File Processing

Simplified view of File System & File I/O

References

1. File Descriptor: https://www.computerhope.com/jargon/f/file-descriptor.htm
2. File System: https://slideplayer.com/slide/5229567/
3. File Structure: https://www.mysterybox.us/blog/2017/7/27/protecting-your-digital-assets-part-3-recovering-from-failure
4. https://www.programiz.com/c-programming/c-file-input-output#closing

Files Storage

- Files are stored in external storage
  - HD
  - SSD (these days)
  - Other storage mediums
  - Occasionally in RAM!
- We are essentially talking about files in HD in the presentation
- File includes *Meta Data* and & *File Data*
- Both data are stored in external storage
- *Inode* is a pointer to Meta-Data

File Storage on Media

File Metadata (Stored in File System)

Name
Folder Path
Owner
Size
Permission
Date Created/Modified
Etc.

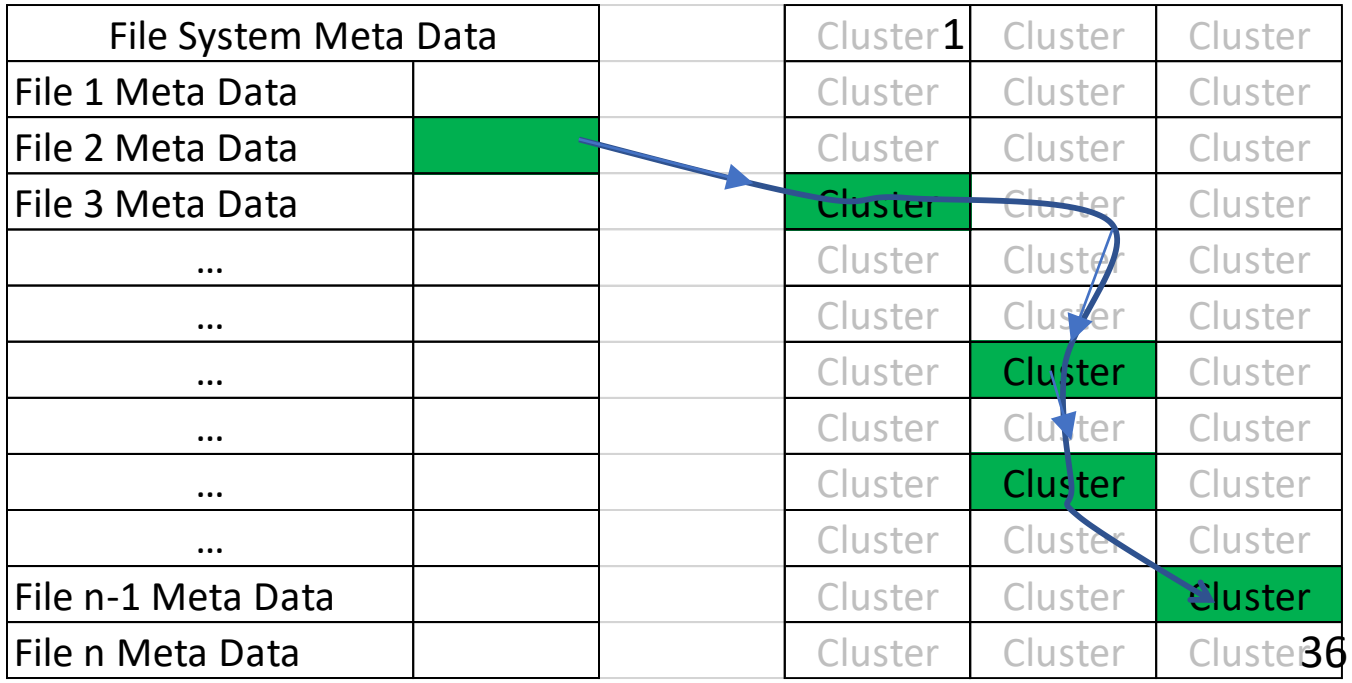
File Data (Actual Contents of File Stored on Disk)

- Block 1
- Block 2
- Block 3
- Block 4
- Block 5
- Block 6
- .....
- .....
- Block 7
- Block 8
- Block 9
- Block N

File metadata, such as the file name, size, etc. is all stored in the file system, for easy access by the operating system. Actual file contents are stored in blocks on the remainder of the media.

