CS 501 HMWK 5

Alex Hagen

11/15/13

Problem 1 - Symbolic Function Wrapping and Function Pointer Passing In order to wrap a C function in another object, the `dlfcn.h` library must be used. It will open the library with `dlopen`, and then a pointer to the functions inside the library can be created with `dlsym`. Then, to pass these functions from the instantiator to other functions within the object, a member can be created corresponding to the pointer to that function, which can then be dereferenced at any point inside the functions of the program. Figure 1 shows the output of the tester script, while Source Code 1 shows the source code of the file `wrapDL.c`.

```
$ python p1_testscript.py
***   Test 1   ***
The desired output is:
Average gain = -0.02
Average # tosses = 132.36
Average gain = -0.02
Average # tosses = 264.22

Your output is:
Average gain = -0.02
Average # tosses = 132.36
Average gain = -0.02
Average # tosses = 264.22
```

Figure 1: Sample Output using Tester `p1_testscript.py`

Source Code 1 C Function Pointer Wrapping and Passing using `dlsym`

```
1  // wrapDL.c
2  #include <stdlib.h>// standard functions
3  #include <stdio.h> // io functions (printf)
4  #include <dlfcn.h> // dynamic linking functions
5
6  typedef struct Coin{
7          void *dl;               //the pointer to the dynamic library
8          void *cointoss;} Coin; //the pointer to the passed function
9
10 Coin *Coin_new(int seed) {  //constructor
11         Coin *coin=(Coin *)malloc(sizeof(Coin));  //allocate the structure's mem
12         srand(seed);                                //seed the rand num generator
13         coin->dl = dlopen("./_playDL.so", RTLD_LAZY);  //open the dynamic library
14         if (coin->dl == NULL) {printf("%s\n", dlerror()); exit(1);}
15           //exit and print error if one occurs
16         typedef int (*coint_t)(int myfunds, int yourfunds, int *tosses);
17           //create a type for the fcn
18         coint_t cointoss = (coint_t)dlsym(coin->dl,"cointoss"); //link the fcn
19         coin->cointoss = cointoss; //make the pointer of that fcn an object var
20         return coin;} //return the structure
21
22 int Coin_toss(Coin *coin, int myfunds, int yourfunds, int *tosses){
23         int total; //make an integer to store the total
24         typedef int (*coint_t)(int myfunds, int yourfunds, int *tosses);
25           //create a type for the fcn (reduntant from instantiator)
26         coint_t cointoss; //create a function with that type
27         cointoss=coin->cointoss; //pass the fcn pointer to the function
28         total = cointoss(myfunds,yourfunds,tosses); //use the function
29         return total;} //return the total loss/gain
30
31 void Coin_delete(Coin *coin) {
32         dlclose(coin->dl); //close the dynamic library
33         free(coin);} //free the structure variable memory
```

Problem 2 - Wrapping and Function Passing from C to Python Wrapping in python is made slightly harder by the requirement of converting things to C language types, but this can be done by the ctype library. Luckily, only integers and pointers to integers must be used in driving this C function, so the wrapper is very simple, and returns the output shown in Figure 2 when run with the source code shown in Source Code 2.

```
$ python p2_testscript.py
***   Test 2   ***
The desired output is:
Average gain = -0.02
Average # tosses = 132.36
Average gain = -0.02
Average # tosses = 264.22

Your output is:
Average gain = -0.02
Average # tosses = 132.36
Average gain = -0.02
Average # tosses = 264.22
```

