Programming Assignment I

CS3331

Spring 2025

1 Running Three Independent Processes

This programming assignment will give you experience with process creation using the `fork()`, `wait()` and `exit()` system calls. To do this, we will create a program that (with maximum concurrency) spawns three child processes that each perform different calculations. The program will wait for all of these children to finish their calculation, and then gracefully exit.

2 Program Structure & Specification

Your program will spawn three separate child processes that each perform their own computation. These three processes must run concurrently!

2.1 n-th Fibonacci Number

The n-th Fibonacci number is defined recursively as:

$$\begin{aligned} f_1 = f_2 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \quad \text{if } n > 2 \end{aligned}$$

In this child process, you will use the above formula to recursively compute (and print) the n-th Fibonacci number. This method of calculating the Fibonacci sequence is (asymptotically) slow, $O(2^n)$, in fact! You are probably aware of much more efficient methods to calculate this number (i.e., a dynamic programming solution), however we want this computation to take a while (to show off the program working concurrently), as such you must use the recursive definition above to calculate the result.

2.2 Ellipse Area Approximation

The area of an ellipse $x^2/a^2 + y^2/b^2 = 1$ is πab, where a is the length of semi-major axis, b is the length of semi-minor axis, and $\pi = 3.1415926\ldots$. Because of symmetry, the area of ellipse $x^2/a^2 + y^2/b^2 = 1$ is four times the area of the first quadrant, where $x \geq 0$ and $y \geq 0$.

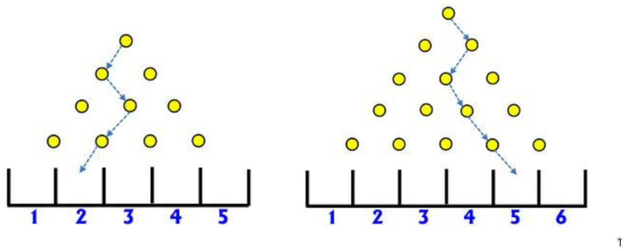
If we randomly pick s points in the rectangle bounded by the x-axis, y-axis, the vertical line $x = a$, and the horizontal line $y = b$, and find out t of them are in the area of the given ellipse (i.e., $x^2/a^2 + y^2/b^2 \leq 1$), then the ratio t/s suggests that the area of the given ellipse in the first quadrant is approximately t/s of that of the rectangle with length a and height b (i.e., $(t/s)ab$). Therefore, $4((t/s)ab)$ should be close to πab.

This child process will do the following:

1. Generate two random numbers x and y where $0 \leq x < a$; and $0 \leq y < b$. This (x, y) represents a point in the rectangle.
2. If $x^2/a^2 + y^2/b^2 leq 1$, then (x, y) is in the given ellipse.
3. Repeat steps 1 and 2 s times.
4. Let the total number of points found to be in the given ellipse be t.
5. Then, $4((t/s)ab)$ should be close to πab (i.e., $4(t/s)4)$ being approximately π), the desired result. This is especially true for very large values of s.
6. Print these results.

2.3 Pinball Game

We can use random numbers to simulate a very simple pinball game. At the bottom of this game are a number of equally sized bins, numbered as $1, 2, 3, \ldots, x$, where x is an input positive integer larger than 1. Above these bins are rows of pins as shown in the diagrams below. If there are x bins, then the last row has $x - 1$ pins as shown in the diagrams above. Above this, there is a row of $x - 2$ pins, etc. Continue this process, the top row has only 1 pin. In this way, we have exactly $x - 1$ rows of pins.



Suppose we drop a ball vertically to the top pin. In the idealized case, when the ball hits each pin, it may go left or right with equal probability (i.e., 0.5). In this way, when the ball hits the top bin, it goes either to the left or to the right. You may also assume that when a ball is falling, it will always hit either the left pin or the right pin below it. As a result, a ball can only go left or right until it finally falls into a bin. In the left diagram above, the ball goes to left, right, left and left, and finally falls into bin 2. In the right diagram, the ball goes right, left, right, right and right, and finally falls into bin 5.

Here is what this process must do:

1. Get x, the number of bins, and y, the number of balls to be dropped, from the command line arguments.
2. Drop y balls using the process described earlier and tally the number of balls that end up in each bin.
3. Print out the number of balls in each bin, the percentage of each bin, and a histogram.

3 Other Requirements

You are to write a C program using the `fork()`, `wait()`, and `exit()` system calls to concurrently perform these computations.