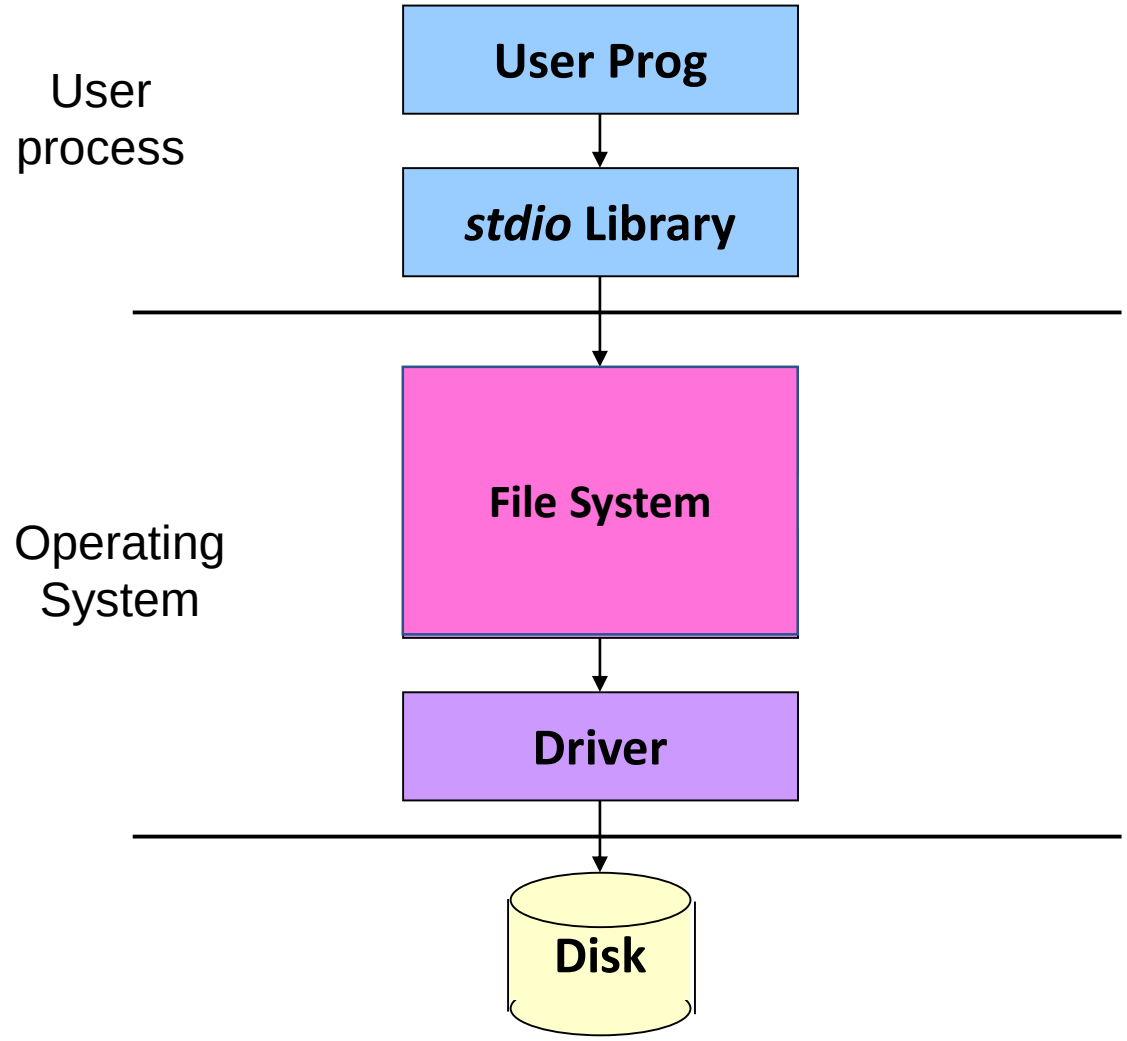
File Cluster - Chaining

- Typically, a large file is stored in multiple *clusters*
- The clusters are *chained* using pointers
- The *Meta Data* for a file includes a *pointer* to the first cluster of the file
- Example: File 2 contains 4 clusters
- *Indexing* can also be used as an alternative to *chaining*



Why a file is in multiple discontinuous clusters?

File system

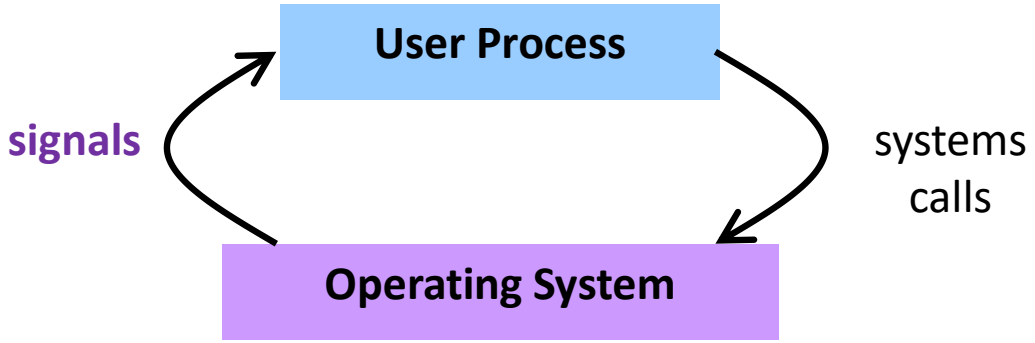


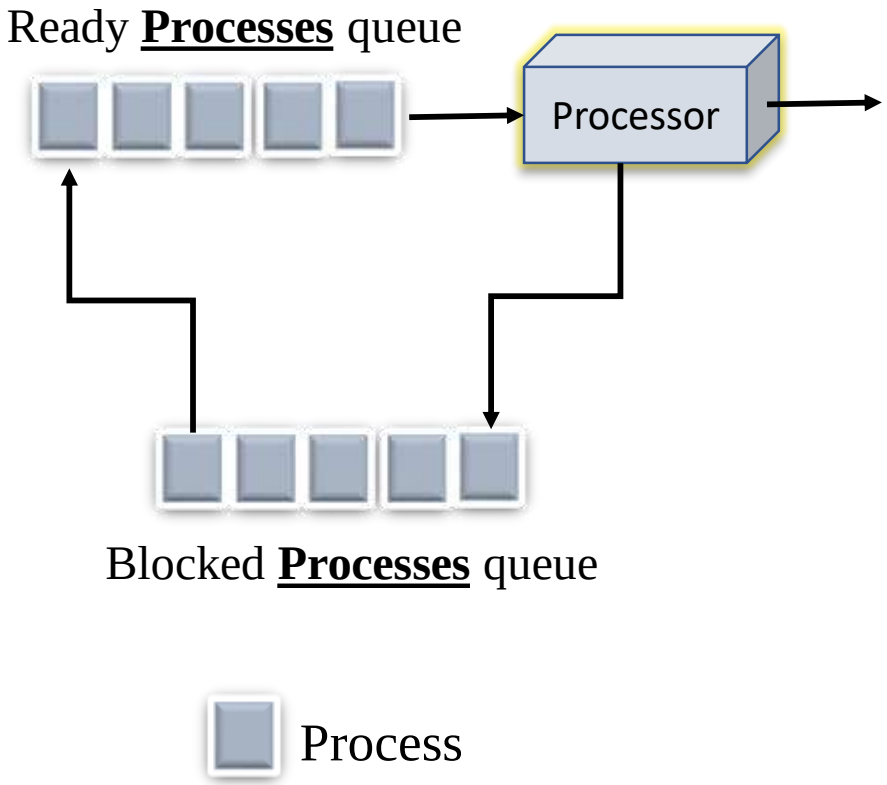
Why File System Abstraction?

