CG3002

Embedded Systems Design Project

Lecture 3

Communications & Firmware

Peh Li Shiuan, Professor, Computer Science
peh@nus.edu.sg

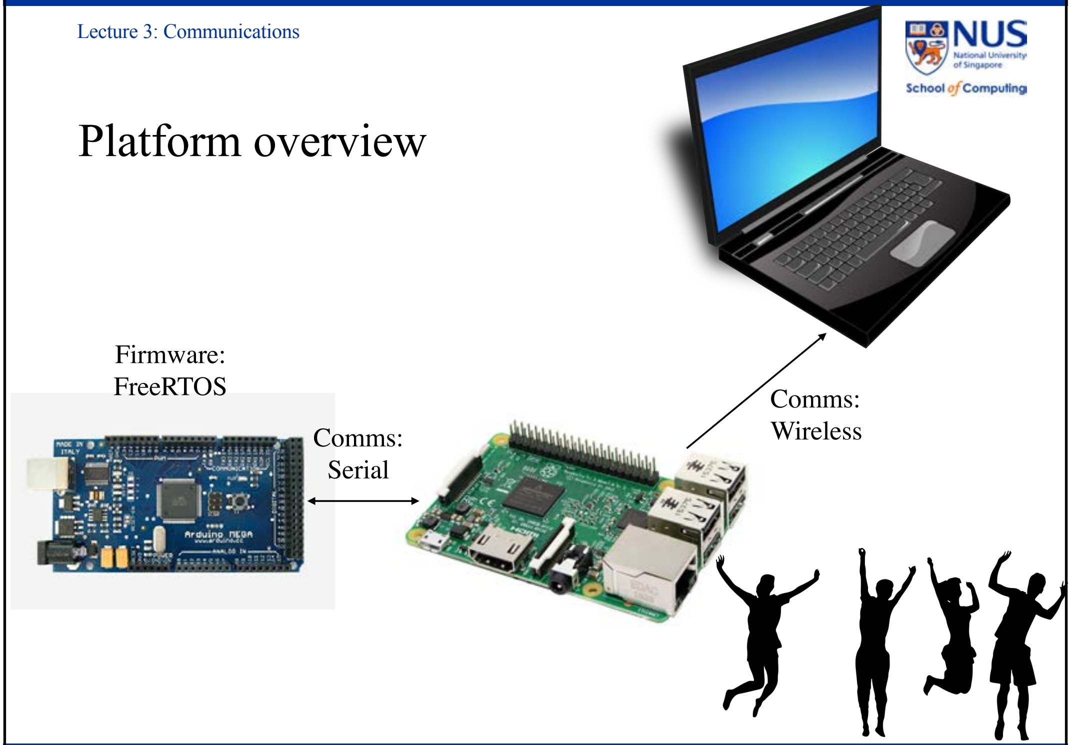
National University of Singapore
School of Computing

[Slides adapted from Dr Colin Tan's previous CG3002 slides]

Communications & Firmware

- **Within the system**
  - Serial communications between Pi and Mega
- **Beyond the system**
  - Secure wireless communications between Pi and server
- **Real-time operating system**
  - FreeRTOS on Arduino Mega