3.1 Input

Input to the program is taken via command line parameters. The program is invoked as

```
./prog1 n a b s x y
```

where `n` is the n-th Fibonacci number to calculate, `a` and `b` are the semi-major and semi-minor axes of the ellipse $x^2/a^2+y^2/b^2=1$, `s` is the number of random points to generate to approximate the area of said ellipse, `x` is the number of bins in the pinball game where `y` balls are dropped. You may assume that all numbers are positive, as well as $n > 2$, $x > 1$, and $a \geq b$.

3.2 Output

Each of the 4 processes will produce different outputs. In order to more easily distinguish which process is printing what, each process will start its output on a different column.

The Fibonacci process will produce the following output, **starting on column 4**:

```
Fibonacci Process Started
Input Number N
Fibonacci Number f(N) is RESULT
Fibonacci Process Exits
```

where `N` is the input number n, and `RESULT` is the n-th Fibonacci number. Since the calculated number could be very large, use the `%ld` formatting string to print the results. For example:

```
Fibonacci Process Started
Input Number 10
Fibonacci Number f(10) is 55
Fibonacci Process Exits
```

The ellipse process will print starting on the **10th column**, producing the following output:

```
Ellipse Area Process Started
Total random Number Pairs S
Semi-Major Axis Length A
Semi-Minor Axis Length B
Total Hits HIT-COUNT
Estimated Area is APPROXIMATION
Actual Area is ACTUAL-AREA
```

where `S`, `A`, and `B` are the aforementioned inputs, `HIT-COUNT` is the number of randomly generate points that landed in the ellipse, `APPROXIMATION` is the approximated area of the ellipse, and `ACTUAL-AREA` is the area calculated via πab. For example:

```
Ellipse Area Process Started
Total random Number Pairs 200000
Semi-Major Axis Length 6
Semi-Minor Axis Length 2
Total Hits 156899
Estimated Area is 37.65576
Actual Area is 37.69911
```

The pinball game simulation will produce output similar to the following, **starting on the first column**:

```
Simple Pinball Process Started
Number of Bins 6
Number of Ball Droppings 3000000
  1-(  94188)-( 3.14%)|*****
  2-( 468531)-(15.62%)|*************************
  3-( 937169)-(31.24%)|**************************************************
  4-( 937427)-(31.25%)|**************************************************
  5-( 468535)-(15.62%)|*************************
  6-(  94150)-( 3.14%)|*****
Simple Pinball Process Exits
```

In the output histogram above, the first column shows the bin numbers printed with `%3d`; the second has the number of balls falling to each bin, because the number of balls can be very large these numbers are printed using `%7d` or `%7ld` depending on the data type you choose to use; and the "percentage" column uses `%5.2f` as the numbers are percentages. The histogram chart requires some explanation. The length (or the number of asterisks) of each horizontal bar is scaled so that the largest percentage is printed using 50 asterisks. Note that it is possible that if a percentage is sufficiently small, there may not be any asterisks shown as after scaling the number of asterisks to be used is less than 1.

Finally, the main process will print to the first column. When the main process beings, it will start by printing the arguments passed to it, like so:

```
Main Process Started
Fibonacci Input            = 10
Total random Number Pairs  = 200000
Semi-Major Axis Length     = 6
Semi-Minor Axis Length     = 2
Number of Bins             = 6
Number of Ball Droppings   = 3000000
```

After it has spawned a child it will print a message corresponding to that child:

```
Fibonacci Process Created
Ellipse Area Process Created
Pinball Process Created
```

After all children have been spawned, and the main process starts to wait for them to die, it will print:

```
Main Process Waits
```

Finally, when all children have died, and the main process is no longer waiting, it will print the following, and exit:

```
Main Process Exits
```

It is very important to remember that **all processes must run concurrently**, and as such may print their output lines in an unpredictable order. As a result, the output lines from a process may mix with output lines from other processes. This is the reason that proper indentation is required to know who prints what. To illustrate this, see the example output.

Finally, make sure that each line prints atomically, so a line from one process will not contain the output from a different process. There are several ways to go about this, but we recommend you `snprintf()` to a buffer, and then `write()` said buffer to `stdout`.

4 Submission Guidelines

The following sections elaborate submission requirements not directly related to the semantics of the program itself.