Why do we need File System Abstraction in OS?

Communicating with OS

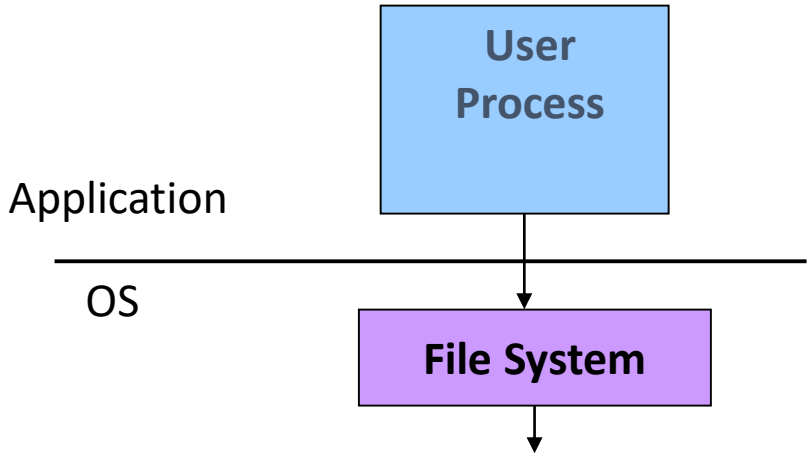
- **System call**
  - Request to the operating system to perform a task that the process does not have permission to perform
- **Signal**
  - ***Asynchronous* notification** sent to a process from OS to notify the occurrence of **an event**
- **Standard Library Calls**
  - Generic I/O support
  - A smart wrapper around I/O-related **system calls.**
  - This means, system calls are normally embedded in some library calls.





System Calls

- Method for user process to invoke *OS services*
- Called just like a function Call; a "protected" function call
- The control is transferred to OS and back

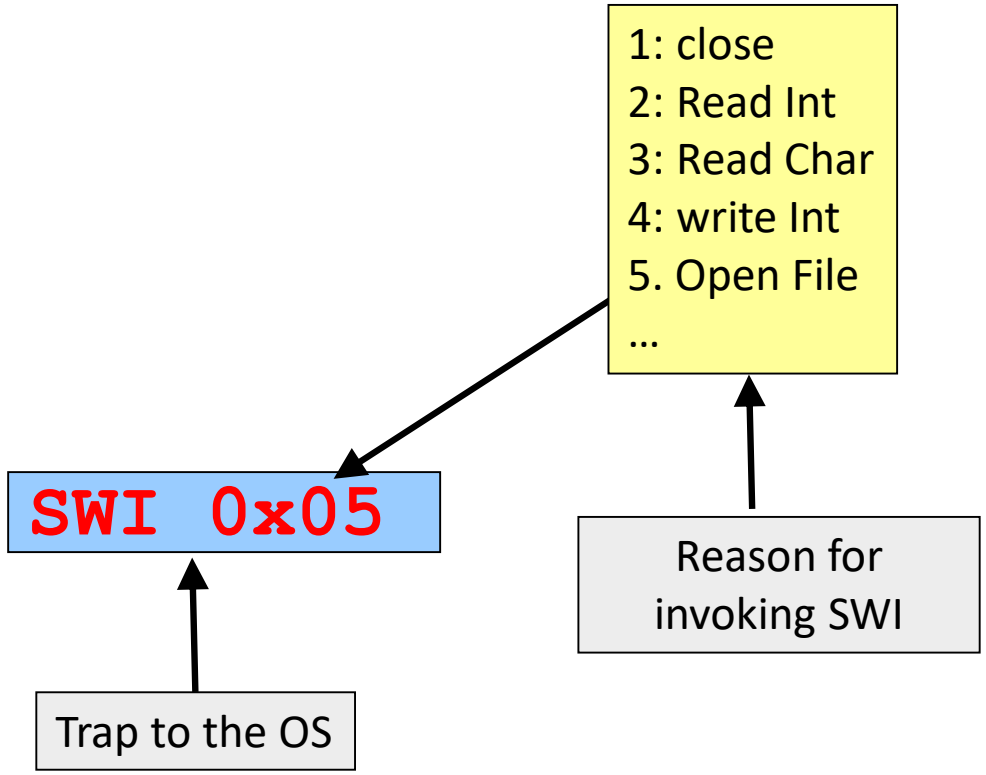


Implementing a System Call

- System calls are implemented using

  Software Interrupt (SWI) AKA trap

  - OS is given control through trap
  - Switches to supervisor mode
  - Performs the service
  - Switches back to user mode
  - Gives control back to user



System-call specific arguments are passed with SWI

Library and API

- **Library** implements reusable code that saves time
- **Library** and **API** are related terms
- Like other C functions, library functions have
  - **Declaration part (API)**
  - **Definition part**

Library Declaration and Definition

Declaration

function

abs

C | C++98 | C++11

```
          int abs (          int n);
     long int abs (     long int n);
long long int abs (long long int n);
```

Absolute value

Returns the absolute value of parameter *n* (/n/).

In C++, this function is also overloaded in header
for complex numbers (see complex abs), and in h

Parameters

n

Integral value.

Return Value

The absolute value of n.

