Università degli Studi di Milano
Corso di Laurea in Informatica, A.A. 2017-2018

System calls


Homepage del corso

Turno A
Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Turno B
Jacopo Essenziale
Dipartimento di Informatica
Via Celoria 20 - 20133 Milano (MI)
AISLab
jacopo.essenziale@unimi.it
+39 02.503.14010

System Calls

- **System call**: permette di utilizzare **servizi** la cui esecuzione è a carico del sistema operativo: tipicamente operazioni di input/output

- Ogni servizio è associato ad un codice numerico univoco (un intero)

- Come si utilizza una system call che ha codice **K**?
  - Caricare **K** nel registro `$v0`;
  - caricare gli argomenti (se necessari) nei registri `$a0, $a1, $a2, $a3, $f12` (opzionale);
  - eseguire l'istruzione `syscall`;
  - leggere eventuali valori di ritorno nei registri `$v0, $f0` (opzionale).

System Calls "Canoniche" (SPIM)

Syscall	Codice	Argomenti	Valore di ritorno	Descrizione
print_int	1	intero da stampare in $a0	nessuno	Stampa l'intero passato in $a0
print_float	2	float da stampare in $f12	nessuno	Stampa il float passato in $f12
print_double	3	double da stampare in $f12	nessuno	Stampa il double passato in $f12
print_string	4	Indirizzo della stringa da stampare in $a0	nessuno	Stampa la stringa che sta all'indirizzo passato in $a0
read_int	5	nessuno	Intero letto in $v0	Legge un intero in input e lo scrive in $v0
read_float	6	nesuno	Float letto in $f0	Legge un float in input e lo scrive in $f0
read_double	7	nessuno	Double letto in $f0	Legge un double in input e lo scrive in $f0
read_string	8	Indirizzo nel segmento dati a cui salvare la stringa in $a0 e lunghezza in byte in $a1	nessuno	Legge una stringa di lunghezza specificata in $a1 e la scrive nel segment dati all'indirizzo specificato in $a0
sbrk	9	Numero di byte da allocare in $a0	Indirizzo del primo dei byte allocati in $v0	Accresce il segmento dati allocanto un numero di byte specificato in $a0, restituisce in $v0 l'indirizzo del primo di questi nuovi byte
exit	10	nessuno	nessuno	Termina l'esecuzione

System Calls "Apocrife" (MARS)

Syscall	Codice	Argomenti	Valore di ritorno	Descrizione
Time	30	nessuno	32 bit meno significativi del system time in $a0, 32 bit più significativi del system time in $a1	Il system time è rappresentato nel formato Unix Epoch time, cioè il numero di millisecondi trascorsi dal 1 Gennaio 1970
random int	41	Id del generatore pseudo-random in $a0	Prossimo numero pseudo random in $a0	Ad ogni chiamata restituisce un numero intero in una sequenza pseudo-random
random in range	42	Id del generatore pseudo-random in $a0, massimo intero generabile in $a1	Prossimo numero pseudo random in $a0	Ad ogni chiamata restituisce un numero intero in una sequenza pseudo-random, ogni numero sarà compreso tra 0 e il massimo passato in $a1
MessageDialog	55	Indirizzo della stringa da stampare in $a0, intero corrispondente al tipo di messaggio in $a1		Mostra una finestra di dialogo con un messaggio dato dalla stringa passata in $a0. Viene anche mostrata una icona che dipende dal tipo di messaggio passato in $a1: errore (0), info, (1), warning (2), domanda(3)
InputDialogInt	51	Indirizzo della stringa da stampare in $a0	Intero letto in $a0, stato in $a1	

Esempio

```
        .data
msg1: .asciiz "Hello world!"
msg2: .asciiz "Inserisci un intero"

        .text
        .globl main
main:

        li $v0 4,
        la $a0, msg1
        syscall

        li $v0 55
        la $a0 msg1
        li $a1 1
        syscall

        li $v0, 51
        la $a0, msg2
        syscall

        li $v0 10
        syscall
```

