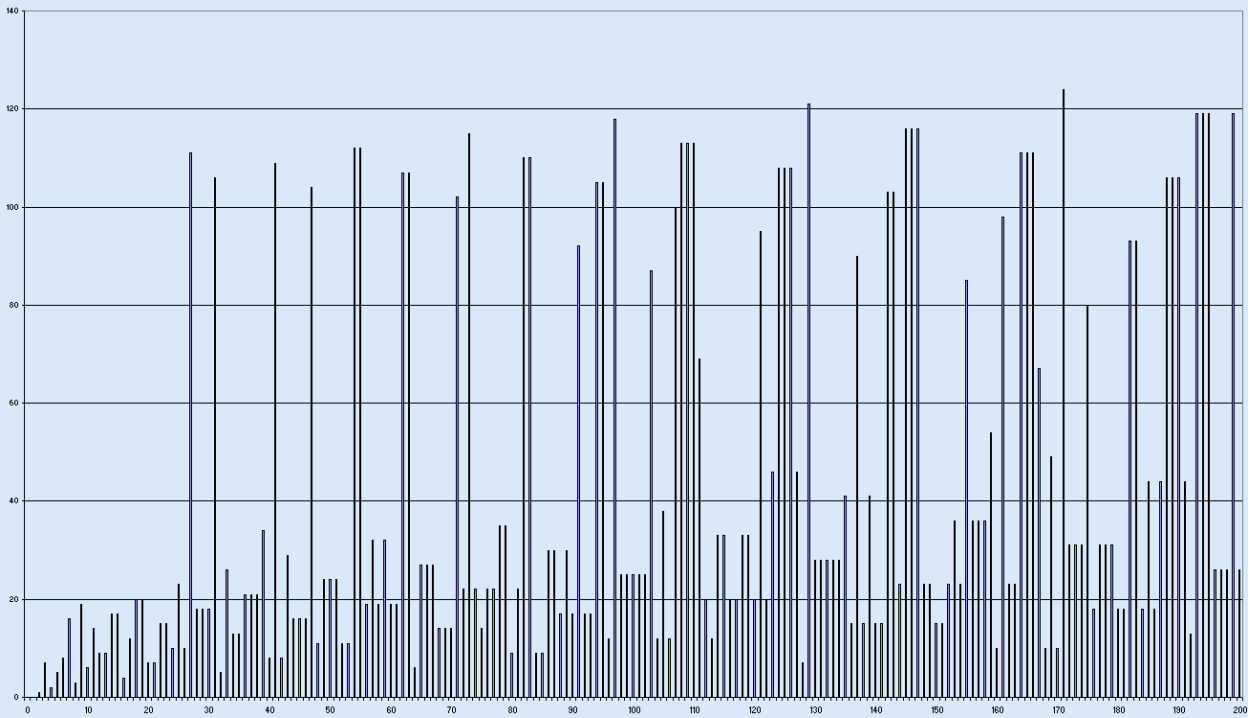
CANOS Project 4

Weird Mathematics and the Operating System

(complete this project in groups of up to 3)



A graph of the number of steps required to get to the number 1 when applying the Collatz function producure to numbers 1-200. (Source: Wikimedia Commons)

In 1937, German mathematician Lothar Collatz proposed a deceptively simple question involving a function f(n) operating on the positive integers. The function operates in the following way:

```
f(n) = n/2 if (n mod 2 = 0)
f(n) = 3n + 1 if (n mod 2 = 1)
```

In other words, if n is odd, f(n) triples it and then adds 1, but if f(n) is even, f(n) divides it by 2. If you plug an integer into f(n), then take the output of f(n) and apply f(n) to it again,

repeatedly iterating the function in this manner, you will often generate sequences that enter patterns of increasing and decreasing value before settling at 1.

Collatz's proposal, which is now called the Collatz conjecture, was that any arbitrary integer iterated upon by f(n) in this way would eventually converge to 1. On the surface, the Collatz conjecture sounds as though it might be provable by your typical skilled mathematician. However, it has proven to be one of modern mathematics' most elusive unsolved problems, with several of the best mathematicians of the 20th and 21st century commenting on how difficult the problem is to solve and how inadequate our current mathematical techniques are for solving it.

Despite this, it is relatively easy to repeatedly apply the f(n) function described above to an integer and see what pattern of numbers is created. In this project, you will write a program to do exactly this, incorporating what you've learned in the third half of the CANOS course about the x86 assembly language and operating system functions.