Definition

```
int abs(int number) {
  return number >= 0 ? number : -number;
}

div_t div(int numer, int denom) {
  div_t result;

  result.quot = numer / denom;
  result.rem  = numer - (result.quot * denom);

  if (numer < 0 && result.rem > 0) {
    result.quot++;
    result.rem -= denom;
  }
```

Using Library Function

EXAMPLE PROGRAM FOR ABS( ) FUNCTION IN C:

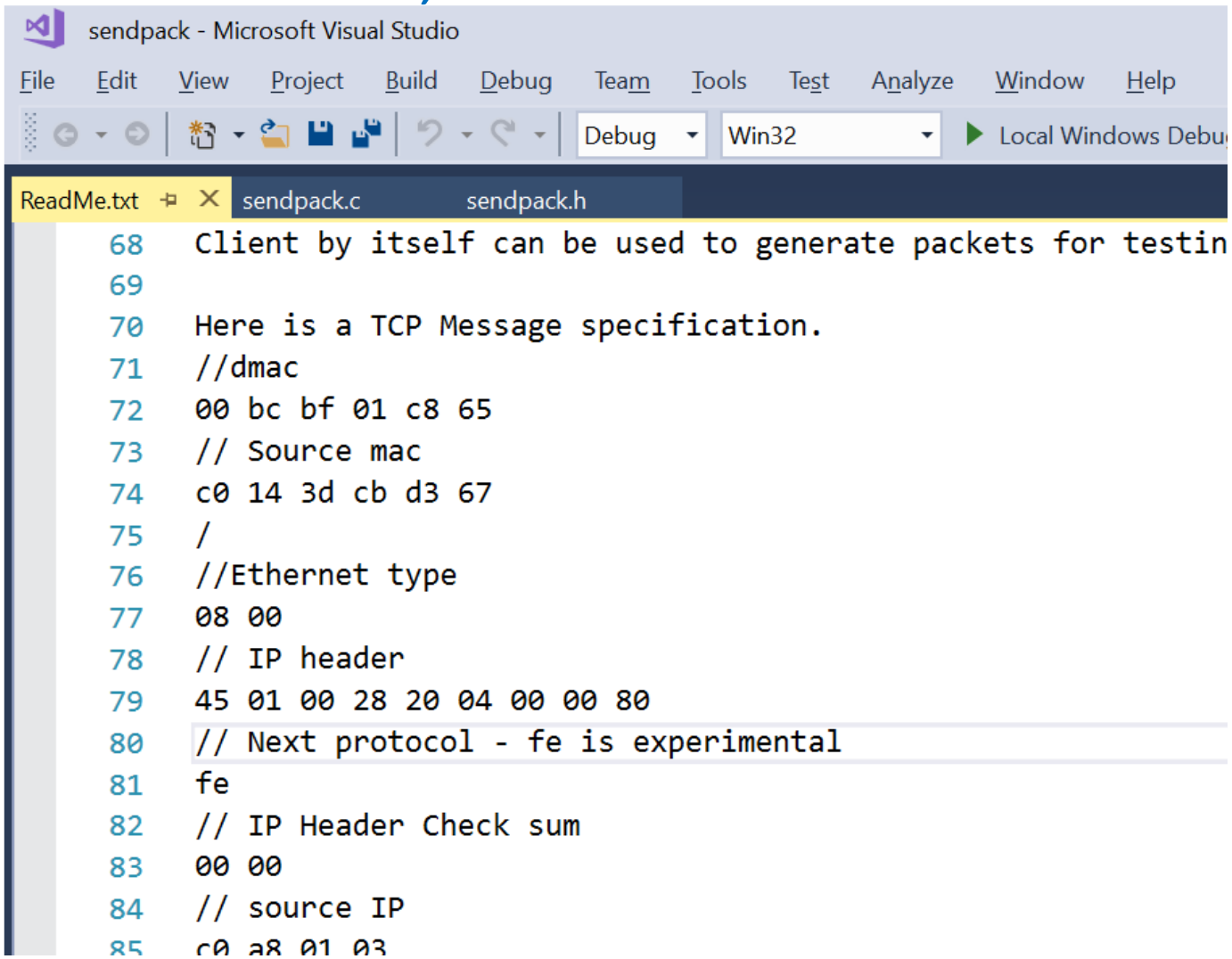
```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int m = abs(200);      // m is assigned to 200
    int n = abs(-400);     // n is assigned to -400

    printf("Absolute value of m = %d\n", m);
    printf("Absolute value of n = %d \n",n);
    return 0;
}
```

Integrated Development Environment (IDE)

- An **IDE** is an integrated set of tools and libraries.
- An **IDE** makes program development easier
- An **IDE** can support programming with multiple languages
- **Example**: **Microsoft Visual Studio** and **Arduino IDE**

