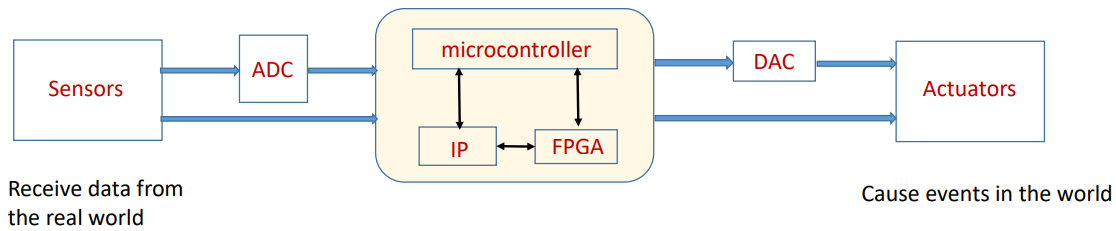
**Embedded Systems**: **What are Embedded Systems?**: Computer-based systems that do not appear to be computers – complexity is hidden from the user. • They interact with users via simple interface They interact with another device(flash drive). **Generic Embedded System Structure**: (below). **Efficient Rules**: Price, Performance, Power, Maintainability, Functionality, Size, Time-to-market, Safety .**Operating systems**: Manages other programs, allows many programs to be executed together, incorporates a nice user interface, needs processing power and memory, slows down the system, makes development easier. **User <-> Application <-> Operating System <-> Hardware**. **Embedded Linux systems**: Embedded systems tend to be application-specific • Perform one task or set of related tasks. Higher design efficiency is possible. **Embedded Operating Systems**: It has basic components of an operating system: Process management • Memory management • File management • I/O management • Network management. **Main characteristics** • Configurable • Real-time. 

**Embedded Linux**: **Basic Linux commands**: **File Handling “mkdir”** – make directories, **“ls”**: list directory contents (“ls - l” show more details such as permissions, ownership, time stamp was file being made). **“cd”** – changes directories. **“pwd”** – print name of current working directory. **“vim”** – vi Improved, a programmers text editor • **“cp”** – copy files and directories • Usage: cp [OPTION] SOURCE DEST. **“mv”** – move (rename) files. • **“rm”** – remove files or directories. **Basic Linux commands:** Text Processing:  **“cat”** – concatenate files and print on the standard output (• cat –n file1.txt shows number line. **“echo”** – display a line of text • **“grep”** – print lines matching a pattern. Ex. ifconfig | grep ‘inet addr: ’ **“wc”** – print the number of newlines, words, and bytes in files sort – sort lines of text files (by Ascii, Ascending order) (sort –r” descending order) . **Permission** Level: “r” means “read only” permission • “w” means “write” permission • “x” means “execute” permission. rwx-rw-r– means owner has all three permissions, group has read and write, others have only read permission Read permission -4, write-2, execute -1 • ex, rwx-rw-r- = 764. **Basic Linux commands**: System Administration: **“chmod”** – change file access permission chmod 744 calculate.sh “passwd” – update a user’s authentication tokens, **“who”** – show who is logged on, • **“su”** – change user ID or become super-user.  **Basic Linux commands**: • **“scp”** – secure copy (remote file copy program) • Scp copies files between hosts on a network • Usage: scp [OPTIONs] [[user]@host1:file1] [[user]@host2:file2]. **Cross-compilation: arm-linux-gnueabihf-gcc main.c -o helloworld. Difference from Linux systems: Embedded Linux: Purpose of using Linux kernel is to perform particular function. Desktop Linux: Purpose of using Linux kernel to many works for user. Basic C programming: Elements of a C program**: **Why C**? Fast execution • Easy memory management programming • Bit operation • But a bit complex concepts of pointer, type conversion and memory allocation. **What is gcc?** gcc • stands for GNU C/C++ Compiler local compilation “gcc -Wall test.c -o test”. **How to optimize a program by using gcc?** -O, -O1, -O2, -O3, -O0, -Os • Various levels of optimization of the code • -O1 to -O3 are various degrees of optimization targeted for speed • If -O is added, then the code size is considered • -O0 means “no optimization” • -Os targets generated code size (forces not to use optimizations resulting in bigger code) **Makefile**- Makefile describes to the make command that how to compile the program. Provide a way for separate compilation • Describe the dependencies among the project files • allows us to only compile those that have changed and the modules that depend upon them. “make –f filename“ **gdb:** Debugger 1. compile program • “gcc –g test.c –o test” • 2. start GDB • “gdb test” • 3. set breaking point • break main • 4. run program • “run” • 5. program execution continue to the next source line : “next” • 6. resume program execution : “continue”. GDB Commands • **list(l)**：list source code • **break(b)** function • break(b) line • break(b) file: line • break (b) line if condition • e.g., break 6 if i=10. • **Info break** : check all possible breakpoints • **delete** [breakpoint number]: delete # breakpoint • Without arguments deletes all breakpoints • **run(r)**: run program • **next(n)**: program execution continue to the next source line, omitting function calls • **step(s):** program execution continue to next source line, going into function calls • **continue(c)**: resume program execution. **print(p)** variable ： check the value of a variable • **finish**： forced return (from a function) • **watch variable**： stops whenever the value of the variable changes • **quit(q)**：quit GDB. **How to generate a uniform random variable?:** 1.0\*rand() / RAND\_MAX (requires #include <stdlib.h>) **How to use array? How to pass array to a function?:** int find\_largest(int a[], int n) { … } **How to get user input from command line?** int main (int argc, char \*argv[]) { ... } **Pointer:** void swap(int \*px, int\* py) { int temp; temp = \*px; \*px = \*py; \*py = temp; } **Structure:** typedef struct { char \*face; char \*suit; } Card; Card deck[52]; **Dynamic memory management: Malloc:** line = (char\*)malloc(line length); Takes number of bytes to allocate as argument. • Use sizeof to determine the size of a type. **free(line)**; **Calloc:** void \*calloc(size\_t nitem, size\_t size); requires two arguments - the number of variables you'd like to allocate memory for and the size of each variable. allocated memory space are zero by default. **Realloc**: If you find you did not allocate enough space use realloc(). If you find you did not allocate enough space use realloc(). ip = (int\*)malloc(100 \* sizeof(int)); /\* need twice as much space \*/ ip = (int\*)realloc(ip, 200 \* sizeof(int));