Platform overview

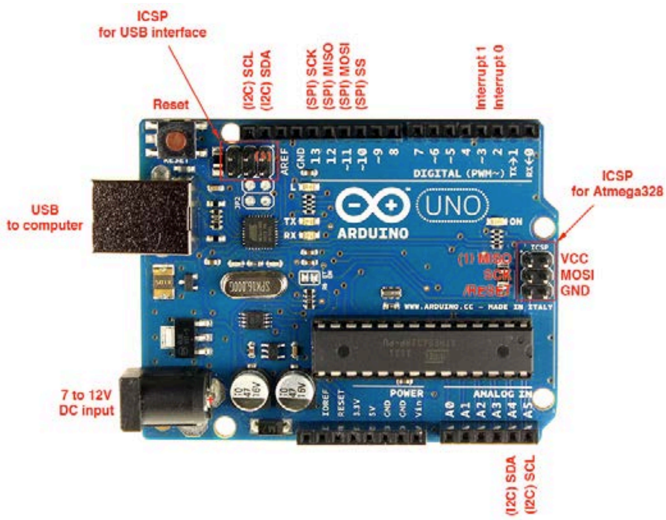


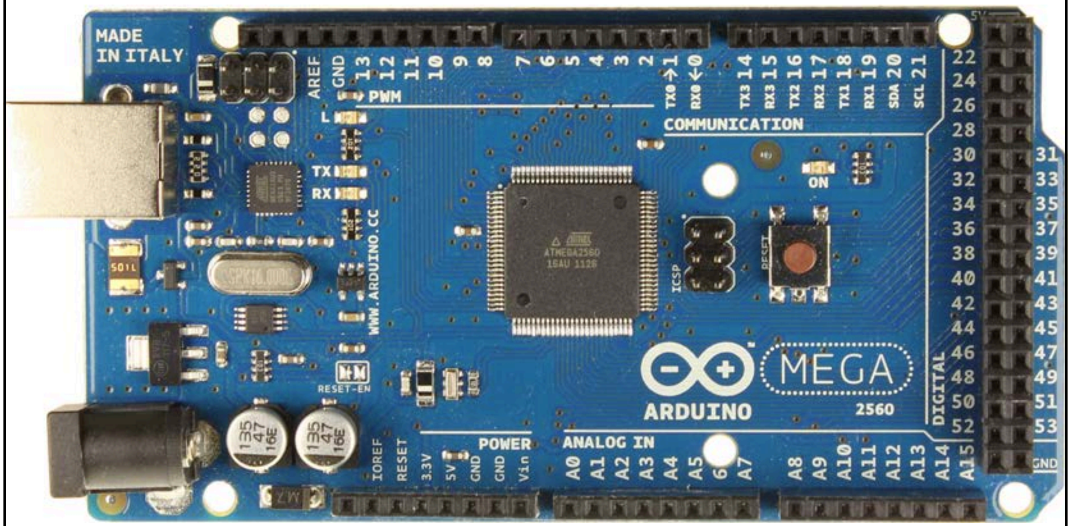
ARDUINO MEGA-RASPBERRY PI3 SERIAL COMMUNICATIONS

What's In It For You?

- **By the end of this session, you will:**
  - Understand the Arduino Mega architecture, and how it is different from the Arduino Uno you are familiar with from CG2271.
  - Learn how to program the serial ports on the Mega (USART) and on the RPi3 (UART).

The Arduino Mega vs Arduino Uno





Arduino Programming / Serial Communication

SERIAL PROGRAMMING

Serial Communication

- **Motivation:**
  - RPi has awesome memory and CPU speed.

	RPi	Arduino Mega
CPU Speed	700MHz	16MHz
RAM/Flash	512MB	256KB Flash, 8KB RAM

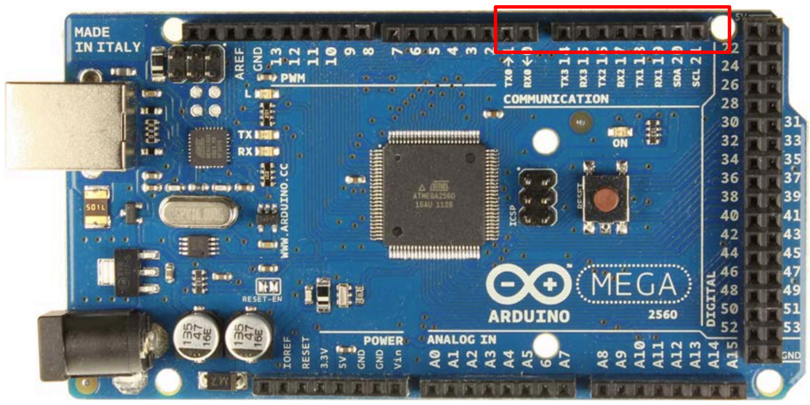
  - BUT I/O options on the RPi are somewhat limited:

I/O Option	RPi	Arduino Mega
GPIO pins	28	54
UART	1	4
I2C	1	1
SPI	1	1
Analog Inputs	-	16
Analog Outputs	-	15

Lecture 3: Communications

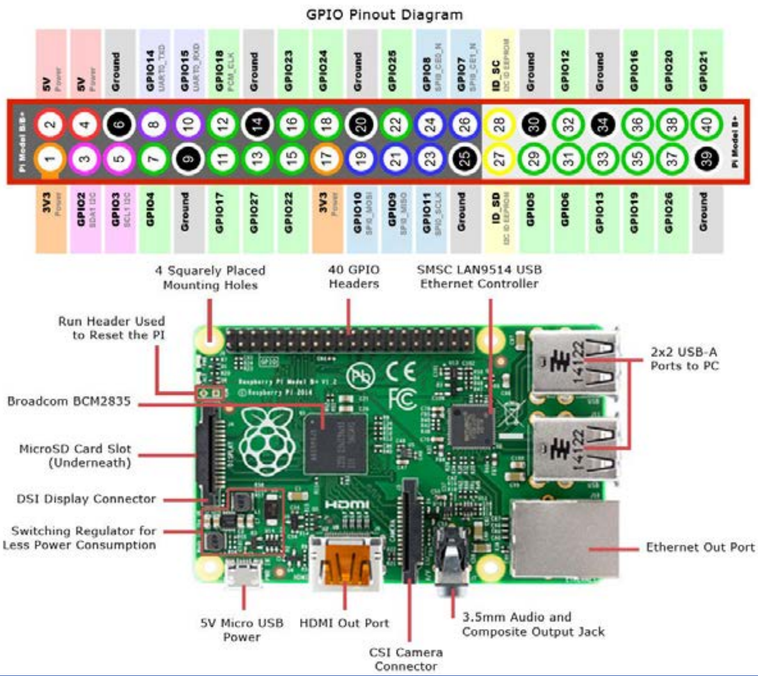
Serial Programming on the Mega

- **The Arduino language supports several primitives to access the serial ports on the Mega.**



