub esp,16              ; move stack pointer - make "gap"
    lea ecx,[ebp-16]        ; set buffer to stack "gap"
    mov bl,10
    div bl                  ; divide by 10 (results in al,ah)
    add ah,48               ; convert remainder to ASCII
    add al,48               ; convert result to ASCII
    mov [ecx],al            ; put 10's digit to buffer
    mov [ecx+1],ah          ; put 1's digit to buffer
    mov [ecx+2],byte 10     ; put end of line to buffer
    mov eax,4               ; sys_write
    mov ebx,1               ; stdout
    mov edx,3               ; set buffer size
    int 0x80                ; syscall - write line
    mov esp,ebp             ; get original stack pointer
    pop ebp                 ; restore original ebp
    ret                     ; return to the caller
```

# Wyświetlenie komunikatu

- Podprogram wyświetla na ekran zadany komunikat
- Komunikat przechowywany jest w pamięci jako tekst formatu C (null-terminated)
- Problem: Przed wywołaniem odpowiedniej funkcji systemowej trzeba określić długość tekstu

# print_msg

```
; Input: ebx - address of the message to print
print_msg:
        mov edi,ebx         ; address to edi for scasb
        mov ecx,80          ; max length of the string
        mov al,0            ; look for a null (0) char
        cld                 ; clear direction flag (down)
        repne  scasb        ; search for al in [edi]
        mov eax,80
        sub eax,ecx         ; calculate string length
        dec eax             ; remove last char (null)

        mov ecx,ebx       ; set buffer address for syscall
        mov edx,eax       ; set message length for syscall
        mov eax,4         ; sys_write
        mov ebx,1         ; stdout
        int 0x80          ; syscall - write line

        ret               ; return to caller
```

# print_msg – program główny

```
segment .data
     msg db 'Komunikat specjalny.',10,0

segment .text
     global _start

_start:
     mov ebx,msg       ; put message address to ebx
     call print_msg    ; call subroutine

     mov eax,1         ; syscall code: sys_exit
     int 0x80          ; do syscall
```

# Program „pole kwadratu”

- **Działanie**: Program pyta użytkownika o długość boku kwadratu i wyświetla pole kwadratu o podanym boku. UWAGA! Program obsługuje liczby w zakresie 0...99.
- **Algorytm**
    1. Wyświetl komunikat – podanie boku
    2. Wczytaj wartość boku kwadratu
    3. Wylicz pole kwadratu
    4. Wyświetl komunikat o wyniku
    5. wyświetl wynik

# Program „pole kwadratu” - program główny

```
segment .data
  msg1 db 'Prosze podac bok kwadratu: ',0
  msg2 db 'Pole kwadratu wynosi: ',0
  msg3 db 'Dziekuje za skorzystanie z programu.',10,10,0

segment .text
  global _start
_start:
    mov ebx,msg1
    call print_msg      ; print msg1
    call read_number    ; read input number
    imul ax,ax          ; compute result
    push eax            ; save eax (result)
    mov ebx,msg2
    call print_msg      ; print msg2
    pop eax             ; restore result in eax
    call write_number   ; write result (ax)
    mov ebx,msg3
    call print_msg      ; print msg3
    mov eax,1         ; syscall code: sys_exit
    int 0x80          ; do syscall
```

# Podsumowanie

- Wykorzystanie funkcji systemowej
- Podstawowe funkcje wejścia i wyjścia
- Wykorzystanie podprogramów
- Zmienne lokalne
- Program „Pole kwadratu” w assemblerze