CMSC 131 Introduction to Computer Organization and Machine-Level Programming
Handout 10: Input/Output System Services

Learning Outcomes

At the end of this session, the student should be able to:

1. Understand how the console output and input works in the Assembly language;
2. Use system services for input and output

Content

Calling System Services

A system service call operates in a logically similar manner as to function calls. However, a system service call may require privileges to operate which is why control is transferred to the operating system after locating the proper function code.

Calling system services also require our assembly programs to place arguments in the standard argument registers. Note that system services do not typically use stack-based arguments. This can limit system service arguments to six but it does not usually present a significant limitation.

There are several system services available in our operating systems (e.g. console output, keyboard input, file services, obtaining date and time, requesting memory allocation, etc.), each service corresponds to specific call codes. These call codes are assigned by the operating system and cannot be changed by application programs.

Call Code *(rax)*	**System Service**	**Description**
0	SYS_read	Read characters rdi = file descriptor (of where to read from) rsi = address of where to store characters rdx = count of characters to read If unsuccessful, it returns a negative value. If successful, it returns the count of characters actually read.
1	SYS_write	Write characters rdi = file descriptor (of where to read from) rsi = address of characters to write rdx = count of characters to write

		If unsuccessful, it returns a negative value. If successful, it returns the count of characters actually written.
2	SYS_open	Open a file rdi = address of NULL-terminated filename rsi = file status flags (typically O_RDONLY) If unsuccessful, it returns a negative value. If successful, it returns the file descriptor.
3	SYS_close	Close an open file rdi = file descriptor of the file to close If unsuccessful, it returns a negative value.
60	SYS_exit	Terminate executing process rdi = exit status (typically 0)

NOTE: The table only presents the most common call codes used in CMSC 131.

To invoke a system service, we must first identify its corresponding call code. The call code will be placed in the **rax** register accompanied by its optional arguments as stated in the table below.

Register	Usage
rax	Call code
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)

Once the call code and all of its arguments are set, the **syscall** instruction is executed. The syscall instruction pauses the current process and transfers control to the operating system in order to perform the service specified in the **rax** register. Once the system service returns, the succeeding processes will resume.

Console Output

SYS_WRITE is the system service used to output characters in the console. The arguments for the write system service are as follows:

Register	SYS_WRITE
rax	Call code = SYS_WRITE (1)
rdi	Output location = STDOUT (1) *STDOUT is the file descriptor for the console*
rsi	Address of characters to output
rdx	Number of characters to output

If we assume the following declarations:

```
LF equ 10              ; line feed (newline)
NULL equ 0             ; end of string
STDOUT equ 1
SYS_WRITE equ 1
msg db "Hello World", LF, NULL
msgLen dq 13
newLine db LF, NULL
```

Then, we could print the message “Hello World” using the following code:

```
mov rax, SYS_WRITE
mov rdi, STDOUT
mov rsi, msg
mov rdx, qword[msgLen]
syscall
```

Console Input

SYS_READ is the system service used to read characters from the console. The arguments for the read system service are as follows:

Register	SYS_WRITE
rax	Call code = SYS_READ (0)
rdi	Input location = STDIN (0) *STDIN is the file descriptor for reading characters from the keyboard*

rsi	Address of where to store characters to read
rdx	Number of characters to read

If we assume the following declarations:

```
STDIN equ 0
SYS_READ equ 0
inChar db 0
```

Then, we could *read a single character* from the keyboard using the code below:

```
mov rax, SYS_READ
mov rdi, STDIN
mov rsi, inChar
mov rdx, 1
syscall
```

String Conversion (string ⇔ integer/number)

We must take note that all the given input read from the console is processed as strings. We need to first convert strings with numeric characters if we are to use them as numbers. To understand the conversion of numeric strings in Assembly, we must first familiarize ourselves with the ASCII table.

DECIMAL VALUE	STRING/SYMBOL	DECIMAL VALUE	STRING/SYMBOL
48	0	53	5
49	1	54	6
50	2	55	7
51	3	56	8
52	4	57	9

So, for example, we have a string equivalent to '0', we just need to subtract 48 (or 30h in hex value), to get the equivalent decimal value of the numeric string that we are converting.

Example:

```
; gets the sum of two single-digit input numbers then prints the result

global _start
```

```
section .data
      LF equ 10
      NULL equ 0
      SYS_EXIT equ 60
      STDOUT equ 1
      SYS_WRITE equ 1
      STDIN equ 0
      SYS_READ equ 0

      msg db "Enter a single-digit number: ", NULL    ; always end your
strings with NULL
      msgLen equ $-msg
      newLine db LF, NULL
      newLineLen equ $-newLine
      num1 db 0
      num2 db 0
      sum db 0

section .text
_start:
      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, msg
      mov rdx, msgLen
      syscall

      mov rax, SYS_READ
      mov rdi, STDIN
      mov rsi, num1
      mov rdx, 2              ; one for the digit and one for the newline
character
      syscall

      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, msg
      mov rdx, msgLen
      syscall

      mov rax, SYS_READ
      mov rdi, STDIN
      mov rsi, num2
      mov rdx, 2              ; one for the digit and one for the newline
character
      syscall

      sub byte[num1], 30h     ; convert to decimal equivalent
      sub byte[num2], 30h

      ; get the sum of the two numbers
      mov al, byte[num1]
      add al, byte[num2]      ; ax register holds the sum

      ; convert back to string for printing
```

```
      mov bl, 10                    ; divide the sum in the ax register
by 10 to get the tens and ones place values
      div bl
      mov byte[num1], al      ; reuse num1 variable to store tens
      mov byte[num2], ah      ; reuse num2 variable to store ones

      add byte[num1], 30h           ; add 30h to get the numeric symbol
equivalent of the decimal value
      add byte[num2], 30h

      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, num1
      mov rdx, 1
      syscall

      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, num2
      mov rdx, 1
      syscall

      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, newLine
      mov rdx, newLineLen
      syscall

exit_here:
      mov rax, SYS_EXIT
      xor rdi, rdi
      syscall
```

String Conversion (uppercase ↔ lowercase)

The same concept may be applied when converting alphabetical characters to uppercase or lowercase.

DECIMAL VALUE	STRING/SYMBOL	DECIMAL VALUE	STRING/SYMBOL
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e

Example:

```
; converts “A” to lower case

global _start

section .data
      LF equ 10
      NULL equ 0
      SYS_EXIT equ 60
      STDOUT equ 1
      SYS_WRITE equ 1

      var db 0
      newLine db LF, NULL
      newLineLen equ $-newLine

section .text
_start:
      mov byte[var], "A"
      add byte[var], 32

      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, var
      mov rdx, 1
      syscall

      mov rax, SYS_WRITE
      mov rdi, STDOUT
      mov rsi, newLine
      mov rdx, newLineLen
      syscall

exit_here:
      mov rax, SYS_EXIT
      xor rdi, rdi
      syscall
```

References

[1] Jorgensen, Ed. 2019. x86-64 Assembly Language Programming with Ubuntu. Version 1.1.40.
[2] https://www.ascii-code.com/