System Call

- We call printf and scanf functions of C library (libc) to read and write in a C program.
- These functions process the input parameters and send request to the OS for IO.
- The request is sent through some machine instructions called system call or software interrupt or trap.

Instruction: int n

- **In a Intel processor, the int n instruction calls the interrupt handler for the interrupt n ($[0 \cdots 255]$).**
- **The values of n in the range $0 \cdots 31$ are reserved for specific exceptions e.g. int 0 is for a divide-by zero and int 14 is for a page-fault**
- **The values of n in the range $32 \cdots 255$ are not reserved.**

Instruction: int 0x80

- Linux uses int 128 or int 0x80 with different parameters for different system-calls e.g. the system-call write().

write()

```
int write(int fd, const void *buf, int count);
```

It uses the following parameters:

- Command - write: eax $\leftarrow$ 4
- File descriptor - STDOUT: ebx $\leftarrow$ 1
- Buffer Pointer: ecx $\leftarrow$ buff
- Number of Characters: edx $\leftarrow$ count

File: write.s

```
   .data
message:
   .string "The First Program using int 128\n"
   .text
   .globl main
main:
   movl  $4, %eax          # 4 for write
   movl  $1, %ebx          # 1 for STDOUT
   movl  $message, %ecx    # string address
   movl  $40, %edx         # length
   int  $128               # Software Interrupt
```

File: write.s

```
# For exit(0)
movl  $1, %eax      # 1 for exit
xorl  %ebx,%ebx
int  $128
ret
```

Assemble and Run

```
$ cc write.s
$ a.out
The First Program using int 128
$
```

Note ···

- **Data is converted to a character string.**
- **The output-buffer is not flushed unless there is a '$\backslash n$' at the end of the string.**

Read and Write

```
    .data
    .global n
    .align 4
    .type n,@object
    .size n,40
n:
    .zero 40
    .text
    .globl main
```

Read and Write

```
main:
   movl   $3, %eax          # 3 for read
   movl   $0, %ebx          # 0 for STDIN
   movl   $n, %ecx          # string address
   movl   $40, %edx         # length
   int    $0x80

   movl   $4, %eax          # 4 for write
   movl   $1, %ebx          # 1 for STDOUT
   movl   $n, %ecx          # string address
   movl   $40, %edx         # length
   int    $0x80
```

Read and Write

```
movl  $1, %eax      # 1 for exit
xorl  %ebx,%ebx
int  $0x80
leave
ret
```

Assemble and Run

```
$cc readWrite.s
$ a.out
IIT Kharagpur - CAOS
IIT Kharagpur - CAOS
$
```

Calling Assembly Code from C: mainInOut.c

```
#include "inOutStr.h"
int main() {
 int i, j ; char c ;

 inString() ;
 for(i=0, j = 38; i<j; ++i, --j) {
   c = data[i] ; data[i] = data[j];
   data[j] = c;
 }
 outString(); printf("\n") ;
}
```

inString.s

```
  .text
  .globl inString
inString:
  movl  $3, %eax          # 3 for read
  movl  $0, %ebx          # 0 for STDIN
  movl  $data, %ecx       # string address
  movl  $40, %edx         # length
  int   $0x80
  ret
  .comm   data,40,32
```

OutString.s

```
  .globl outString
outString:
  movl  $4, %eax          # 4 for write
  movl  $1, %ebx          # 1 for STDOUT
  movl  $data, %ecx       # string address
  movl  $40, %edx         # length
  int   $0x80
  ret
```

inOutStr.h

```
void inString(), outString() ;
extern char data[40]
```

Compile and Run

```
$ cc -c inString.s
$ cc -c outString.s
$ cc -c mainInOut.c
$ cc inString.o outString.o mainInOut.o
$ a.out
IIT Kharagpur - CAOS

SOAC - rupgarahK TII
$
```

Assignment-III

Write two functions myRead() and myWrite() satisfying the following prototypes and specifications.

Assignment-III (cont.)

- **Prototype:** int myRead(char, void *)
- **Valid first parameters:** 'i' (integer), 'f' (float) and 'c' (character).
- **The second parameter:** data pointer.
- To read an integer we pass 'i' as the first parameter and an integer pointer (pointing to a valid location) as the second parameter. Other two cases are also similar.

Assignment-III (cont.)

- The function uses int 0x80 to read the input string, converts it to the appropriate data type according to the first parameter (it may use atof, atoi library functions) and then stores the value in the location pointed by the second parameter.

Assignment-III (cont.)

- **Prototype:** int myWrite(char, void *)
- **Valid first parameters:** 'i' (integer), 'f' (float), 'c' (character), and 's' (string).
- **The second parameter:** data pointer.
- To write an integer we pass 'i' as the first parameter and an integer pointer (pointing to a valid location) as the second parameter. Other three cases are also similar. A string is terminated by a '\0' as usual.

Assignment-III (cont.)

- The function converts the data to a string of ASCII characters e.g. a floating-point number 123.50 will be converted to 0x31 0x32 0x33 0x2E 0x35. Then it uses int 0x80 to write the string.
- In both the cases the return value is to handle an error (0 - no error, 1 - error).

Assignment-III (cont.)

- **Assume that the function prototypes are available in myStdin.h which is to be included.**

URL

http://world.std.com/~slanning/asm/syscall_list.html

http://linuxassembly.org/articles/linasm.html#Intro

How to Create and Use an Archive

- Let the function myRead() and myWrite() be available in files myRead.c and myWrite.c.
- We can compile them as object modules as follows:

  $cc -Wall -c myRead.c

  $cc -Wall -c myWrite.c
- We can construct the archive as:

  $ar -qv libmyRW.a myRead.o myWrite.o

How to Create and Use an Archive

- If the archive is in the current directory, it can be used in the following way:
  $cc test.c -L. -lmyRW
- If the archive can be kept in /usr/lib it can be used as:
  $cc test.c -lmyRW