Walkthrough demonstration of Visual Studio

IDE – Tool Chain, Libraries

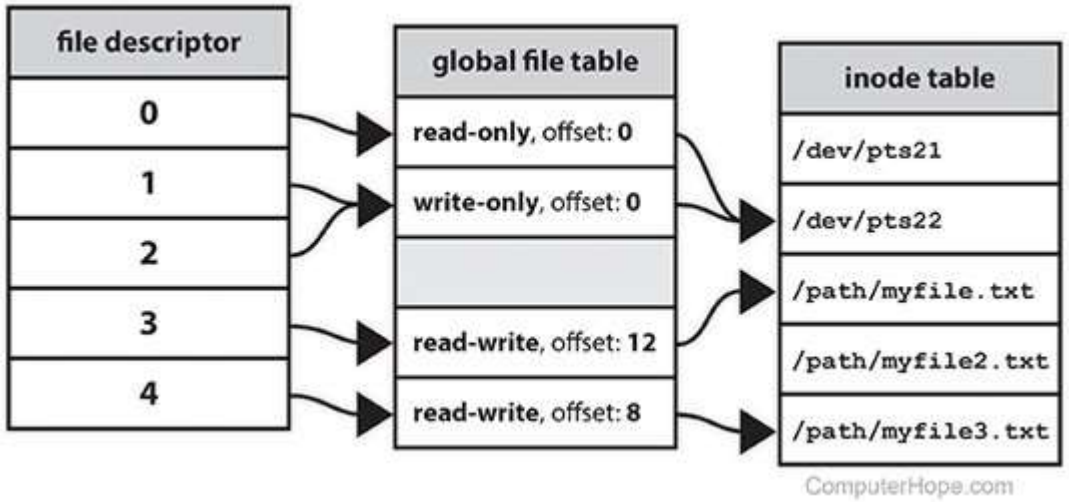


Framework

- Like library, a **Framework** also enables code reuse
- A **framework** offers customizable generic programs and libraries for a specific Application
- **Framework**: Developer's code is embedded to framework to create custom application – **on the other hand**
- **Library**: Library code is embedded within developer's code

File Handle

- When a *process* makes a successful request to **open a file**, the OS *returns* a ***file descriptor*** which points to an entry in the OS's **global file table**.
- The file table entry contains pointer to ***inode*** of the file, byte offset, and the access restrictions etc.
- The ***inode*** is the *Meta Data* for a file



Special Files: STDIN, STDOUT, and STDERR

Name	File descriptor	Description	Abbreviation
Standard input	**0**	This defaults to **keyboard in**put from the user.	**STDIN**
Standard output	**1**	This defaults to the **user's screen**.	**STDOUT**
Standard error	**2**	This defaults to **the user's screen**.	**STDERR**
Your File	3	In HD	filename

File with Tcl

Why Tcl? Simpler and easy to work with in classrooms. Our objective here is to learn File I/O not to deal with logistical issues. Here also we may encounter syntactical issues, but we can build and test code faster

Tcl *File I/O* Commands

- **open** *fileName access*
- **close** *filedescriptor*
- ***gets*** *filedescriptor*
- ***puts*** *filedescriptor*
- ***Read*** *filedescriptor*
- *There are also a number file commands that operate on a file's name or attributes*
- ***eof*** *filedescriptor*
- *Tcl also supports c-like "seek" and "tell" commands*

File Access Parameter

- r : Open the file for reading only; the file must already exist. This is the default value if access is not specified.
- r+: Open the file for both reading and writing; the file must already exist.
- w: Open the file for writing only. If it doesn't exist, create a new file.
- w+: Open the file for reading and writing. If it doesn't exist, create a new file.
- a: Open the file for writing only. The file must already exist, and the file is positioned so that new data is appended to the file.
- a+ : Open the file for reading and writing. If the file doesn't exist, create a new empty file. Set the initial access position to the end of the file.

File I/O – Example

```
proc fileRead {fname} {
        set fd [open $fname r]
        while {![eof $fd]} {
            set line [gets $fd]
            puts $line
        }
        close $fd
}
```

Tcl *seek* and *tell* Commands

- Seek command is used to change the access position for an open channel (file)
  - Syntax: seek channelId offset origin
  - Offset is an integer, origin is start/current/end
- Tell command returns the current access position for an open channel (file)
  - Syntax: tell <channelid>
- Note: <CR> and <LF> are included in count

Hexdump

- Hexa decimal dump of file can allow us to seen the content of file with control characters like <LF>(\n -0xd) and <CR> (\r - 0Xa) etc
- Hex Editor Neo – Download and install on your laptop
- Use this to view the content of a file
- Understand how cursor movement over file treats <LF> and <CR>

Examples of *seek* and *tell*

```
% set fd [open test.tcl r]
file1537c0
% tell $fd
0
% seek $fd 10 start
% tell $fd
10
```

Exercise

1. Open a file (input.txt) in a read mode
2. Open another file (output.txt) in Write mode
3. Copy the contents of input file to output file
4. Close both files – validate your program
5. Check out the cursor position while reading the content of a file
6. Move the cursor to the print and read it again

ARMSIM

File Processing & Embest Peripherals

ARM Assembler Directives

- Assembler directives are instructions to assembler
- Assembler directives are used for initializing memory, creating symbolic literal, etc
- Do not interleave Assembler instructions with Assembler directives
- It is good practice to have all the directives either at the top or at the bottom

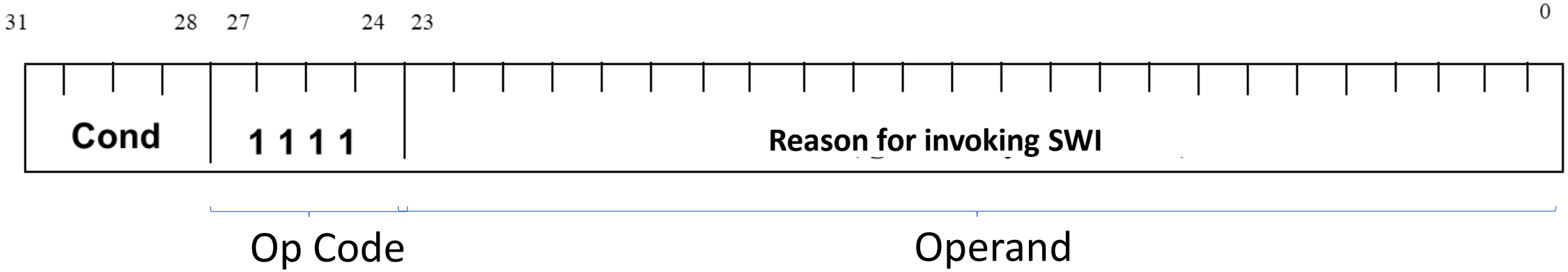
- Hello: **.asciz** "Hello Class\n“
- Address: **.word 0**
- **.equ** SEG_A, 0x80
- **.equ** Ten, 10
- **.set** counter, 1
- **.set** counter, counter+1
- **.align**