The Arduino USART Pins

- **Port 0:**
  - Pin 0 – RX
  - Pin 1 – TX
- **Port 1:**
  - Pin 19 – RX
  - Pin 18 - TX
- **Port 2:**
  - Pin 17 – RX
  - Pin 16 – TX
- **Port 3:**
  - Pin 15 – RX
  - Pin 14 - TX



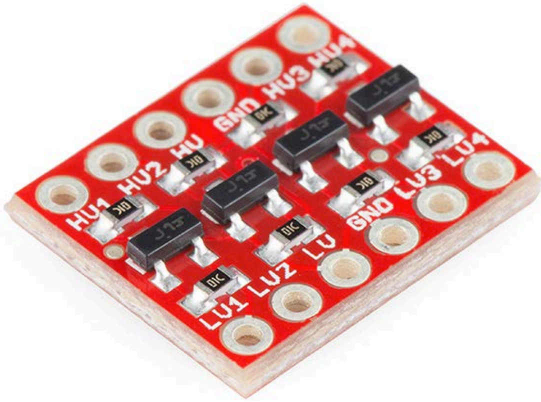
UART on the Raspberry Pi

- **There is only one UART port on the Raspberry Pi:**
  - RX is on pin 10
  - TX is on pin 8
- **YOU CANNOT CONNECT THE MEGA'S USART PINS DIRECTLY TO THE PI'S UART PINS!!!**
  - Mega: 5v Pi: 3.3v
  - If you connect them directly you will destroy the Pi.
  - Use level shifter (See next slide)
- **YOU ALSO CANNOT CONNECT THE MEGA OR PI USART/UART PINS TO A PC'S RS232 PORT!!**

Level Shifting

- **There are several ways to shift from 5v to 3.3v:**
  - Voltage Divider
  - Level shifter (see next slide).
- **Generally there is no need to shift from 3.3v to 5v**
  - 3.3v is above the ~2.7v "high" level on the Mega.
- **The next slide shows the Sparkfun Bidirectional Level Converter**

Level Shifting.



- **Pin usage:**
  - Connect 5v pin from Mega to HV.
  - Connect 3.3v reference voltage (from Mega or Rpi) to LV.
  - Connect HV1,HV2, HV3,HV4 to Mega.
  - Connect LV1, LV2, LV3, LV4 to Rpi.

Arduino Serial Communication Primitives

- **In each of the following, *X={1,2,3}* designates the port number. E.g. for serial port 2 (pins 15 and 16), we would have Serial2.begin, Serial2.write, etc.**
- **Accessing the serial port:**
  - Serial*X*.begin(*baud rate*)
    - ✓**Initializes the port to the requested baud rate (e.g. 9600 bps), default 8N1 configuration.**
  - Serial*X*.available()
    - ✓**Returns the number of bytes available for reading from the serial buffer.**
  - Serial*X*.read()
    - ✓**Reads the next available byte from the buffer.**

Arduino Serial Communication Primitives

- Serial*X*.write(val)
  - ✓**Writes a numeric value to the serial port. Returns number of bytes written.**
- Serial*X*.write(str)
  - ✓**Writes a string to the serial port. Returns number of bytes written.**
- Serial*X*.write(buf, len)
  - ✓**Writes *len* bytes from buffer *buf* to the serial port. Returns number of bytes written.**

Arduino Serial Communication Primitives

- Serial*X*.print(str)
  - ✓**Prints *str* as a human-understandable string, including numbers.**
  - ✓**Understands escape codes like \t (tab) or \n (newline).**
- Serial*X*.println(str)
  - ✓**Like print but automatically inserts a newline at the end.**

- **Note: X={1,2,3} denoting serial ports 1 to 3. Serial port 0 is accessed using the Serial class as usual.**

Arduino Serial Example

```
/*
 Mega multple serial test

 Receives from the main serial port, sends to the others.
 Receives from serial port 1, sends to the main serial (Serial 0).

 This example works only on the Arduino Mega

 The circuit:
 * Any serial device attached to Serial port 1
 * Serial monitor open on Serial port 0:

 created 30 Dec. 2008
 modified 20 May 2012
 by Tom Igoe & Jed Roach

 This example code is in the public domain.

 */
```


Arduino Serial Example

```
void setup() {
  // initialize both serial ports:
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop() {
  // read from port 1, send to port 0:
  if (Serial1.available()) {
    int inByte = Serial1.read();
    Serial.write(inByte);
  }

  // read from port 0, send to port 1:
  if (Serial.available()) {
    int inByte = Serial.read();
    Serial1.write(inByte);
  }
}
```