Stampiamo una stringa nello standard output (la console)

Stampiamo una stringa in una finestra di diaologo

Leggiamo un intero in input con una finestra di dialogo

Exit

Esercizio 3.1

- Nome del file sorgente: *intsuccessivo.asm*

- Si scriva codice assembly che:
  - chieda all'utente di inserire un intero `i` e lo acquisisca
  - calcoli l'intero successivo `i+1` e lo stampi

Esercizio 3.1

```
        .data
        string1: .asciiz "Inserire numero intero: "
        string2: .asciiz "L'intero successivo è: "

        .text
        .globl main

main:
        li $v0, 4               # selezione di print_string (codice = 4)
        la $a0, string1         # $a0 = indirizzo di string1
        syscall                 # lancio print_string

        li $v0, 5               # Selezione read_int (codice = 5)
        syscall                 # lancio read_int
        add $t0, $zero, $v0     # leggo risultato $t0 = $v0

        li $v0, 4               # selezione di print_string
        la $a0, string2         # $a0 = indirizzo di string2
        syscall                 # lancio print_string

        addi $t0, $t0, 1        # $t0 += 1

        li $v0, 1               # selezione di print_int (codice = 1)
        add $a0,$zero,$t0       # $a0 = $t0
        syscall                 # lancio print_int

        li $v0, 10              # exit
        syscall
```

Esercizio 3.2

- Nome del file sorgente: *successivoarray.asm*

- Si scriva codice assembly che:
  - chieda all'utente di inserire un intero `i` e lo acquisisca
  - calcoli l'intero successivo `i+1`
  - memorizzi `i` e `i+1` in un array di 2 word in memoria
  - stampi i due numeri

Esercizio 3.2

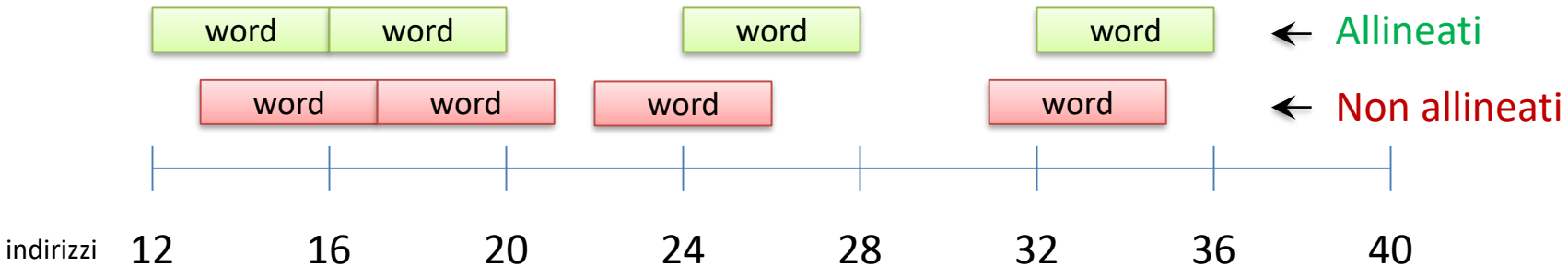
```
# con direttiva di allineamento
.data
string1: .asciiz "Dammi un intero: "
string2: .asciiz "I due numeri sono: "
string3: .ascii ", "
.align 2
array: .space 8
```

```
        .text
        .globl main
main:
        li $v0, 4               # selezione print_string
        la $a0, string1         # $a0 = indirizzo di string1
        syscall                 # lancio print_string

        li $v0, 5               # selezione di read_int
        syscall                 # lancio read_int (in $v0)

        la $s1, array           # $s1 base address di array
        add $t0, $zero, $v0     # $t0 = $v0
        sw $t0, 0($s1)          # array[0] = $t0
        addi $t0, $t0, 1        # $t0 += 1
        sw $t0, 4($s1)          # array[1] = $t0

        li $v0, 4               # selezione di print_string
        la $a0, string2         # $a0 = indirizzo di string2
        syscall                 # lancio print_string

        li $v0, 1               # selezione di print_int
        lw $t0, 0($s1)
        move $a0, $t0           # $a0 = array[0]
        syscall                 # lancio print_int

        li $v0, 4               # selezione di print_string
        la $a0, string3         # $a0 = indirizzo di string3
        syscall                 # lancio print_string

        li $v0, 1               # selezione di print_int
        lw $t0, 4($s1)
        move $a0, $t0           # $a0 = array[1]
        syscall                 # lancio print_int

        jr $ra
```