Questions:

1.) (2 pts) If you are running Windows, download Cygwin and install it on your machine. Cygwin is a program that creates a Unix-style (also Linux-style) terminal environment on a non-Unix machine such as Windows, allowing the use of the Unix-style system calls discussed in our lectures. This is the environment in which we will compile and run our program. (If you're well-versed in Unix terminal commands, note that the Cygwin terminal will understand them.)

While going through the Cygwin installer, you will see dialog "Cygwin Setup - Select Package". Under the "View" you should select "Full", and in the "Search" edit you should enter "gcc". Further in the list below in "Package" column find "gcc-core" row and change the box in the "New" column from "Skip" value to the latest version of gcc, Then, continue the install process.

Once Cygwin is installed, boot it up and type this command:

```
cygpath -w ~
```

This will display Cygwin's home directory. You can use your normal file browser to put files into this folder and access them within Cygwin.

For Question 1 in the report, include a screenshot of the output of cygpath being displayed in a Cygwin terminal window. (This only needs to be done for one

member of the team.) If you are already running a Unix-based operating system, simply state which operating system you are running in Question 1.

2.) (3 pts) Create a Hello World program in the C language. Put it in your Cygwin home folder (or any folder if you are running Unix), then compile it with gcc and run the resulting executable within Cygwin to display “Hello World!” or some other simple message at the terminal.

For Question 2 in the report, include a screenshot showing a successful gcc compile and a successful execution of your program. For help, refer to the guide “Compiling a C program using GCC” under Tips and Further Reading.

3.) (20 pts) You will now write a program in C that performs a “race” to the number 1 between three integers that are operated upon by the Collatz function described in the introduction. It should have the following properties:

a. It should accept three integer inputs from the user at the terminal. If the user inputs a number that is not a positive integer, the program should report that it is in “random number mode” and generate three random positive integers in place of the three user-generated inputs.
b. After accepting the three integers, it should fork twice to create a total of three processes. (Process forking is covered in “Inter-Process Communication” lecture.) Each of the three processes should iterate the Collatz function described in the introduction on one of the three integers. Each time one of the processes iterates, it should output its process PID to the terminal as well as the current value of the number being iterated upon. When the process reaches a value of 1, it should report that it has reached 1, display a terminal message that this process is complete, then the process should terminate.
c. Each time a process completes one iteration of the function, it should either wait for the user to press a key or wait for a short time interval. You should accomplish this using a Unix system call. Further documentation on the Unix system calls is at the end of this document.
d. When a process completes, it should tell the other two processes that it is complete using a pipe. Each process should have an internal counter of how many processes are running (starting with 3) that decrements when it receives a notice of a process completion. The first process to complete (i.e. the process that completes with a counter of 3) should print “I WIN!” to the terminal upon completion. The last process to complete (i.e. the process that completes with a counter of 1) should print “I LOSE!” to the terminal upon completion.

The code can be written with a regular text editor or the IDE program of your choice and then dropped into your Cygwin home directory to be compiled and run within Cygwin. When you have finished writing your code, submit it to the Project 4 Dropbox on Gradescope.

4.) (5 pts) Compile your program to human-readable assembly code using gcc -S as described in the "Compiling a C Program using GCC" link. Include this assembly code in your report. Comment on features of your C program that you can see in the assembly code, and comment on the way the program has changed during the assembly process. Can you see your in-line assembly instructions? Have they changed at all?

5.) (5 pts) Describe some of the results of the "number race" that your program performed. Name 3 numbers that you input into the program and describe which number reached 1 and terminated its process first. Were any of the results of the race surprising?

6.) (5 pts) Describe what you have learned about iUnix system calls. Did anything about the syntax or usage confuse you?

7.) (5 pts) If you worked as a team, briefly describe how the work was divided amongst your teammates.

8.) (5 pts) Explain your design process in a paragraph or two. How did you set about designing this program? Did you try anything that ultimately did not work and had to be revised?

Tips and Further Reading

- The Collatz Conjecture (background reading)
- getrandom() - for random number generation
- The gcc compiler within Cygwin should notify you of any C libraries that should be referenced in your program. However, you are likely to need <stdio.h>, <unistd.h> (for system calls), and <stdlib.h>.
- Compiling a C Program Using GCC
- Unix System Calls
- Demonstration of fork() and pipe()