Serial Programming on the RPi

- **To do serial programming on the Pi using Python you need to install the PySerial package.**
  - `sudo apt-get install python-serial`
- **The serial primitives are simple:**
  - Open the serial port.
  - Use read/write to communicate

Serial Programming on the RPi

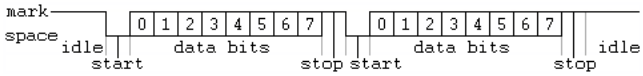
```
import serial

def readlineCR(port)
     rv="”
     while True:
             ch=port.read()
             rv+=ch
             if ch=='\r’ or ch==‘’:
                     return rv

port=serial.Serial(“/dev/ttyAMA0”, baudrate=115200, timeout=3.0)

while(True)
     port.write(“\r\nSay something: “)
     rcv=readlineCR(port)
     port.write(“\r\nYou sent:” +rcv)
```

Arduino Programming / Serial Communication

SERIAL PROTOCOL DESIGN

Three things to do:

- **Decide on physical connection:**
  - Only one UART port on the RPi. Easy enough.
  - Which USART to use on the Mega?
    - ✓**Don't use USART 0 – This is the USB connection to your PC.**
- **Decide on bit-level protocol:**
  - Decide baud rate.
  - Decide data length.
  - Decide # of parity bits.
  - Decide # of stop bits.
  - Standard is 9600 8N1.



Three Things to Do:

- **Decide a communication protocol:**
- **Some options to be explored today:**
  - Periodic push by Mega to RPi.
  - Periodic poll by RPi to Mega.
- **Each have their advantages/disadvantages.**
  - Decide what's best for your application.

Arduino Programming / Serial Communication

BUILDING A PROTOCOL

Assign an ID to each device

- **You need to be able to identify sensors (actuators) to read from (send data to).**

Device ID	Device
0	Sonar 1
1	Sonar 2
2	Touch Sensor 1
3	Touch Sensor 2
4	Buzzer
5	Tactile feedback motor
...	...

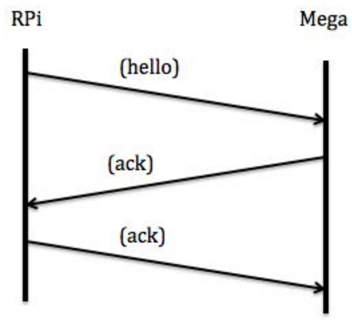
Create Packet Types

- **So both sides know what sort of packets are being sent (and the appropriate response)**

Packet Type	Packet Code
ACK	0
NAK	1
Hello	2
Read	3
Write	4
Data Response	5
...	...

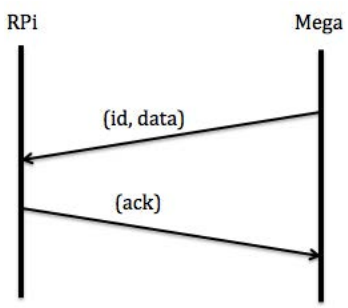
Bootup 3-way Handshake

- **Objective:**
  - ▪So RPi and Mega both know that each is ready to communicate.



  - ▪Do this at the very start of your programs on both RPi and Mega.

Periodic Push By Mega

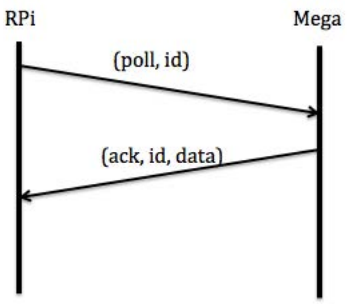


- **Mega sends data whenever it is available.**
- **RPi monitors and buffers data as it comes in.**

  +Mega sends data whenever it is available.

  -RPi needs to buffer incoming data.

Periodic Poll by RPi



- **Mega waits for poll packets from RPi.**
- **RPi requests data when it needs it.**

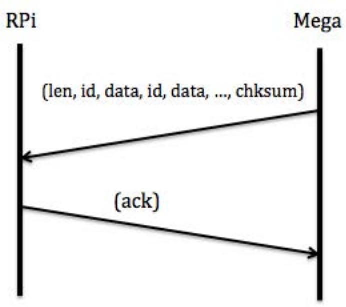
  +RPi decides when it needs the data and sends poll packet.

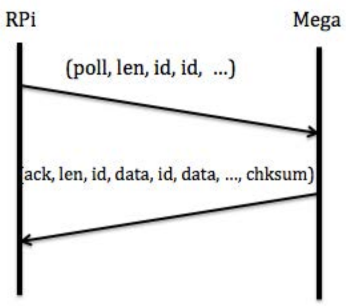
  -If RPi doesn't poll often enough, may lose data on Mega (Mega has small memory).