**BS: Which one is NOT one of the four phases in the compilation system?** Compilation, Linking, Preprocessing, Assembly, **Processing.** Which command is used to measure the running time of a C program in Linux. **time ./hello. The primary purpose of an operating system is to make the most efficient use of the computer hardware.** What is phony target in the makefile? **The target which is not a filename.** Using the standard library function can be more efficient because “all of the above”. Which of the following must every C program have? **Main.** Lines beginning with a # are processed at **preprocessor** time. Sensors: Provide simple information , example photoresistor - /light intensity ADC: Analog to Digital converter, converts detected analog signal into a digital form for processing. Microcontroller: An integrated circuit that executes a program. Traditionally slower than a microprocessor (<500 MHz) (sends commands, receives data, need programming). IP Core: Circuit that performs one function. FPGA: Field Programmable gate array, Hardware controlled via RAM, faster than software. DAC: Converts Digital to analog. Acututor; causes events to occurs in the environment (LEDS, LCD, 7-seg display, servo motor).

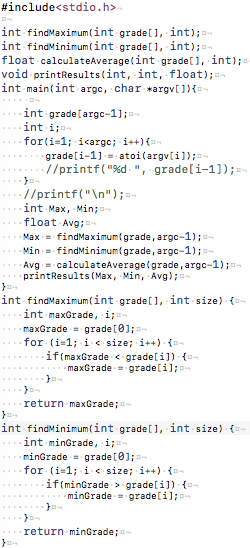
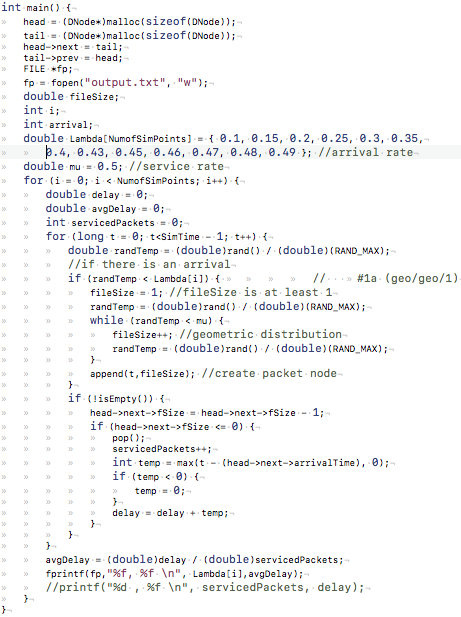
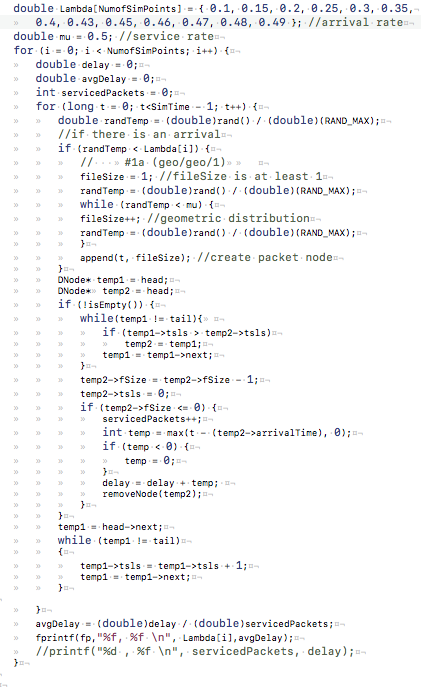
**Properties of TCP:** Connection-oriented: setup required between client and server processes; Reliable transport between sending and receiving process; **Flow control**: sender won’t overwhelm receiver; **Congestion control**: throttle sender when network overloaded; Does not provide: timing, minimum throughput guarantees, security; **Example:** Web, SSH, SMTP (sending mail), IMAP/POP (receiving mail) **Properties of UDP**: Unreliable data transfer between sending and receiving process **Does not provide**: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security; **Example:** Media steaming (lost frame are ok), Games that don’t care if you get every update **Why is Networking Needed?** To enhance many devices, To access data or computational power outside of the device