Figure 2: Sample Output using Tester `p2_testscript.py`

Source Code 2 C Function Pointer Wrapping and Passing to Python using `ctypes`

```
#!/usr/bin/env python # wrapDL.py

from ctypes import CDLL, c_int, POINTER
from ctypes.util import find_library

class Coin(object):
        def __init__(self, seed):
                #seed the random number generator
                cseed = c_int(); cseed.value = seed
                CDLL(find_library("c")).srand(cseed)
                #open the dynamic library
                dynlib = CDLL('./playDL.so')
                #load the cointoss function to a pointer
                dynlib.cointoss.restype = c_int
                self.coin_toss = dynlib.cointoss

        def toss(self, myfunds, yourfunds):
                #Make Ctype variables myfunds and yourfunds, and pointer tossesp
                cmyfunds = c_int(); cmyfunds.value = myfunds
                cyourfunds = c_int(); cyourfunds.value = yourfunds
                ctosses = c_int(); ctossesp = POINTER(c_int)(ctosses)
                #get the result from the c function
                total=self.coin_toss(cmyfunds,cyourfunds,ctossesp)
                #extract the value from the pointer
                tosses = ctosses.value
                #return desired values
                return total,tosses

if __name__ == '__main__':
        print 'This class should be called with driveD2.py, please!'
```

Problem 3 - Standalone playPipe.c and Wrapper

Standalone playPipe.c from stdin to stdout The standalone program is very simple, replacing the function call with a main program, and instead of input values and return values, using the functions `scanf` and `printf` to import and export from stdin and stdout. The Source Code given in Source Code 3 will be utilized further in Subparagraph .

Source Code 3 Standalone C function `playPipe.c` with inputs from stdin and return to stdout

```c
// playPipe.c
#include <stdlib.h>
#include <stdio.h>
int main(void){
        //initialize all the c variables
        int myfunds; int yourfunds; int tossesp;
        //start at zero tosses
        tossesp=0;
        //read stdin into two different integer variables
        scanf("%d\n%d",&myfunds,&yourfunds);
        //exact process from playDL.c
        int myfunds0=myfunds;
        while (myfunds > 0 && yourfunds > 0){
      (tossesp)++;
      if (rand() % 2 == 0){
         myfunds++; yourfunds--;}
      else {
         myfunds--; yourfunds++;}}
    //print the outputs (total and tosses) to stdout
        printf("%d %d\n",myfunds - myfunds0,tossesp);
}
```

Python Wrapper for standalone playPipe.c Utilizing the above function with unnamed pipes is simple using the subprocess library from python. A process is opened, with PIPEs specified for inputs and outputs. Then, simply formatting strings into the correct format, these can be passed to the process as if they were stdin and stdout. The output in Figure 3 is obtained when using the tester script, showing the efficacy of this process when run with Source Code 4.

```
$ python p3_testscript.py
***   Test 3   ***
The desired output is: -7 139
Your output is: -7 139
***   Test 4   ***
The desired output is:
Average gain = 7.00
Average # tosses = 115.00
Average gain = 14.00
Average # tosses = 230.00

Your output is:
Average gain = 7.00
Average # tosses = 115.00
Average gain = 14.00
Average # tosses = 230.00
```

Figure 3: Sample Output using Tester Script `p3_testscript.py`

Source Code 4 Source Code Wrapper `wrapPipe.py` for Running `playPipe.c` through a Python Interface

```
#!/usr/bin/env python # wrapPipe.py

from ctypes import CDLL, c_int
from ctypes.util import find_library
from subprocess import Popen, PIPE
from string import split

class Coin(object):
        def __init__(self, seed):
                #seed the random number generator
                cseed = c_int(); cseed.value = seed
                CDLL(find_library("c")).srand(cseed)

        def toss(self, myfunds, yourfunds):
                #Open a new process that calls the playPipe program
                # and then uses pipes for stdin, stdout, and stderr
                proc = Popen('./playPipe', shell=True, \
                        stdin=PIPE,stdout=PIPE,stderr=PIPE)
                #concatenate the myfunds and yourfunds into formatted input
                idata = str(myfunds)+"\n"+str(yourfunds)+"\n"
                #get the output data and errdata from the process opened
                odata, edata = proc.communicate(input=idata)
                #split those lines to separate the funds and tosses
                lines=odata.split(" ");
                #reconvert the strings back to integers
                return int(lines[0]),int(lines[1])

if __name__ == '__main__':
        print 'This class should be called with p3_testscript.py, please!'
```

Problem 4 - Standalone playBin.c and Wrapper

Standalone playBin.c from Named Pipes The standalone program is very simple, replacing the function call with a main program, and instead of input values and return values, using the functions `fread` and `fwrite` to read and write from binary files. The Source Code given in Source Code 5 will be utilized further in Subparagraph .