Lecture 3: Communications

Sending Chunks of Data

- **Polling/Pushing individual sensor data can be expensive.**
- **Might be better (??) to send data for multiple sensors.**





Finding Checksums

- **Checksums are used to check that data is received correctly.**
- **Sender side:**
  - Compute checksum

    `checksum = b1 XOR b2 XOR b3 XOR b4 XOR…`
  - Attach to end of packet.
- **Receiver side:**
  - Compute checksum using data in packet, except checksum.
  - Compare against attached checksum.
  - If equal, reply with ACK, else reply with NAK.

Finding Checksums

- **For the most part:**
  - Serial comms is reliable.
  - Hence we don't normally compute checksums (or send ACK for that matter).
- **However:**
  - Your set up is not going to be perfect. (headers and pins while dancing!)
  - If you are sending relatively large amounts of data, higher chance of errors.

Serializing Structures

- **Serializing: Converting a structure into a stream of bytes.**
  - Get a pointer to the structure.
  - Copy into an array of char.
  - May want to include information on packet length and checksum.

Serializing Structures

```
typedef struct con
{
  unsigned char devCode;
  double maxValue;
  double minValue;

} TConfigPacket;

void sendConfig()
{
    TConfigPacket cfg;
    char buffer[64];
    cfg.devCode=deviceCode;
    cfg.minValue = minValue;
    cfg.maxValue = maxValue;
    unsigned len = serialize(buffer, &cfg, sizeof(cfg));
}
```

Serializing Structures

```
unsigned int serialize(char *buf, void *p, size_t size)
{
  char checksum = 0;
  buf[0]=size;
  memcpy(buf+1, p, size);
  for(int i=1; i<=size; i++)
  {
    checksum ^= buf[i];
  }
  buf[size+1]=checksum;
  return size+2;
}

void sendSerialData(char *buffer, int len)
{
  for(int i=0; i<len; i++)
    Serial1.write(buffer[i]);
```


Deserializing Structures

- **Deserialize: Convert a stream of bytes back to structures.**
  - Get a pointer to the structure.
  - Copy buffer of bytes to that pointer:
    - ✓**May need to remove packet length and compute checksums first.**

Deserializing Structures

```
Void readConfig()
{
    char buffer[MAX_BUF_LEN];
    int len;
    TConfigPacket cfg;

    readSerial(buffer, &len);
    deserialize(&cfg, len);

    // Process cfg.
        …
}
```

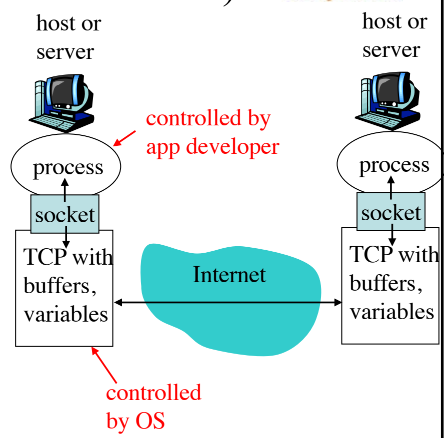

Deserializing Structures

```
unsigned int deserialize(void *p, char *buf)
{
  size_t size = buf[0];
  char checksum = 0;

  for(int i=1; i<=size; i++)
    checksum ^= buf[i];

  if(checksum == buf[size+1])
  {
    memcpy(p, buf+1, size);
    return PACKET_OK;
  }
  else
  {
    printf("CHECKSUM ERROR\n");
    return PACKET_BAD_CHECKSUM;
  }
```

SECURE WIRELESS COMMUNICATIONS BETWEEN SYSTEM AND SERVER

Introduction to sockets (Flashback to CS2105)

- **Interface between the application layer and transport layer**
- **OS-controlled interface (a "door") into which application process can both send and receive messages**
- **Addressing**
  - Host address + process identifier
  - Eg. IP address + port number
    - ✓**Eg HTTP – port 80, SMTP – port 25, ftp – port 21**



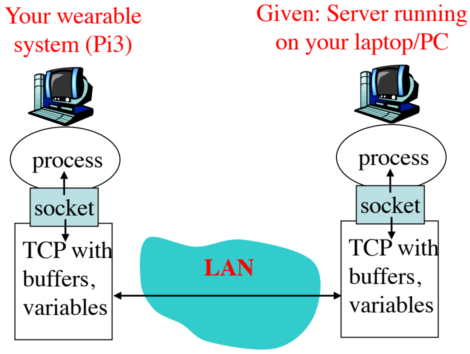
Socket: endpoint for comm. A combination of host IP address and port number.