|  |  |  |
| --- | --- | --- |
| <!DOCTYPE html>  <html>  <head>  <title>Page Title</title>  </head>  <body>  <a href="https://www.w3schools.com/html/">Visit our HTML tutorial</a> <!--comment -->  <img src="pulpitrock.jpg" alt="Mountain View" width="500" height="377">  <h1>This is a Heading</h1>  <p>This is a paragraph.</p>  <table style="width:100%">  <tr>  <th>First Name</th>  </tr>  <tr>  <td>Jill</td>  </tr>  </table>  <frameset rows = "10%,80%,10%">  <frame name = "top" src = "/html/top\_frame.htm" />  <frame name = "main" src = "/html/main\_frame.htm" />  <frame name = "bottom" src = "/html/bottom\_frame.htm" />  <noframes> <body>Your browser does not support frames.</body> </noframes> </frameset>  </body>  </html> | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*mult\_v1.c\*\*\*\*\*\*\*\*\*\*/  #include <stdio.h> #include <stdlib.h> int main(void) {  char \*data;  long m,n; printf("%s%c%c\n","Content-Type:text/html;charset=iso-8859-1",13,10); printf("<TITLE>Multiplication Operation</TITLE>\n");printf("<form><div><label>Multiplicand 1: <input name=\"m\"  size=\"5\"></label></div>\n"); printf("<div><label>Multiplicand 2: <input name=\"n\"  size=\"5\"></label></div>\n");  printf("<div><input type=\"submit\" value=\"Multiply!\"></div></form>");  data = getenv("QUERY\_STRING");  if(data == NULL)  printf("<P>");  else if(sscanf(data,"m=%ld&n=%ld",&m,&n)!=2)  printf("<P>Error! Invalid data. Data must be numeric.");  elseprintf("<P>The product of %ld and %ld is %ld.",m,n,m\*n); return 0;  //TCPClient  import socket  import sys  sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  address = raw\_input("ip address: ")  port = input("port number: ")  sock.connect((address, port))  try:  while True:  message = raw\_input(">> ");  sock.sendall(message)  finally:  sock.close() | b % a = 0  a\*\*b =10 to the power 20  (!=) ==  >=  <= less than  ~ Binary Ones Complement  for num in range(10,20): (10-19)  for num in range(4,10,2): (4,6,8) print '%d equals %d \* %d' % (num,i,j)  len([1, 2, 3]) = 3  [1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]  ['Hi!'] \* 4 = ['Hi!', 'Hi!', 'Hi!', 'Hi!']  3 in [1, 2, 3] = True  for x in [1, 2, 3]: print x = 1 2 3  def printme( str ):  print str  return;  //TCPServer  import socket  import sys  sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  address = socket.gethostbyname(socket.gethostname())  port = input("port number: ")  sock.bind((address,port))  sock.listen(1)  while True:  connection, client\_address = sock.accept()  try:  while True:  data = connection.recv(16)  if data:  print data  Else:  break  finally:  connection.close() |

**OSI Model**: **Application** - Network Process to Application -> **Presentation** - Data representation and Encryption -> **Session** - Interhost communication -> **Transport** - End-to-End connection and Reliability -> **Network** - Path Determination and IP (Logical Addressing) -> **Data Link** - MAC and LLC (Physical addressing) -> **Physical** - Media, Signal and Binary Transmission.

**CGI Flow** - (Browser -> Server) ) 1. Browser requests Document, (Server -> CGI ) 2a. Server invokes CGI, (CGI Program -> Data) 2b. CGI executes and accesses disk or data center as needed, (CGI Program -> Server) 2c. CGI sends “html” to the server (Server -> Browser) 3. Server returns “html” formatted file to browser.

|  |  |
| --- | --- |
| //UDPServer  import socket  import sys  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  port = input("host port: ")  sock.bind(('',port))  while True:  data, addr = sock.recvfrom(1024)  if data == 'stop':  break  else:  print "From addr: '%s', msg: '%s'" % (addr[0], data)  sock.close() | //UDPClient  import socket  import sys  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  address = raw\_input("host ip address: ")  port = input("host port: ")  while True:  data = raw\_input('>> ')  if len(data) == 0:  break  else:  if sock.sendto(data, (address, port)):  print "sending message"  sock.close() |



Round Robin FCFS