Workshop: Connecting to hardware

Goal of the excercises

The goal of this excercise is to try and abstract the different tasks to perform and separate them into different modules in the program. The excercises both simulate progress in the hardware and how it should be addressed, and in the demands that are made by the scientist using the application.

General instructions

Try to solve these problems in such a way that a small change in the "hardware" to be controlled or a small change in the kind of “experiment" you want to do with the hardware will only need small changes in your code.

Description of the environment

The "hardware" is represented in this excercise as a server that is running on port 7777 of a computer attached to the network. You can connect to it using a TCP socket connection. You send it an ASCII string command, followed by a carriage-return (ASCII 13). The server will give you an answer structured like:

- 1 character protocol-id ("1")
- 5 characters representing an integer status:
  - status = 0 = ERROR
  - status = 1 = OK
- 5 characters representing a block length (n)
- a carriage-return character
- n characters of data or n characters of error message

After the data block, the connection is closed by the server.

The server has an unknown number (call it x) of strings that it can return; these are numbered from 0 to x-1, and can be retrieved by sending a string representation of the number ("%d", followed by ASCII 13 as was said earlier) as a command to the server. Some other, secret, commands are available in the server as well, you will find out about these during the excercises.

Socket programming

To complete this excercise the program must make an internet socket. The use of such a socket consists of a few steps:

- Create a socket. Parameters are:
    - "domain" is PF_INET or AF_INET
    - "type" is SOCK_STREAM
    - "protocol" (if needed in your language) is 0
- Create a socket address from the host name or IP address and the port number
- Connect the socket to the address
- Use send() and recv() to communicate over the socket. IO

In case you have no experience programming with sockets, an example of how the first exercise can be solved is given on subsequent pages of this document. Examples are available in python and in C, and should not be used as an example of how to solve the exercise well, but only on how to use socket connections.

The excercises

Excercise 1: Make a client that connects to the server, and retrieves
and prints out string 0 (zero). That string is the next excercise.

Example program in C

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <stdlib.h>

int port = 7777;

int s;
struct sockaddr_in serv_addr;
struct hostent *servent;

/* Receive a message from a socket based on knowledge of
the size of
   the packet to be received */
int recvfixed(int s, int n, char *x) {
  int i = 0;
  int tmp;
  while (i<n) {
    tmp = recv(s, x, n-i, MSG_WAITALL);
    if (tmp == -1) { return (-1); }
    i += tmp;
  }
  s[n] = '\0';
  return (0);
}

/* Open our socket and connect it to the server */
int sopen() {
  int s;
  s = socket(PF_INET, SOCK_STREAM, 0);
  servent = gethostbyname("127.0.0.1");
  memset(&serv_addr, 0, sizeof(serv_addr));
  serv_addr.sin_family = servent->h_addrtype;
  memcpy((char *) &serv_addr.sin_addr.s_addr, servent-
>h_addr_list[0],
           servent->h_length);
  serv_addr.sin_port = htons(port);
  connect(s, (struct sockaddr *)&serv_addr, sizeof
(serv_addr));
  return (s);
}

int main() {
  /* All the strings we need */
```

```
  char protocol[2];
  char s_status[6];
  char s_length[6];
  char s_sep[2];
  char *mess;
  /* Two of the strings correspond to integers */
  int status, length;
  /* Open the socket */
  s = sopen();
  /* Send the command 0 */
  send(s, "0\r", 2, 0);
  /* Interpret the protocolled answer */
  if (recvfixed(s, 1, protocol)) {
    fprintf(stderr, "Could not receive protocol\n");
    return (1);
  }
  if (strncmp(protocol, "1", 1)) {
    fprintf(stderr, "Protocol mismatch\n");
    return (1);
  }
  if (recvfixed(s, 5, s_status)) {
    fprintf(stderr, "Could not receive status\n");
    return (1);
  }
  status = strtol(s_status, NULL, 10);
  if (recvfixed(s, 5, s_length)) {
    fprintf(stderr, "Could not receive length\n");
    return (1);
  }
  length = strtol(s_length, NULL, 10);
  if (recvfixed(s, 1, s_sep)) {
    fprintf(stderr, "Could not receive separator\n");
    return (1);
  }
  mess = malloc(length+1);
  if (recvfixed(s, length, mess)) {
    fprintf(stderr, "Could not receive message\n");
    return (1);
  }
  if (status == 0) {
    fprintf(stderr, "ERROR: %s\n", mess);
    return (2);
  } else {
    fprintf(stdout, "%s\n", mess);
  }
  return (0);
}
```

Example program in Python

```
import sys,socket

host = 'localhost'
port = 7777

class Error(Exception):
    pass

def recvfixed(s, n):
    ret = ''
    while n > 0:
        str = s.recv(n)
        ret += str
        n -= len(str)
    return ret

def sopen():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    return s

def main():
    s = sopen()
    s.send("0\r")
    try:
        if recvfixed(s, 1) != '1':
            print >>sys.stderr, "Protocol mismatch"
            sys.exit(1)
    except IOError:
        print >>sys.stderr, "Could not receive protocol"
        sys.exit(1)
    try:
        status = int(recvfixed(s, 5))
    except IOError:
        print >>sys.stderr, "Could not receive status"
        sys.exit(1)
    try:
        length = int(recvfixed(s, 5))
    except IOError:
        print >>sys.stderr, "Could not receive length"
        sys.exit(1)
    try:
        sep = recvfixed(s, 1)
    except IOError:
        print >>sys.stderr, "Could not receive separator"
        sys.exit(1)
    try:
```

```
        mess = recvfixed(s, length)
    except IOError:
        print >>sys.stderr, "Could not receive message"
        sys.exit(1)
    if status == 0:
        print >>sys.stderr, "ERROR: %s" % mess
        sys.exit(2)
    else:
        print mess

if __name__ == "__main__":
    main()
```