4.1 General Rules

1. All programs must be written in C, programs written in other languages will receive a 0.
2. Your work must be submitted as a `.zip` file via canvas. This `.zip` will contain your source code, your `Makefile`, and your `README`.
3. Your `Makefile` should produce an executable file named `prog1`. We will simply run the command `make` to build your program.
4. When your program is being tested, we will redirect standard out to a file (to make grading easier), like so:

   ```
   ./prog1 n a b s x y > output.txt
   ```

   Please be sure to test your program with redirecting `stdout`, as some output functions (e.g. `printf()`) behave differently when `stdout` is redirected.

   This procedure may be repeated a number of times with different input values to see if your program behaves correctly.
5. Your implementation should fulfill the program specification as stated.

4.2 Program Style and Documentation

- At the top of the `prog1.c` file, the first comment should be a program header to identify yourself like so:

  ```
  // -----------------------------------------------------------
  // NAME : John Smith User ID: xxxxxxxx
  // DUE DATE : mm/dd/yyyy
  // PROGRAM ASSIGNMENT #
  // FILE NAME : xxxx.yyy
  // PROGRAM PURPOSE :
  // A couple of lines describing your program briefly
  // -----------------------------------------------------------
  ```

  Here, User ID is the one you use to log in to MTU systems.
- Your programs must contain a reasonable amount of concise and to-the-point comments. Do not write a novel.
- Your program should have good indentation.

4.3 The `README` File

A `README` file should be included with your submission. This file must either be a plaintext file named `README`, or a PDF file named `README.pdf`.

You are to answer the following questions in your `README`. Please make sure to clearly indicate which question you are answering in your document.

1. Draw a diagram showing the parent-child relationship if the following program is run with command line argument 4. How many processes are created? Explain step-by-step how these processes are created, especially who is created by whom.

```
void main(int argc, char **argv) {
  int i, n = atoi(argv[1]);

  for (i = 1; i < n; i++)
    if (fork())
      break;
  printf("Process %ld with parent %ld\n",
         getpid(),
         getppid());
  sleep(1);
}
```

2. Draw a diagram showing the parent-child relationship if the following program is run with command line argument 4. How many processes are created? Explain step-by-step how these processes are created, especially who is created by whom.

```
void main(int argc, char **argv) {
  int i, n = atoi(argv[1]);

  for (i = 0; i < n; i++)
    if (fork() <= 0)
      break;
  printf("Process %ld with parent %ld\n",
         getpid(),
         getppid());
  sleep(1);
}
```

3. Draw a diagram showing the parent-child relationship if the following program is run with command line argument 3. How many processes are created? Explain step-by-step how these processes are created, especially who is created by whom.

```
void main(int argc, char **argv) {
  int i, n = atoi(argv[1]);

  for (i = 0; i < n; i++)
    if (fork() == -1)
      break;
```

```
        printf("Process %ld with parent %ld\n",
                getpid(),
                getppid());
        sleep(1);
    }
```

4.4 Submission File Structure

Your submission should follow this file structure:

```
submission/
  prog1.c # main program implementation
  README{.pdf,.txt,}
  Makefile
```

Please do not write code in files outside the ones listed above, as you may lose points for not following the program file structure. Remember that file names in UNIX-like systems are case-sensitive, so `README.txt` is a different file from `readme.txt`!

5 Collaboration Policy

Collaboration is NOT allowed on this programming assignment. This includes the "Empty Hands" policy.

6 Final Remarks

Start early, so that if you're having issues you can seek help before the program is due!

If you are unfamiliar with `fork()` and friends, I suggest that you write and debug code that executes the functionality of each child first. Then create a main process that creates and waits for three children. Finally merge these two into one program, as your final submission.

Finally, run through this quick checklist before and after you submit!

- My submission consists entirely of my own work, and I know I risk failing the entire course if I have plagiarized from any source.
- I am submitting the correct program to the correct assignment on canvas.
- My program follows the program file structure described in 4.4.
- I have included a `Makefile`.
- I have included my `README` file, and it is either a plaintext or PDF file.

- I have tested to make sure my `Makefile` works correctly, and successfully compiles the program using the command given in section 4.1.
- I have tested my program to ensure it works with several inputs, including the given example input.
- My program's output conforms to the output requirements described in section 3.2.
- My program takes input as command line parameters, as described in section 3.1.
- My program is written in C.
- I have tested both compiling and running my program on a CS lab machine, or `colossus.it.mtu.edu`.
- My program runs the child processes concurrently. I do not `fork()` then immediately `wait()` on each child process individually!