[Slide from CS3103, Dr Anand Bhojan]

Image Source: Chapter 2 - Kurose& Ross

Sockets in CG3002

- **Interface between the application layer and transport layer**
- **OS-controlled interface (a "door") into which application process can both send and receive messages**

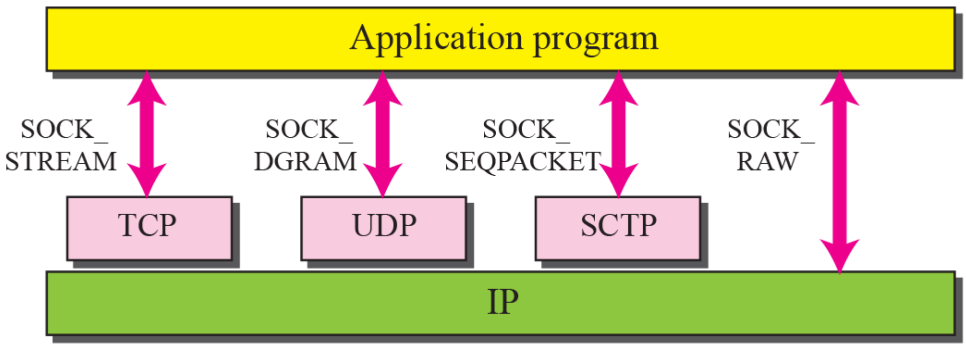


- **Addressing**
  - Host address + process identifier
  - Eg. IP address + port number
    - ✓**IP address: IP address of server on LAN**
    - ✓**Port number: You define**

Socket: endpoint for comm. A combination of host IP address and port number.

Image Source: Chapter 2 - Kurose& Ross

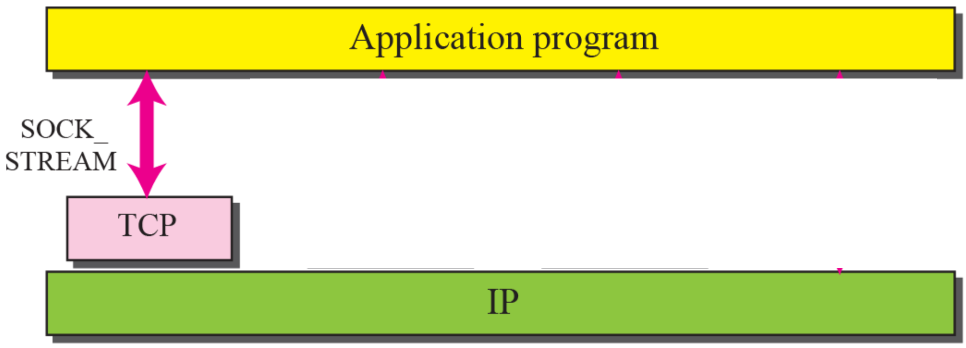
Socket Types



- Types of transport service via socket:
  - unreliable datagram
  - reliable, byte stream-oriented
- Lower level protocols and network interfaces can be accessed through 'RAW Socket'.
- The new SCTP socket provides multiple types of service

[Slide from CS3103]

Socket Types [in CG3002]



- TCP: **reliable, byte stream-oriented**
- Server process must be running first
- Server must have created socket (door) that welcomes client's contact
- Client creates client-side local TCP socket specifying IP address, port number of server process to bind to the server
- When client creates socket: client TCP establishes connection to server TCP

Socket API in Python (on Pi3 client – You!)

- **TCP socket: SOCK_STREAM**
- **Server IP address and socket number**
  - Depends on the network you run

- **Socket library:** **import socket**

- **Creating a socket:** **sock = socket.socket(…)**

- **Connecting to a socket:** **sock.connect()**
- **Sending using a socket:** **sock.sendall()**
- **Receiving from a socket:** **sock.recv()**

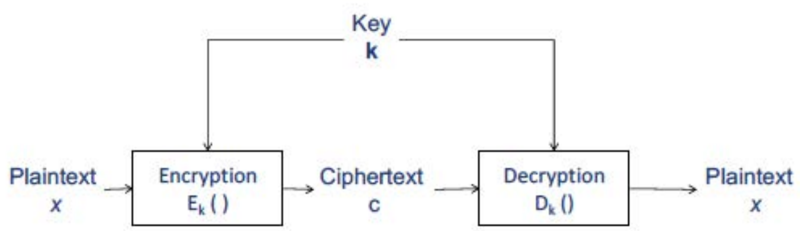
- **Closing a connection:** **sock.close()**

Socket API in Python (on server – provided, run on your laptop/PC)

- **Socket library:** **import socket**
- **Creating a socket:** **sock = socket.socket(…)**
- **Binds server address to socket:** **sock.bind(server_address)**
- **Listens to the socket for client messages:** **sock.listen()**
- **Accepts client connection:** **sock.accept()**
- **Receives data from socket:** **data = connection.recv()**
- **Sends status after handshaking:** **connection.send(status.encode())**

Encryption: Flashback from CS2103

An *encryption scheme* (also known as *cipher*) consists of two algorithms: ***encryption and decryption***



Correctness: For any plaintext x and key k,
$D_k(E_k(x)) = x$

Security: From the ciphertexts, it is "difficult" to derive useful information of the key k, and the plaintext x. The ciphertexts should resemble sequences of random bytes. (There are many refined formulations of security requirements, e.g. semantic security. In this module, we will not go into details).

