Unix
Bash
C
GNU
Systems

Software Systems

Lectures Week 11

Systems Programming 2

(void *, sockets, system calls)

Prof. Joseph Vybihal

Computer Science

McGill University

Week 11 Lecture 1

void *

Readings: http://www.geeksforgeeks.org/void-pointer-c/

About void *

An important systems feature in C is the ability to make a pointer that can point to anything.

There exist things in a computer that we would like to manipulate that do not fall under the standard types (int, char, float, double, pointer, struct, object).

Examples

- Peripherals
  - Computers are connected to multiple secondary devices, like, keyboards, mice, printers, clickers, cell phone, joystick, trackpads, VR devices, robotic sensors, etc.
  - Straightforward ways to interface with these devices are needed
- Dynamic algorithms
  - Sophisticated applications require interchangeable software components. Plug-and-play software that goes beyond function calls, recursion and OO programming.

void* Pointers

- Usage

```
int x=5,y;
void *p; // void * has no type

p=&x;
y = *p;              // warning message
y = *((int *)p); // type casting, no warning

printf("%d\n",y);
```

void* Pointers

- Can point to many types, even structs

```
int x=5,y;
char c='A',d;
void *p;

p=&x;                    // p is referencing int
y = *((int *)p);
p=&c;                    // p now references char
d = * ((char *)p);
```

Peripherals and void *

A peripheral device connects with a computer either directly with the CPU or indirectly through RAM.

The void* is used with peripherals that connect through RAM. Examples include:

- Keyboard, mouse, joystick, printer
- Graphics card
- Any device that connects to the system board slots

Peripherals and void *

RAM based peripheral connections are based on “registers”.

A register is a section of memory that contains information about the device.

Common registers:

- Status: is it on, is it ready, error codes
- Data: data to read or write
- Command: telling the device what to do

Peripherals and void *

Each register has an address and a size in bytes. It may or may not have a type.

Assume we know a printer's status register is at address 52 in RAM and it 1 byte long.

Assume this status register looks like this:

Bit 0 = 1 for on, 0 for off

Bit 1 = 1 for ready to print, 0 not ready to print

Bits 2-7 = error code

Peripherals and void *

Assume this status register looks like this:

Bit 0 = 1 for on, 0 for off

Bit 1 = 1 for ready to print, 0 not ready to print

Bits 2-7 = error code

How can we find out if printer is on?

```
void *p = 52; // from last slide, the address of the status register
int x;

x = *((short *) p) & 1; // 1 = 00000001

if (x == 0) // then the printer is off
```

Important

Operating systems run in two modes:

- Privileged
  - Programs can access registers
- Standard
  - Programs have limited or no access to registers

By default modern OS locks users in Standard mode.

Pointers to functions

"Pointers to functions" is not that same as "using pointers with functions"

Pointers with functions

- void aFunction(int x, int *y) { …. }
  - In this example aFunction receives an integer and a pointer to an integer as parameters, and does not return anything.
- int *aFunction2(int x, char c) { …. }
  - In this example aFunction2 receives an interger and a character as parameters and returns a pointer to an integer.

Pointers with Function example

Swap:

```
void swap(int *p, int *q) {
   int temp = *p;
   *p = *q;
   *q = temp;
}

int main() {
   int x = 5, y = 10;
   swap(&x, &y);
   printf("%d %d\n", x, y);
}
```

Create:

```
struct STUD* create() {
   struct STUD *p;
   p=malloc(sizeof(struct STUD));
   return p;
}

int main() {
   struct STUD *newStudent;
   newStudent = create();
   printf("%d\n", newStudent->age);
}
```

Pointers to functions

Declaration comparison:

(1) `int *fn()` pointer with function example

(2) `int (*fn)()` pointer to a function that returns an integer

Invocation:

`if ((*fn)(a,b) <= 0)` needs the extra brackets to interpret correctly

```
int x = (*fn)(a,b);
```

Binary Search

(stdlib.h)

Declaration:

```
void *bsearch(const void *key, const void *base, size_t nitems, size_t size,
              int (*compar)(const void *, const void *))
```

Parameters:

key	This is the pointer to the object that serves as key for the search, type-casted as a void*.
base	This is the pointer to the first object of the array where the search is performed, type-casted as a void*.
nitems	This is the number of elements in the array pointed by base.
size	This is the size in bytes of each element in the array.
compar	This is the function that compares two elements.