Source Code 5 Standalone C function `playBin.c` with inputs from binary input file and return to named pipe

```c
// playBin.c
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
        //initialize all the c variables
        int myfunds; int yourfunds; int tossesp;
        //start at zero tosses
        tossesp=0;
        //open input file for reading
        char *ifile=argv[1];
        FILE * fi;
        fi=fopen(ifile,"rb");
        //read two integers from file
        int buffer[2];
        fread(buffer,sizeof(int),2,fi);
        //set these ints to the correct variables
        myfunds=buffer[0]; yourfunds=buffer[1];
        fclose(fi);
        //exact process from playDL.c
        int myfunds0=myfunds;
        while (myfunds > 0 && yourfunds > 0){
      (tossesp)++;
      if (rand() % 2 == 0){
         myfunds++; yourfunds--;}
      else {
         myfunds--; yourfunds++;}}
    //open outputfile for writing
    char *ofile=argv[2];
    FILE * fo;
        fo=fopen(ofile,"wb");
        //put the ints into a buffer array
        buffer[0]=myfunds-myfunds0; buffer[1]=tossesp;
    //print the outputs (total and tosses) to stdout
        fwrite(buffer,sizeof(int),2,fo);
        fclose(fo);}
```

Python Wrapper for standalone playPipe.c Utilizing the above function with unnamed pipes is simple using the subprocess library from python. A process is openedand the input and output filenames appended to the function call. Then, writing numpy arrays to binary files can produced the required input and read from the binary output of the program. The output in Figure 4 is obtained when using the tester script, showing the efficacy of this process when run with Source Code 6.

```
$ python ./p4_testscript.py
***   Test 5   ***
The desired output is:
[ -7 139]
Your output is:
[ -7 139]
***   Test 6   ***
The desired output is:
Average gain = 7.00
Average # tosses = 115.00
Average gain = 14.00
Average # tosses = 230.00

Your output is:
Average gain = 7.00
Average # tosses = 115.00
Average gain = 14.00
Average # tosses = 230.00
```

Figure 4: Sample Output using Tester Script `p4_testscript.py`

Source Code 6 Source Code Wrapper `wrapBin.py` for Running `playBin.c` through a Python Interface

```
#!/usr/bin/env python # wrapBin.py

from os import remove
from ctypes import CDLL, c_int
from ctypes.util import find_library
from subprocess import Popen, PIPE
from numpy import getbuffer,frombuffer,array

class Coin(object):
        def __init__(self, seed):
                #seed the random number generator
                cseed = c_int(); cseed.value = seed
                CDLL(find_library("c")).srand(cseed)

        def toss(self, myfunds, yourfunds):
                iname="tempp4input.dat"; oname="tempp4output.dat";
                #Open a new process that calls the playBin program with the
                #input file iname and output file iname
                proc = Popen('./playBin '+iname+' '+oname, shell=True)
                #concatenate the myfunds and yourfunds into the ifile
                ifile = open(iname,'wb');
                idata = getbuffer(array([myfunds,yourfunds],dtype="int32"))
                ifile.write(idata)
                ifile.close()
                proc.communicate()
                #get the output data from the output file
                ofile = open(oname,'rb')
                buf = ofile.read()
                [total,tosses]=frombuffer(buf,dtype="int32",count=2)
                #close and remove any leftover files
                ofile.close()
                remove(iname)
                remove(oname)
                #reconvert the strings back to integers
                return int(total),int(tosses)

if __name__ == '__main__':
        print 'This class should be called with p4_testscript.py, please!'
```