



Chapter 9

SYSTEM SERVICES

- + Mỗi dịch vụ đc gán 1 con số (mã dịch vụ), với ~169 d.vụ
- + Cho mã d.vụ vào RAX >> Chuẩn bị thông tin khác (RSI, RDI, RDX,...) >> Ra lệnh syscall



Introduction

- There are many operations that an application program must use the operating system to perform
- Such operations include console output, keyboard input, file services (open, read, write, close, etc.), obtaining the time or date, requesting memory allocation, and many others.
- Accessing system services is how the application requests that the operating system perform some specific operation (on behalf of the process)
- More specifically, the *system call* is the interface between an executing process and the operating system



Calling System Services

- A system service call is logically similar to calling a function, where the function code is located within the operating system
- The function may require privileges to operate which is why control must be transferred to the operating system.
- When calling system services, arguments are placed in the standard argument registers.
- System services do not typically use stack-based arguments. This limits the arguments of a system services to six (6)
- To call a system service, the first step is to determine which system service is desired



- The general process is that the system service **call code** is placed in the **rax** register.
- The **call code** is a number that has been assigned for the specific system service being requested.
- These are assigned as part of the operating system and cannot be changed by application programs
- If any are needed, the arguments for system services are placed in the **rdi**, **rsi**, **rdx**, **r10**, **r8**, and **r9** registers (in that order).
- The following table shows the argument locations which are consistent with the standard calling convention.



- Each system call will use a different number of arguments (from none up to 6). However, the system service call code is always required
- After the call code and any arguments are set, the **syscall** instruction is executed

Register	Usage
rax	Call code (see table)
rdi	1st argument (if needed)
rsi	2nd argument (if needed)
rdx	3rd argument (if needed)
r10	4th argument (if needed)
r8	5th argument (if needed)
r9	6th argument (if needed)



System Calls

- Each x86-64 system call has a unique ID number
- Examples:

<i>Number</i>	<i>Name</i>	<i>Description</i>
0	read	Read file
1	write	Write file
2	open	Open file
3	close	Close file
4	stat	Get info about file
57	fork	Create process
59	execve	Execute a program
60	_exit	Terminate process
62	kill	Send signal to process

kết thúc tiến trình



Newline Character

- In the context of output, a newline means move the cursor to the start of the next line.
- The many languages, including C, it is often noted as “\n” as part of a string
- Nothing is displayed for the newline, but the cursor is moved to the start of the next line.
- In Unix/Linux systems, the linefeed, abbreviated LF with an ASCII value of 10 (or 0x0A), is used as the newline character.
- In Windows systems, the newline is carriage return, abbreviated as CR with an ASCII value 13 (or 0x0D) followed by the LF.

Mã ASCII:
10: newline
13:



Console Output

- The system service to output characters to the console is the system write (SYS_write).
- Like a high-level language characters are written to standard out (STDOUT) which is the console
- The STDOUT is the default file descriptor for the console
- The arguments for the write system service are as follows:

Register	SYS_write
rax	Call code = SYS_write (1) rax=1 (write)
rdi	Output location, STDOUT (1) rdi=1 (console)
rsi	Address of characters to output.
rdx	Number of characters to output.

- Assuming the following declarations:

```

STDOUT      equ    1           ; standard output
SYS_write    equ    1           ; call code for write

msg          db      "Hello World"
msgLen       dq      11

```

- For example to output “Hello World” (it’s traditional) to the console, the system write (SYS_write) would be used. The code would be as follows:

```

mov rax, 1
mov rdi, 1
mov rdx, 11

```

```

mov    rax, SYS_write
mov    rdi, STDOUT
mov    rsi, msg           ; msg address
mov    rdx, qword [msgLen] ; length value
syscall

```

Example, Console Output

- This example is a complete program to output some strings to the console.

```

; Example program to demonstrate console output.
; This example will send some messages to the screen.

; *****

section    .data

; -----
; Define standard constants.

LF          equ    10           ; line feed
NULL        equ    0           ; end of string
TRUE        equ    1
FALSE       equ    0

```

```

EXIT_SUCCESS    equ    0                ; success code

STDIN           equ    0                ; standard input
STDOUT          equ    1                ; standard output
STDERR          equ    2                ; standard error

SYS_read        equ    0                ; read
SYS_write       equ    1                ; write
SYS_open        equ    2                ; file open
SYS_close       equ    3                ; file close
SYS_fork        equ    57               ; fork

SYS_exit        equ    60               ; terminate
SYS_creat       equ    85               ; file open/create
SYS_time        equ    201              ; get time

; -----
; Define some strings.

message1        db    "Hello World.", LF, NULL
message2        db    "Enter Answer: ", NULL
newLine         db    LF, NULL