Return Value:

This function returns a pointer to an entry in the array that matches the search key.
If key is not found, a NULL pointer is returned.

Binary Search

```
#include <stdio.h>
#include <stdlib.h>

int values[] = { 5, 20, 29, 32, 63 };

int cmpfunc(const void * a, const void * b)
{
   return ( *(int*)a - *(int*)b );
}
int main ()
{
   int *item;
   int key = 32;

   /* using bsearch() to find value 32 in the array */
   item = (int*) bsearch (&key, values, 5, sizeof (int), cmpfunc);
   if( item != NULL )
   {
      printf("Found item = %d\n", *item);
   }
   else
   {
      printf("Item = %d could not be found\n", *item);
   }

   return(0);
}
```

Other built in void* functions

- void qsort(void *base, size_t nitems, size_t size,
  int (*compar)(const void *, const void*))
  - Performs quick sort (stdlib.h)
- int atexit(void (*func)(void))
  - Calls a function at normal termination (stdlib.h)

Example qsort

```
#include <stdio.h>
#include <stdlib.h>

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b) {
  return ( *(int*)a - *(int*)b );
}

int main() {
  int n;

  printf("Before sorting the list is: \n");
  for( n = 0 ; n < 5; n++ ) {
    printf("%d ", values[n]);
  }

  qsort(values, 5, sizeof(int), cmpfunc);

  printf("\nAfter sorting the list is: \n");
  for( n = 0 ; n < 5; n++ ) {
    printf("%d ", values[n]);
  }

  return(0);
}
```

Week 11 Lecture 2

Socket Communication

Readings: http://www.binarytides.com/socket-programming-c-linux-tutorial/

What is a socket?

A socket connects two computers over a network.

A socket is composed of:

- A data structure
- A network
- A communication algorithm
- A socket is uni-directional

Example:

- Typing www.google.com at your browser causes the browser to create a socket between you and Google so it can send and receive data.

Network addresses

Every computer on a network is assigned a unique address.

The address is used to identify the computer. Information is sent to a specific computer using its unique address.

The Internet has an address format called IPv4 and IPv6 (small addresses vs large addresses)

- Example IP: 123.222.333.000

Network port

Every process running on your computer has a PID (a unique process ID). The OS uses this PID to identify and communicate with the process.

The network uses unique port ID numbers to connect a PID with an IP address.

Example:

- Computer 1: IP 123.456.789.01, Port 50, PID 17 listening
- Computer 2: IP 111.222.333.000, Port 50, PID 35 sending
- This way computer 2 can send data to computer 1

The socket data structure

```
// IPv4 AF_INET sockets:
struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET, AF_INET6
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};

struct in_addr {
    unsigned long s_addr;          // load with inet_pton()
};

struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

Connect to a server

```
#include<stdio.h>
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[]) {
    int socketID;
    struct sockaddr_in server;

    //Create socket
    socketID = socket(AF_INET , SOCK_STREAM , 0);
    if (socketID == -1) {
        printf("Could not create socket");    exit(1);
    }

    server.sin_addr.s_addr = inet_addr("74.125.235.20");
    server.sin_family = AF_INET;
    server.sin_port = htons( 80 );

    //Connect to remote server
    if (connect(socketID , (struct sockaddr *)&server , sizeof(server)) < 0) {
        puts("connect error"); exit(2);
    }

    puts("Connected");  // now we can send data after this
    return 0;
}
```

Sending data to server

```
char *message;

//Send some data
message = "GET / HTTP/1.1\r\n\r\n";

if( send(socketID , message , strlen(message) , 0) < 0)
{
    puts("Send failed");
    return 1;
}

puts("Data Sent\n");  // if you see this then the string was sent to the server

// After this we wait for a reply from the server
```

Reply from server

```
char server_reply[2000];

//Receive a reply from the server
if( recv(socket_desc, server_reply , 2000 , 0) < 0)
{
    puts("recv failed");
}

puts("Reply received\n");
puts(server_reply);
```

Sample Output

```
Connected

Data Sent

Reply received

HTTP/1.1 302 Found
Location: http://www.google.com/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Set-Cookie:
PREF=ID=0edd21a16f0db219:FF=0:TM=1324644706:LM=1324644706:S=z6hDC9cZfGEowv_o;
expires=Sun, 22-Dec-2013 12:51:46 GMT; path=/; domain=.google.com
Date: Fri, 23 Dec 2011 12:51:46 GMT
Server: gws
Content-Length: 221
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
```

Packet header

Packet Data

A simple server