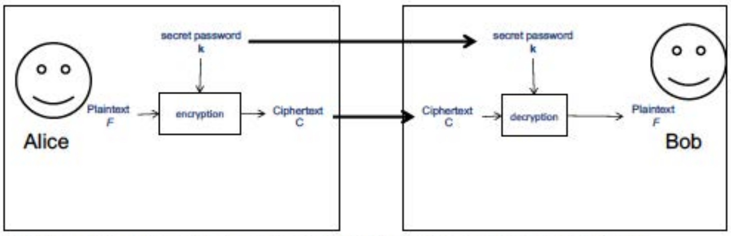
[Slide from CS2103, Prof. Chang Ee Chien]

Encryption: Flashback from CS2103

A simple application scenario.

Alice has a large file ***F*** (say info on her bank accounts and financial transactions in Excel). She "encrypted" the file ***F*** using winzip with a password "13j8d7wjnd" and obtained the ciphertext ***C***. Next, she called Bob to tell him the 10-character password, and subsequently, she sent the ciphertext to Bob via email attachment. Later, Bob received ***C*** and decrypted the ciphertext with the password to obtain the plaintext ***F***.

Anyone who has obtained C, without knowing the password, is unable to get any information on ***F***. Although C indeed contains info of F, the information is "hidden". To someone who doesn't know the secret, C is just a sequence of random bits.



Remark: Winzip is ***not*** an encryption scheme. It is an application that employs standard encryption schemes such as AES.

[Slide from CS2103, Prof. Chang Ee Chien]

Why do we need encryption in our Dance Dance system?

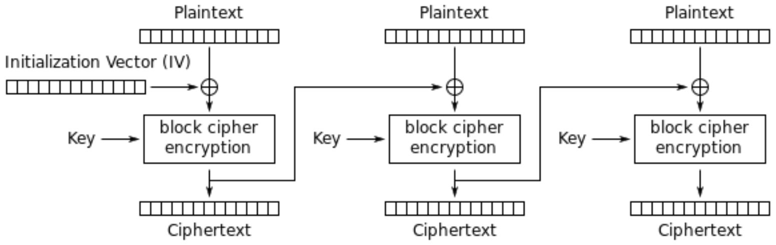
- **Open wireless networks**
- **Personal data privacy**
- **Authentication**

- **Key**
  - Your choice – Tell us during evaluation so we can decrypt
  - Don't store it in plaintext ☺

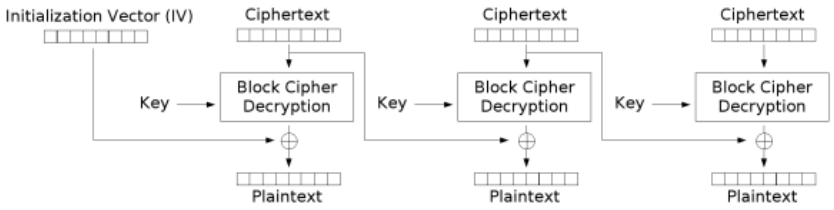
Using the AES Encryption scheme

- **AES:** Popular and widely adopted symmetric encryption standard
- **Crypto Cipher library in python**
  - from Crypto.Cipher import AES
- **AES**
  - mode CBC
  - Base 64
  - Secret key
  - Initial value: random
  - Padding

AES



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Authentication: Server (provided)

- decodedMSG = base64.b64decode(encodedMsg)
- iv = decodedMSG[:16]
- cipher = AES.new(secret_key,AES.MODE_CBC,iv)
- decryptedText = cipher.decrypt(decodedMSG[16:]).strip()

Authentication: Client (You! ☺)

- iv = Random.new().read(AES.block_size)
- cipher = AES.new(secret_key,AES.MODE_CBC,iv)
- encoded = base64.b64encode(iv + cipher.encrypt(msg))

Python code provided (on IVLE)

- **server_auth.py**
- **sample_auth_server.py**

  - Server expects a secret key
  - Server expects message string of this format: '#action | voltage | current | power | cumulativepower|'
  - AES expects base64 encoded message of 128-bits initial value + message
  - AES expects padding

- **Tips**
  - Test your wireless comms client on your laptop first, localhost
  - Use a wireless hotspot so Pi and laptop are on the same wireless LAN, rather than NUS WiFi
  - Test socket comms and encryption/decryption separately; and serial comms separately first

Python code provided (on IVLE)