```

```

;-----
section .text
global _start
_start:

; -----
; Display first message.

    mov     rdi, message1
    call    printString

; -----
; Display second message and then newline

    mov     rdi, message2
    call    printString

    mov     rdi, newLine
    call    printString

; -----
; Example program done.

exampleDone:
    mov     rax, SYS_exit
    mov     rdi, EXIT_SUCCESS

```

```

; String must be NULL terminated.
; Algorithm:
; Count characters in string (excluding NULL)
; Use syscall to output characters

; Arguments:
; 1) address, string
; Returns:
; nothing

global printString
printString:
    push    rbx

; -----
; Count characters in string.

    mov     rbx, rdi
    mov     rdx, 0
strCountLoop:
    cmp     byte [rbx], NULL
    je      strCountDone
    inc     rdx
    inc     rbx
    jmp     strCountLoop
strCountDone:

```

```

; -----
; Call OS to output string.

    mov     rax, SYS_write      ; system code for write()
    mov     rsi, rdi           ; address of char's to write
    mov     rdi, STDOUT        ; standard out
                                ; RDX=count to write, set above
    syscall                    ; system call

; -----
; String printed, return to calling routine.

prtDone:
    pop     rbx
    ret


```

The output would be as follows:

```


Hello World.
Enter Answer: _

```



Console Input

- The system service to read characters from the console is the system read (SYS_read).
- Like a high-level language, for the console, characters are read from standard input (STDIN).
- We will need to declare an appropriate amount of space to store the characters being read
 - If we request 10 characters to read and the user types more than 10, the additional characters will be lost
 - If the user types less than 10 characters, for example 5 characters, all five characters will be read plus the newline (LF) for a total of six characters.



- The arguments for the read system service are as follows:

Register	SYS_read
rax	Call code = SYS_read (0) rax=0
rdi	Input location, STDIN (0) rdi=0 (chứa mã thiết bị bàn phím)
rsi	Address of where to store characters read.
rdx	Number of characters to read.

- Assuming the following declarations:

```

STDIN      equ      0                ; standard input
SYS_read   equ      0                ; call code for read

inChar     db        0
  
```




Example

- Read a single character from the keyboard, the system read (SYS_read) would be used. The code would be as follows:

```

mov    rax, SYS_read  mov rax, 0
mov    rdi, STDIN     mov rdi, 0
mov    rsi, inChar
mov    rdx, 1          ; msg address
                        ; read count
syscall

```



Example, Console Input

- Read a line of 50 characters from the keyboard, and then echo the input back to the console to verify that the input was read correctly

- Since space for the newline (LF) along with a final NULL termination is included, an input array allowing 52 bytes would be required.

```

; Example program to demonstrate console output.
; This example will send some messages to the screen.
; *****

```

```

section    .data

```

```

; -----
; Define standard constants.

```

```

LF          equ    10          ; line feed
NULL        equ    0          ; end of string

TRUE        equ    1
FALSE       equ    0

EXIT_SUCCESS equ    0          ; success code

STDIN       equ    0          ; standard input
STDOUT      equ    1          ; standard output
STDERR      equ    2          ; standard error

SYS_read    equ    0          ; read
SYS_write   equ    1          ; write
SYS_open    equ    2          ; file open
SYS_close   equ    3          ; file close
SYS_fork    equ    57         ; fork
SYS_exit    equ    60         ; terminate
SYS_creat   equ    85         ; file open/create
SYS_time    equ    201        ; get time

; -----
; Define some strings.

STRLEN      equ    50

pmpt        db      "Enter Text: " , NULL

```

```

section .bss
chr          resb    1
inLine      resb    STRLEN+2          ; total of 52

;-----

section .text
global _start
_start:

; -----
; Display prompt.

    mov     rdi, pmpt
    call    printString

; -----
; Read characters from user (one at a time)

```

```

    mov    rbx, inLine          ; inLine addr
    mov    r12, 0               ; char count
readCharacters:
    mov    rax, SYS_read        ; system code for read
    mov    rdi, STDIN           ; standard in
    lea    rsi, byte [chr]      ; address of chr
    mov    rdx, 1               ; count (how many to read)
    syscall                     ; do syscall

    mov    al, byte [chr]        ; get character just read
    cmp    al, LF               ; if linefeed, input done
    je     readDone

    inc    r12                  ; count++
    cmp    r12, STRLEN          ; if # chars ≥ STRLEN
    jae    readCharacters       ; stop placing in buffer

    mov    byte [rbx], al        ; inLine[i] = chr
    inc    rbx                  ; update tmpStr addr

    jmp    readCharacters
readDone:
    mov    byte [rbx], NULL      ; add NULL termination