For a Comprehensive list of ARM Directives refer the following:

https://community.arm.com/processors/b/blog/posts/useful-assembler-directives-and-macros-for-the-gnu-assembler

Software Interrupt (SWI)

- SWI is one of the special ARM assembly instructions
- In general, SWI invocation behaves like a function call
- Invocation of SWI passes the control to Privileged System
- SWI is a single operand instruction; operand size is 24-bits in size
- The operand value informs the system the reason for invoking SWI
- ARM SWI is used to perform privileged functions
- ARMSim uses this mechanism in supporting two Plugins

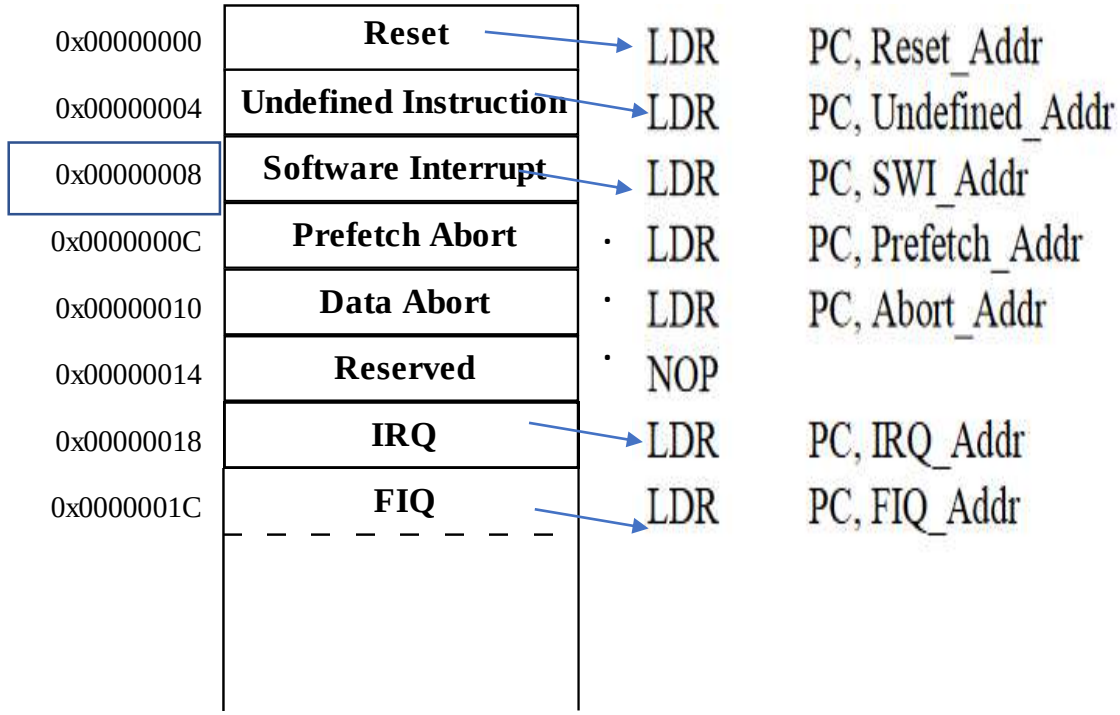


ARMSim and SWI

- ARMSim uses SWI to support File I/O and other Simulator specific Peripherals (Embest Plugin)
- Example:
  - Open File: SWI 0X66
  - Close File: SWI 0X68
- More on following slides