- **Log file generated by sample_auth_server.py helps you debug your system**

- **log.csv:**

  timestamp,action,goal,time_delta,correct,voltage,current,power,cumpower

  1503804719.855472,windowcleaning,frontback,39.464357137680054,False,6.543525192117988,1.0289350400167643,90.97080238196381,4.724291640460015

  1503815162.881185,jumping,jumping,51.3844780921936,True,9.853852457869436,2.793253964411687,0.9615081914457146,6.628246015246175

- **performanceMetrics.py**

FIRMWARE

Real-time Operating System

- **Why RTOS?**
- **FreeRTOS port on Arduino Mega**
- **Tasks**
- **Scheduling**

FreeRTOS Task Creation (Flashback from CG2271)

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```

Pointer to the function representing the entry point of the new task

[Slides from CG2271, Prof. Tulika Mitra]

FreeRTOS Task Creation

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```

Human-readable name of the task being created
Useful during debugging

[Slides from CG2271..]

FreeRTOS Task Creation

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```

How many stack words must be reserved for the task stack

[Slides from CG2271..]

FreeRTOS Task Creation


```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```


This pointer will be passed on to the task entry point
Commonly used to point at a shared memory structure holding tasks parameters

[Slides from CG2271..]

FreeRTOS Task Creation

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```

Baseline priority of the new task expressed as a positive integer

[Slides from CG2271..]

Lecture 3: Communications

FreeRTOS Task Creation

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```

Pointer to the task handle that should be used in the future to refer to the new task

[Slides from CG2271..]

FreeRTOS Task Creation

```
portBASE_TYPE xTaskCreate(
    pdTASK_CODE pvTaskCode,
    const char * const pcName,
    unsigned short usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pvCreatedTask);
```

Return status code: If the value is pdPASS, task creation is successful, error otherwise

[Slides from CG2271..]

MISCELLANEOUS TIPS…

Debugging on the Mega

- **Debugging is a challenge on the Mega. You can however create your own version of printf that prints to the USART0/USB.**
  - Code on next slide shows how to do this.
  - Use it like a normal printf statement, e.g. dprintf(“Hello %s”, name);
  - Don’t print strings that are too long, nor print too frequently. The Mega’s USART buffer is very small.

Debugging on the Mega

```
char debugBuffer[1024];
void debugPrint(const char *str)
{
		Serial.println(str);
		Serial.flush();
}

void dprintf(const char *fmt, ...)
{
	va_list argptr;
	va_start(argptr, fmt);
	vsprintf(debugBuffer, fmt, argptr);
	va_end(argptr);
	debugPrint(debugBuffer);
}
```

Optimizing Memory

- **The recommended sensor sets have been tested to work on FreeRTOS running on the Mega.**
- **HOWEVER you may still find your Mega hanging when you write your app:**
  - This is most likely due to running out of memory.
- **You will need to trim memory usage as much as possible.**
  - One place: Look at HardwareSerial.cpp and see which serial ports you can comment out. ☺

HardwareSerial.cpp Example

```
#if defined(USART1_RX_vect)
  void serialEvent1() __attribute__((weak));
  void serialEvent1() {}
  #define serialEvent1_implemented
  SIGNAL(USART1_RX_vect)
  {
    unsigned char c = UDR1;
    store_char(c, &rx_buffer1);
  }
#elif defined(SIG_USART1_RECV)
  #error SIG_USART1_RECV
#endif
```

Finding Free Memory

- **You can check the amount of free RAM left using this code:**

```
int freeRam () {
        extern int __heap_start, *__brkval;
        int v;
        return (int) &v -
(__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

- **Use this together with dprintf to see if you've run out of memory.**
  - Note: You are likely to still see a small amount of free memory, ~1200 bytes, at the point your app crashes.

DO NOT USE DYNAMIC MEMORY!

- **The Arduino Library provides you with "new" and "delete" calls.**
- **DO NOT USE THESE!!!**
  - Unpredictable timing.
  - Free-space fragmentation.
- **In particular DO NOT USE THE STRING OBJECT!**
  - Very tempting. Let you do things like:

```
String str="The value we computed is " + String(value);
Serial.print(str);
```

  - The String object only allocates, never de-allocates!
    - ✓**You will run out of space very fast.**

Some Final Tips

- **Have a separate high to mid-level priority task to handle serial communications.**
  - Should be obvious, but do this on BOTH sides.
- **Use the SLEEP function (e.g. OSSleep in ArdOS) to periodically poll/send data.**
- **Use synchronization mechanisms (e.g. semaphores and globals, or queues) to coordinate tasks that receive sensor data and tasks that use them.**

Announcements

- **Next lecture is a lecture by Prof. Wang Ye on machine learning basics for activity detection**
- **Thursday's lab session starting first lab on Aug 31st (@DSA-lab)**
- **TAs:**
  - Hardware – Ahmad, Boyd and Yuan Ren
  - Comms/Firmware – Abdelhak and Ayush
  - Software --- Dania and Zhao Na