```
int main(int argc, char *argv[]) {
   int listenfd = 0, connfd = 0;
   struct sockaddr_in serv_addr;

   char sendBuff[1025];
   time_t ticks;

   listenfd = socket(AF_INET, SOCK_STREAM, 0); // AF_INET=IPv4, SOCKET_STREAM=TCP, 0=auto protocol
   memset(&serv_addr, '0', sizeof(serv_addr));
   memset(sendBuff, '0', sizeof(sendBuff));

   serv_addr.sin_family = AF_INET;                          // Accept IPv4 addresses
   serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);           // Accept any address
   serv_addr.sin_port = htons(5000);                        // communication through port 5000

   bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)); // assign values to socket

   listen(listenfd, 10); // permit maximum of 10 users on this socket

   while(1) {                                               // servers never stop working…
      connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);  // server sleeps until a connection made
                                                            // connfd= is client socket ID
      ticks = time(NULL); // this server program simply return the time to the client socket (get time)
      snprintf(sendBuff, sizeof(sendBuff), "%.24s\r\n", ctime(&ticks)); // (format time as string)
      write(connfd, sendBuff, strlen(sendBuff));                // (send string to client)

      close(connfd);      // close connection with client… work is done.
      sleep(1);           // give a chance for other programs on server to run…
    }
}
```

Week 11 Lecture 3

Other Systems Calls

Common System Calls

- int remove(const char *filename)
  - Delete file from path (stdio.h)
- int rename(const char *old_file, const char *new_file)
  - Rename file from path (stdio.h)
- void rewind(FILE *stream)
  - Moves pointer back to first character of file/stream(stdio.h)
- int ungetc(int char, FILE *stream)
  - Put the last read character back into the stream (stdio.h)
- FILE * popen (const char *command, const char *mode)
- int pclose (FILE *stream)
  - Like system but creates a read/write ASCII pipe (stdio.h)

Example: popen

```
#include <stdio.h>
#include <stdlib.h>

void  write_data (FILE * stream) {
 int i;
 for (i = 0; i < 100; i++) fprintf (stream, "%d\n", i);
 if (ferror (stream)) {
    fprintf (stderr, "Output to stream failed.\n");
    exit (EXIT_FAILURE);
  }
}

int main (void) {
 FILE *output;

 output = popen ("more", "w");
 if (output == NULL) {
    fprintf (stderr, "incorrect parameters or too many files.\n");
    return EXIT_FAILURE;
  }
 write_data (output);
 if (pclose (output) != 0) fprintf (stderr, "Could not run more or other error.\n");

 return EXIT_SUCCESS;
}
```

Example: ungetc

```
#include <stdio.h>

int main () {
  FILE *fp;
  int c;
  char buffer [256];

  fp = fopen("file.txt", "r");
  if( fp == NULL )  {
    perror("Error in opening file");
    return(-1);
  }
  while(!feof(fp)) {
    c = getc (fp);

    if( c == '!' )  // assumes 1st char could be an !, replace ! with +
      ungetc ('+', fp);
    else
      ungetc(c, fp);

    fgets(buffer, 255, fp);
    fputs(buffer, stdout);
  }
  return(0);
}
```

Example: rewind

```
#include <stdio.h>
int main() {
   char str[] = "This is tutorialspoint.com";
   FILE *fp;
   int ch;

   fp = fopen( "file.txt" , "w" );  // write some content in the file
   fwrite(str , 1 , sizeof(str) , fp );
   fclose(fp);

   fp = fopen( "file.txt" , "r" );
   while(1) {
      ch = fgetc(fp);
      if( feof(fp) ) break ;
      printf("%c", ch);
   }
   rewind(fp);
   printf("\n");
   while(1) {
      ch = fgetc(fp);
      if( feof(fp) ) break ;
      printf("%c", ch);
   }
   fclose(fp);
   return(0);
}
```

Example: remove and rename

```
#include <stdio.h>
int main() {
  char name[100], name2[100];
  int result;

  printf("File Name: ");
  scanf("%s", name);

  if (strcmp(name,"abc.txt")==0)
            result = remove(name);
            if (result != 0) exit(1);
  else {
            printf("Destination: ");
            scanf("%s", name2);
            result = rename(name, name2);
            if (result != 0) exit(2);
  }
  return(0);
}
```