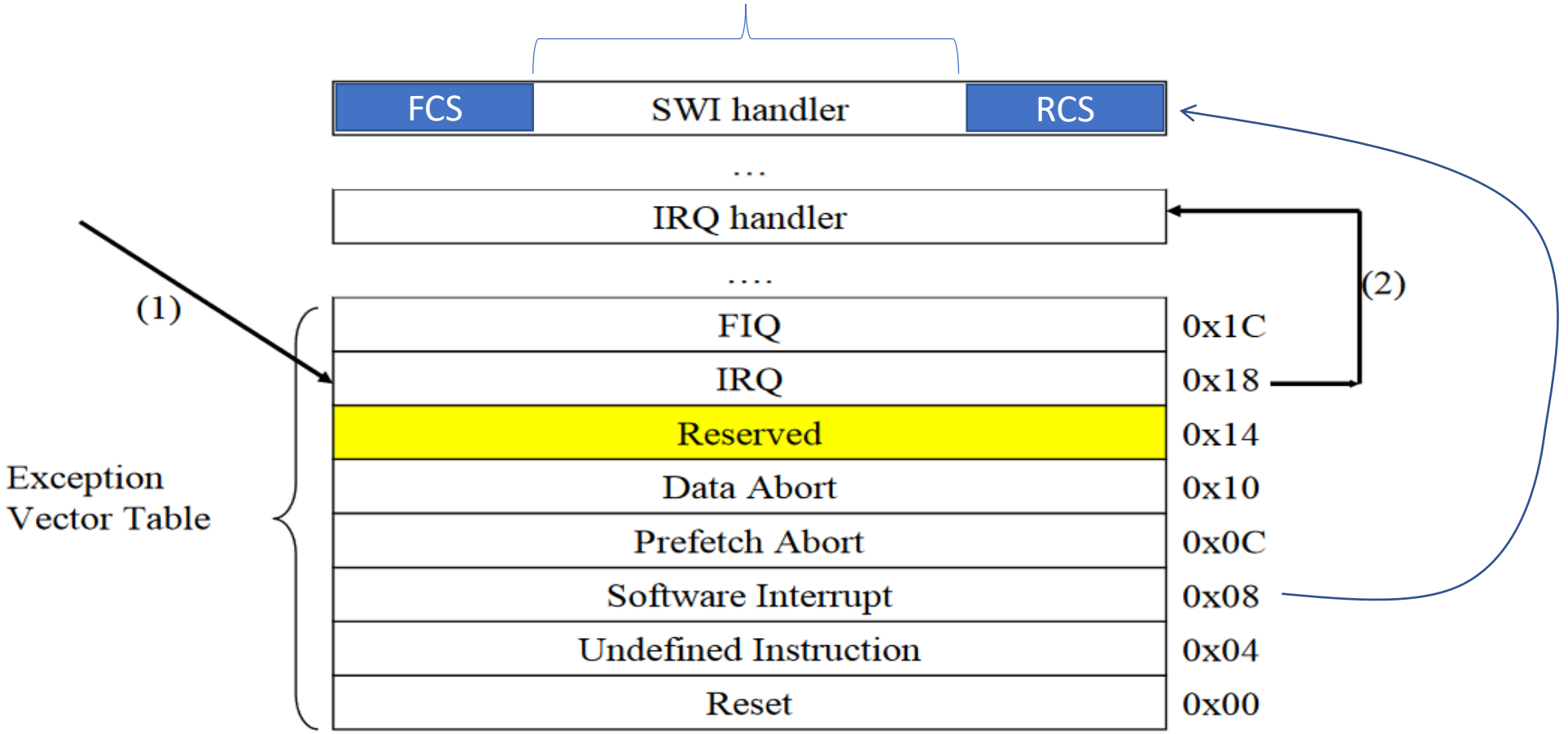
Handling SWI with Vector Table

- When an exception occurs**, the core:**
  - Copies CPSR into SPSR_<mode>
  - Sets appropriate CPSR bits (**especially the mode**)
  - Maps in appropriate banked registers
  - Stores the "*return address*" in LR_<mode>
  - Sets PC to vector address - 0x00000008
- To return, exception handler (sw) needs to:
  - Restore CPSR from SPSR_<mode>
  - **Restore PC from LR_<mode>**
  - **Leave data in designated registers (return values)**



SWI Handler

Codes handler for various values of operand, like file close, open etc



ARMsim* – File I/O Plugin Exceptions

Opcode	Description and Action	Inputs	Outputs	EQU
swi 0x00	Display Character on Stdout	r0: the character		SWI_PrChr
swi 0x02	Display String on Stdout	r0: address of a null terminated ASCII string	(see also 0x69 below)	
swi 0x11	Halt Execution			SWI_Exit
swi 0x12	Allocate Block of Memory on Heap	r0: block size in bytes	r0:address of block	SWI_MeAlloc
swi 0x13	Deallocate All Heap Blocks			SWI_DAlloc
swi 0x66	Open File (mode values in r1 are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0:file handle If the file does not open, a result of -1 is returned	SWI_Open
swi 0x68	Close File	r0: file handle		SWI_Close
swi 0x69	Write String to a File or to Stdout	r0: file handleor Stdout r1: address of a null terminated ASCII string		SWI_PrStr

Printing a Character & a string to STDOUT

```
• Hello: .asciz    "Hello Class!\n"
•        mov       r0, #'1'
•        swi       0x00          ; character in r0 to stdout
•        mov       r0,#':'
•        swi       0x00          ; character in r0 to stdout
•        ldr       r0, =Hello    ; load starting byte address
•        swi       0x02          ; string to stdout
•        Swi       0x11          ; Halt
```

File Handle/Descriptor Size is 1 Word;

FD 0 for STDIN, FD 1 for STDOUT, and FD 2 for STDERR

ARM SIM – Opening a File for Input/Output

Opcode	Description and Action	Inputs	Outputs	EQU
swi 0x00	Display Character on Stdout	r0: the character		SWI_PrChr
swi 0x02	Display String on Stdout	r0: address of a null terminated ASCII string	(see also 0x69 below)	
swi 0x11	Halt Execution			SWI_Exit
swi 0x12	Allocate Block of Memory on Heap	r0: block size in bytes	r0:address of block	SWI_MeAlloc
swi 0x13	Deallocate All Heap Blocks			SWI_DAlloc
swi 0x66	Open File (mode values in r1 are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0:file handle If the file does not open, a result of -1 is returned	SWI_Open
swi 0x68	Close File	r0: file handle		SWI_Close
swi 0x69	Write String to a File or to Stdout	r0: file handleor Stdout r1: address of a null terminated ASCII string		SWI_PrStr

1. FileName: .asciz “test.s"
2. .align
3. FileHandle: .word 0
4. ldr r0,=FileName
5. mov r1,#0
6. swi 0X66
7. ldr r1,=FileHandle
8. str r0, [r1]
9. swi 0x11

Opening a file for output is done by changing #0 to #1 in line 5

ARM SIM – Reading a string from a file

Opcode	Description and Action	Inputs	Outputs	EQU
swi 0x6a	Read String from a File	r0: file handle r1: destination address r2: max bytes to store	r0: number of bytes stored	SWI_RdStr
swi 0x6b	Write Integer to a File	r0: file handle r1: integer		SWI_PrInt
swi 0x6c	Read Integer from a File	r0: file handle	r0: the integer	SWI_RdInt
swi 0x6d	Get the current time (ticks)		r0: the number of ticks (milliseconds)	SWI_Timer

1. **ldr r0,=FileHandle**
2. **ldr r0, [r0]**
3. **ldr r1,=CharArray**
4. **mov r2,#80**
5. **swi 0x6a**
6. **bcs ReadError**
7. **...**
8. **FileHandle: .word 0**
9. **CharArray: .skip 80**