```

```

; -----
; Output the line to verify successful read

    mov    rdi, inLine
    call   printString

; -----
; Example done.

exampleDone:
    mov    rax, SYS_exit
    mov    rdi, EXIT_SUCCESS
    syscall

; *****
; Generic procedure to display a string to the screen.
; String must be NULL terminated.
; Algorithm:
;   Count characters in string (excluding NULL)
;   Use syscall to output characters

; Arguments:
;   1) address, string
; Returns:
;   nothing

```

```

global printString
printString:
    push    rbx

; -----
;  Count characters in string.


    mov     rbx, rdi
    mov     rdx, 0

strCountLoop:
    cmp     byte [rbx], NULL
    je      strCountDone
    inc     rdx
    inc     rbx
    jmp     strCountLoop
strCountDone:

    cmp     rdx, 0
    je      prtDone

; -----
;  Call OS to output string.

```



```

    mov     rax, SYS_write    ; system code for write()
    mov     rsi, rdi          ; address of char's to write
    mov     rdi, STDOUT       ; standard out
                                ; RDX=count to write, set above
    syscall                   ; system call

; -----
;  String printed, return to calling routine.

prtDone:
    pop     rbx
    ret

```



File Open Operations

- In order to perform file operations such as read and write, the file must first be opened.
- There are two file open operations, open and open/create.
- After the file is opened, in order to perform file read or write operations the operating system needs detailed information about the file, including the complete status and current read/write location.
- If the file open operation fails, an error code will be returned.
- If the file open operation succeeds, a file descriptor is returned.



- The complete set of information about an open file is stored in an operating system data structure named File Control Block (FCB)

■ File Open

- The file open requires that the file exist in order to be opened. If the file does not exist, it is an error
- The file open operation also requires the parameter flag to specify the access mode. The access mode must include one of the following
 - Read-Only Access → O_RDONLY
 - Write-Only Access → O_WRONLY
 - Read/Write Access → O_RDWR

- The arguments for the file open system service are as follows:

Register	SYS_open
rax	Call code = SYS_open (2)
rdi	Address of NULL terminated file name string
rsi	File access mode flag thuộc tính đọc/ghi

- Assuming the following declarations:


```

SYS_open      equ      2           ; file open

O_RDONLY      equ      000000q    ; read only
O_WRONLY      equ      000001q    ; write only
O_RDWR        equ      000002q    ; read and write

```

- After the system call, the **rax** register will contain the return value.
- If the file open operation fails, **rax** will contain a negative value (i.e., < 0).
- If the file open operation succeeds, **rax** contains the file descriptor
- **File Open/Create**
 - A file open/create operation will create a file
 - If the file does not exist, a new file will be created
 - If the file already exists, it will be erased and a new file created.
 - Since the file is being created, the access mode must include the file permissions that will be set when the file is created.



- The arguments for the file open/create system service are as follows: **mở dạng tạo mới**

Register	SYS_creat
rax	Call code = SYS_creat (85)
rdi	Address of NULL terminated file name string
rsi	File access mode flag


- Assuming the following declarations:

```

SYS_creat      equ      85           ; file open

O_CREAT        equ      0x40
O_TRUNC        equ      0x200
O_APPEND       equ      0x400

S_IRUSR        equ      00400q      ; owner, read permission
S_IWUSR        equ      00200q      ; owner, write permission
S_IXUSR        equ      00100q      ; owner, execute permission
  
```



- File Read**
 - A file must be opened with the appropriate file access flags before it can be read.
 - The arguments for the file read system service are as follows:

Register	SYS_read
rax	Call code = SYS_read (0)
rdi	File descriptor (of open file)
rsi	Address of where to place characters read
rdx	Count of characters to read

- Assuming the following declarations:

```

SYS_read      equ      0           ; file read
  
```

- If the file read operation does not succeed, a negative value is returned in the **rax** register. If the file read operation succeeds, the number of characters actually read is returned



■ File Write

- The arguments for the file write system service are as follows:

Register	SYS_write
rax	Call code = SYS_write (1)
rdi	File descriptor (of open file) bộ mô tả file
rsi	Address of where to place characters to write
rdx	Count of characters to write

- Assuming the following declarations:

```
SYS_write    equ    1                ; file write
```
- If the file write operation does not succeed, a negative value is returned in the **rax** register. If the file write operation does succeed, the number of characters