Allineamento dati

- L'accesso a memoria si dice allineato su n byte se:
  - ogni dato ha dimensione n byte
  - n è una potenza di 2
  - l'indirizzo di ogni dato è multiplo di n

- Nel nostro caso:
  - un dato è una word che ha dimensione 4 byte, quindi n=4
  - $(4 = 2^2)$
  - l'indirizzo di ogni word deve essere multiplo di 4



Esempio

```
        .data
        string: .asciiz "Ciao"
        A: .space 8

        .text
        .globl main

main:

        la $t0, A
        li $t1, 5
        sw $t1 0($t0)
```

Esempio

Il segmento dati inizia qui (indirizzo **0x10010000**), i dati che seguono sono allocati in modo sequenziale

```
.data
string: .asciiz "Ciao"
A: .space 8

.text
.globl main

main:

la $t0, A
li $t1, 5
sw $t1 0($t0)
```

Esempio

Il segmento dati inizia qui (indirizzo **0x10010000**), i dati che seguono sono allocati in modo sequenziale

```
.data
string: .asciiz "Ciao"
A: .space 8

.text
.globl main

main:

la $t0, A
li $t1, 5
sw $t1 0($t0)
```

La stringa «Ciao» verrà quindi allocata a partire dall'inizio del segmento:

Indirizzo	Valore
0x10010000	c
0x10010001	i
0x10010002	a
0x10010003	o
0x10010004	\0
0x10010005	prima word di A
0x10010009	seconda word di A

Esempio

Il segmento dati inizia qui (indirizzo **0x10010000**), i dati che seguono sono allocati in modo sequenziale

```
.data
string: .asciiz "Ciao"
A: .space 8

.text
.globl main

main:

la $t0, A
li $t1, 5
sw $t1 0($t0)
```

La stringa «Ciao» verrà quindi allocata a partire dall'inizio del segmento:

Indirizzo	Valore
0x10010000	c
0x10010001	i
0x10010002	a
0x10010003	o
0x10010004	\0
0x10010005	prima word di A
0x10010009	seconda word di A

Proseguendo nel segmento dati incontriamo **A**, la prima posizione disponibile per allocarlo è nel byte all'indirizzo **0x10010005**

Convertendo in base 10 si osserva che non è multipo di 4

$(0x10010005)_{16}=(268500997)_{10}$

Esempio

Il segmento dati inizia qui (indirizzo **0x10010000**), i dati che seguono sono allocati in modo sequenziale

```
.data
string: .asciiz "Ciao"
A: .space 8

.text
.globl main

main:

la $t0, A
li $t1, 5
sw $t1 0($t0)
```

La stringa «Ciao» verrà quindi allocata a partire dall'inizio del segmento:

Indirizzo	Valore
0x10010000	c
0x10010001	i
0x10010002	a
0x10010003	o
0x10010004	\0
0x10010005	prima word di A
0x10010009	seconda word di A

Proseguendo nel segmento dati incontriamo **A**, la prima posizione disponibile per allocarlo è nel byte all'indirizzo **0x10010005**

Convertendo in base 10 si osserva che non è multipo di 4
$(0x10010005)_{16}=(268500997)_{10}$

Cosa succede se tento di accedere ad un indirizzo non allineato con `sw` o `lw`?