Error handling is critical to developing good program

ARM SIM – Writing a string to a file

Opcode	Description and Action	Inputs	Outputs	EQU
`swi 0x00`	Display Character on Stdout	r0: the character		`SWI_PrChr`
`swi 0x02`	Display String on Stdout	r0: address of a null terminated ASCII string	(see also 0x69 below)	
`swi 0x11`	Halt Execution			`SWI_Exit`
`swi 0x12`	Allocate Block of Memory on Heap	r0: block size in bytes	r0:address of block	`SWI_MeAlloc`
`swi 0x13`	Deallocate All Heap Blocks			`SWI_DAlloc`
`swi 0x66`	Open File (mode values in r1 are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0:file handle If the file does not open, a result of -1 is returned	`SWI_Open`
`swi 0x68`	Close File	r0: file handle		`SWI_Close`
`swi 0x69`	Write String to a File or to Stdout	r0: file handleor Stdout r1: address of a null terminated ASCII string		`SWI_PrStr`

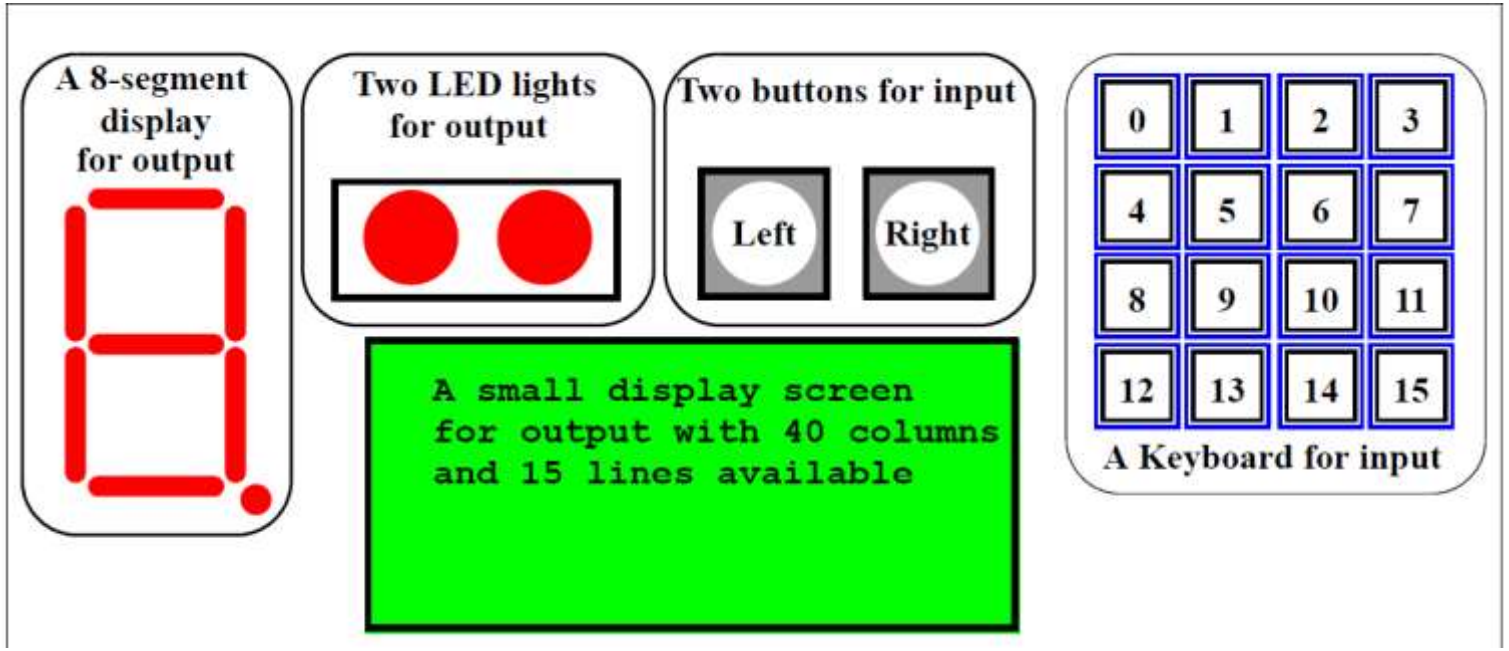
1. **ldr r0,=OutFileHandle**
2. **ldr r0,[r0]**
3. **ldr r1,=TextString**
4. **swi 0x69**
5. **bcs WriteError**
6. **…**
7. **TextString: .asciz “Hello!\n "**

The file handle for STDOUT is #1. No need for Open for STDOUT, STDIN and STDERR

File Close

```
load the file handle
ldr       r0,=InFileHandle
ldr       r0,[r0]
swi       SWI_Close
```

ARMSIM – Working with Embest Plugin



- Embest plugin includes a number of simple I/O devices
  - 8-segment display, LED lights, Input toggle buttons, and a numeric key board
  - SWI implementation supports exception handling to work with these devices

ARMSIM – Embest LED – Exception Code

`swi 0x201`	Light up the two LEDs .	r0: the LED Pattern, where: Left LED on = `0x02` Right LED on = `0x01` Both LEDs on = `0x03` (i.e. the bits in position 0 and 1 of r0 must each be set to 1 appropriately)	Either the left LED is on, or the right, or both

1. **mov r0,#0x02**
2. **swi 0x201 ; left LED on**
3. **mov r0,#0x01**
4. **swi 0x201 ; right LED on**
5. **mov r0,#0x03**
6. **swi 0x201 ; both LEDs on**
7. **mov r0,#0x00**
8. **swi 0x201 ; both LEDs on**

1. Read and understand the SWI Exception tables for both I/O and Embest Plugin
2. No need to remember the exception or code

Exercise

- Read Section 9 of ARMSim User Guide with objective to write code for **Embest Board Plugin**
- **This section will not be taught in the class!**