Example, File Write

- Program writes a short message to a file
 - The file created contains a simple message, a URL in this example.
 - The file name and message to be written to the file are hard-coded

```
; Example program to demonstrate file I/O. This example
; will open/create a file, write some information to the
; file, and close the file. Note, the file name and
; write message are hard-coded for the example.
;-----

section    .data

; -----
; Define standard constants.
```



```

LF          equ    10          ; line feed
NULL        equ    0          ; end of string
TRUE        equ    1
FALSE       equ    0
EXIT_SUCCESS equ    0          ; success code
STDIN       equ    0          ; standard input
STDOUT      equ    1          ; standard output
STDERR      equ    2          ; standard error

SYS_read    equ    0          ; read
SYS_write   equ    1          ; write
SYS_open    equ    2          ; file open
SYS_close   equ    3          ; file close
SYS_fork    equ    57         ; fork
SYS_exit    equ    60         ; terminate
SYS_creat   equ    85         ; file open/create
SYS_time    equ    201        ; get time

```

```

O_CREAT     equ    0x40
O_TRUNC     equ    0x200
O_APPEND    equ    0x400

O_RDONLY    equ    000000q    ; read only
O_WRONLY    equ    000001q    ; write only
O_RDWR      equ    000002q    ; read and write

S_IRUSR     equ    00400q
S_IWUSR     equ    00200q
S_IXUSR     equ    00100q

; -----
; Variables for main.

newLine     db      LF, NULL
header      db      LF, "File Write Example."
            db      LF, LF, NULL
fileName    db      "url.txt", NULL
url         db      "http://www.google.com"
            db      LF, NULL
len         dq      $-url-1

writeDone   db      "Write Completed.", LF, NULL
fileDescrip dq      0
errMsgOpen  db      "Error opening file.", LF, NULL
errMsgWrite db      "Error writing to file.", LF, NULL

```

```

section    .text
global _start
_start:

; -----
;  Display header line...

        mov     rdi, header
        call    printString

; -----
;  Attempt to open file.
;  Use system service for file open

;  System Service - Open/Create
;      rax = SYS_creat (file open/create)
;      rdi = address of file name string
;      rsi = attributes (i.e., read only, etc.)

```

```

; Returns:
;     if error -> eax < 0
;     if success -> eax = file descriptor number

; The file descriptor points to the File Control
; Block (FCB). The FCB is maintained by the OS.
; The file descriptor is used for all subsequent
; file operations (read, write, close).

openInputFile:
    mov     rax, SYS_creat           ; file open/create
    mov     rdi, fileName           ; file name string
    mov     rsi, S_IRUSR | S_IWUSR  ; allow read/write
    syscall                          ; call the kernel

    cmp     rax, 0                  ; check for success
    jl      errorOnOpen

```

```

    mov qword [fileDescrip], rax      ; save descriptor
; -----
; Write to file.
; In this example, the characters to write are in a
; predefined string containing a URL.

; System Service - write
; rax = SYS_write
; rdi = file descriptor
; rsi = address of characters to write
; rdx = count of characters to write
; Returns:
; if error -> rax < 0
; if success -> rax = count of characters actually read

    mov     rax, SYS_write
    mov     rdi, qword [fileDescriptor]
    mov     rsi, url
    mov     rdx, qword [len]
    syscall

```

```

    cmp     rax, 0
    jnl    errorOnWrite

    mov     rdi, writeDone
    call    printString

    jmp    exampleDone

; -----
; Close the file.
; System Service - close
; rax = SYS_close
; rdi = file descriptor

    mov     rax, SYS_close
    mov     rdi, qword [fileDescriptor]
    syscall

```

```

        jmp exampleDone

; -----
; Error on open.
; note, rax contains an error code which is not used
; for this example.

errorOnOpen:
    mov     rdi, errMsgOpen
    call    printString

    jmp exampleDone

; -----
; Error on write.
; note, rax contains an error code which is not used
; for this example.

errorOnWrite:
    mov     rdi, errMsgWrite
    call    printString

    jmp exampleDone

```

```

; Example program done.

exampleDone:
    mov     rax, SYS_exit
    mov     rdi, EXIT_SUCCESS
    syscall

; *****
; Generic procedure to display a string to the screen.
; String must be NULL terminated.
; Algorithm:
; Count characters in string (excluding NULL)
; Use syscall to output characters

; Arguments:
; 1) address, string
; Returns:
; nothing

```

```

global printString
printString:
    push    rbp
    mov     rbp, rsp
    push    rbx

; -----
; Count characters in string.

    mov     rbx, rdi
    mov     rdx, 0
strCountLoop:
    cmp     byte [rbx], NULL
    je      strCountDone
    inc     rdx
    inc     rbx
    jmp     strCountLoop
strCountDone:
    cmp     rdx, 0
    je      prtDone

; -----
; Call OS to output string.

```

```

    mov     eax, SYS_write          ; code for write()
    mov     rsi, rdi                ; addr of characters
    mov     rdi, STDOUT             ; file descriptor
                                     ; count set above
    syscall                         ; system call

; -----
; String printed, return to calling routine.

prtDone:
    pop     rbx
    pop     rbp
    ret

; *****

```