Esempio

```
        .data
string: .asciiz "Ciao"
A: .space 8

        .text
        .globl main

main:

        la $t0, A
        li $t1, 5
        sw $t1 0($t0)
```

```
Go: running mips1.asm

Error in D:\Jacopo Essenziale\MEGA\MIPS_Stuff\mars\mips1.asm line 8: Runtime exception at 0x0040000c: store address not aligned on word boundary 0x10010005

Go: execution terminated with errors.
```

Le istruzioni di `sw` e `lw` richiedono di operare con accesso allineato con parole da 32 bit, quindi se specifichiamo un indirizzo **non** multiplo di 4 in MARS otteniamo un errore.

Esempio

```
        .data
string: .asciiz "Ciao"
        .align 2
        A: .space 8

        .text
        .globl main
main:

        la $t0, A
        li $t1, 5
        sw $t1 0($t0)
```

Aggiungendo la direttiva di allineamento viene lasciato spazio libero per mantenere l'allineamento

Ora A è viene allocato all'indirizzo $(0x10010008)_{16}=(67125250)_{10}$ che è multiplo di 4

Indirizzo	Valore
0x10010000	c
0x10010001	i
0x10010002	a
0x10010003	o
0x10010004	\0
0x10010005	
0x10010006	
0x10010007	
0x10010008	prima word di A
0x1001000C	seconda word di A

Esercizio 3.3

- Nome del file sorgente: *slotmachine.asm*

- Si scriva codice assembly che:
  - Richieda all'utente attraverso una finestra di dialogo l'inserimento di un numero intero NUM.
  - Estragga un numero casuale R nel range [-NUM,NUM] , (il seed del generatore di numeri casuali può essere inizializzato con un qualsiasi numero intero)
  - Sommi R al numero inserito NUM (RESULT = NUM + R).
  - Mostri all'utente attraverso una nuova finestra di dialogo il nuovo credito dell'utente dopo la scommessa RESULT.
  - Il programma dovrà terminare in maniera "pulita" restituendo il controllo al sistema operativo.

Esercizio 3.3

```
        .data
in_msg: .asciiz "Quanti $ vuoi scommettere?"
out_msg:.asciiz "Ora possiedi: $"

        .text
        .globl main

main:
        la $a0, in_msg          # Carica l'indirizzo della stringa di richiesta in $a0
        li $v0, 51              # Carica il codice della syscall "InputDialogInt" in $v0
        syscall                 # Esegui la syscall

        move $t0, $a0           # Sposta l'intero letto da $a0 in $t0
                                # (pseudo-istruzione) = add $t0, $zero, $a0

        mul $t1, $t0, 2         # $t1 = $t0 * 2
        add $t1, $t1, 1         # Il generatore di numeri casuali, estrae un numero tra 0
                                # e un limite massimo (ESCLUSO), per ottenere un numero casuale
                                # tra -num e +num dovremo estrarre un numero casuale tra [0 e (2 * num + 1)]
                                # dopo di che sottrarre al numero estratto il valore di num

        li $a0, 0               # Imposta il seed del generatore a 0
        move $a1, $t1           # Carica l'upper bound per il generatore in $a1
        li $v0, 42              # Carica il codice della syscall "random int range"
        syscall                 # Esegui la syscall

        move $t2, $a0           # Sposta in $t2 il numero casuale generato

        sub $t2, $t2, $t0       # Sottraiamo il valore inserito dall'utente dal numero casuale estratto
                                # per riportare il numero casuale dal range [0, 2 * num]
                                # nel range [-num, num]

        add $t3, $t0, $t2       # Somma il numero casuale estratto al valore inserito dall'utente

        la $a0, out_msg         # Carica l'indirizzo del messaggio di risposta in $a0
        move $a1, $t3           # Carica l'intero da stampare in $a1
        li $v0, 56              # Carica il codice della syscall "MessageDialogInt" in $v0
        syscall

        li $v0, 10              # Carica il codice della syscall "Exit" in $v0
        syscall                 # Uscita pulita dell'applicazione
```

Esercizio 3.4

- Eseguire e analizzare il seguente codice assembly e descrivere, in linguaggio di alto livello, le operazioni che esegue.

```
        .data
        v: .byte 2,0,0,0,4,0,0,0
        array: .byte 2,0,0,0,3,0,0,0,5,0,0,0,7,0,0,0,11,0,0,0,13,0,0,0,17,0,0,0,19,0,0,0


        .text
        .globl main

main:
        la $s1, array
        la $s2, v

        lw $t0, 0($s2)
        addi $t0, $t0, -1
        mul $t0, $t0, 4
        add $t1, $s1, $t0
        lw $t2, 0($t1)
        addi $t2, $t2, 1

        lw $t0, 4($s2)
        addi $t0, $t0, -1
        mul $t0, $t0, 4
        add $t3, $s1, $t0
        lw $t4, 0($t3)
        addi $t4, $t4, -1

        sw $t2, 0($t3)
        sw $t4, 0($t1)

        jr $ra
```

Esercizio 3.4

- Eseguire e analizzare il seguente codice assembly e descrivere, in linguaggio di alto livello, le operazioni che esegue.

```
        .data
        v: .byte 2,0,0,0,4,0,0,0
        array: .byte 2,0,0,0,3,0,0,0,5,0,0,0,7,0,0,0,11,0,0,0,13,0,0,0,17,0,0,0,19,0,0,0


        .text
        .globl main

main:
        la $s1, array           # carico gli indirizzi
        la $s2, v

        lw $t0, 0($s2)          # load valore 2
        addi $t0, $t0, -1       # sottraggo 1 perchè indirizzerà il secondo elemento
        mul $t0, $t0, 4         # offset (in bytes)
        add $t1, $s1, $t0       # indirizzo secondo elemento array (base + offset)
        lw $t2, 0($t1)          # load secondo elemento array
        addi $t2, $t2, 1        # incremento secondo elemento array

        lw $t0, 4($s2)          # load valore 4
        addi $t0, $t0, -1       # sottraggo 1 perchè indirizzerà il quarto elemento
        mul $t0, $t0, 4         # offset (in bytes)
        add $t3, $s1, $t0       # indirizzo quarto elemento array (base + offset)
        lw $t4, 0($t3)          # load quarto elemento array
        addi $t4, $t4, -1       # decremento quarto elemento array

        sw $t2, 0($t3)          # store in posizioni scambiate
        sw $t4, 0($t1)

        jr $ra
```

Esercizio 3.4

- Eseguire e analizzare il seguente codice assembly e descrivere, in linguaggio di alto livello, le operazioni che esegue.

```
        .data
        v: .byte 2,0,0,0,4,0,0,0
        array: .byte 2,0,0,0,3,0,0,0,5,0,0,0,7,0,0,0,11,0,0,0,13,0,0,0,17,0,0,0,19,0,0,0


        .text
        .globl main

main:
        la $s1, array
        la $s2, v

        lw $t0, 0($s2)
        addi $t0, $t0,
        mul $t0, $t0,
        add $t1, $s1,
        lw $t2, 0($t1)
        addi $t2, $t2,

        lw $t0, 4($s2)          # load valore 4
        addi $t0, $t0, -1       # sottraggo 1 perchè indirizzerà il quarto elemento
        mul $t0, $t0, 4         # offset (in bytes)
        add $t3, $s1, $t0       # indirizzo quarto elemento array (base + offset)
        lw $t4, 0($t3)          # load quarto elemento array
        addi $t4, $t4, -1       # decremento quarto elemento array

        sw $t2, 0($t3)          # store in posizioni scambiate
        sw $t4, 0($t1)

        jr $ra
```

Nome del file sorgente: *scambia.asm*

```
temp = array[v[0]-1];
array[v[0]-1] = array[v[1]-1] - 1;
array[v[1]-1] = temp + 1;
```

Università degli Studi di Milano
Laboratorio di Architettura degli Elaboratori II
Corso di Laurea in Informatica, A.A. 2017-2018

Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Hanno contribuito alla realizzazione di queste slides:

- Iuri Frosio
- Andrea Ferretti
- Jacopo Essenziale