Homework 2

CSCE 5640 Operating System Design, Fall 2024

Due 10/19/2024 at 11:59pm

All programs must compile and execute on the CSE machines! This is an individual assignment.

1. (20) The Collatz conjecture concerns what happens when we take any positive integer *n* and apply the following algorithm:

$$n = \begin{cases} n/2 & if\ n\ is\ even \\ 3 \times n + 1 & if\ n\ is\ odd \end{cases}$$

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if *n* = 35, the sequence is

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Write a C++ program using the **fork()** system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if **8** is passed as a parameter on the command line, the child process will output **8, 4, 2, 1**.

Design this program to establish a shared-memory object between the parent and child processes. This technique allows the child to write the contents of the sequence to the shared-memory object. The parent can then output the sequence when the child completes. Because the memory is shared, any changes the child makes will be reflected in the parent process as well.

This program will be structured using POSIX shared memory as described in Section 3.7.1. The parent process will progress through the following steps:

a) Establish the shared-memory object (**shm_open()**, **ftruncate()**, and **mmap()**).
b) Create the child process and wait for it to terminate.
c) Output the contents of shared memory.
d) Remove the shared-memory object.

One area of concern with cooperating processes involves synchronization issues. In this exercise, the parent and child processes must be coordinated so that the parent does not output the sequence until the child finishes execution. These two processes will be synchronized using the **wait()** system call: the parent process will invoke **wait()**, which will suspend it until the child process exits.

Add error checking code, comment your code, and submit it in a file named **collatz.cpp**

<u>Sample outputs:</u>
$./collatz
Syntax program num

$./collatz -1
Please provide a positive integer.

$./collatz 35
35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

$/collatz 50
50, 25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

2. (20) An interesting way of calculating π is to use a technique known as ***Monte Carlo***, which involves randomization. This technique works as follows:

   Suppose you have a circle inscribed within a square, as shown in Figure 4.25. (Assume that the radius of this circle is 1.)

   a. First, generate a series of random points as simple (x, y) coordinates. These points must fall within the Cartesian coordinates that bound the square. Of the total number of random points that are generated, some will occur within the circle.
   b. Next, estimate π by performing the following calculation:

      π = 4 × (number of points in circle) / (total number of points)

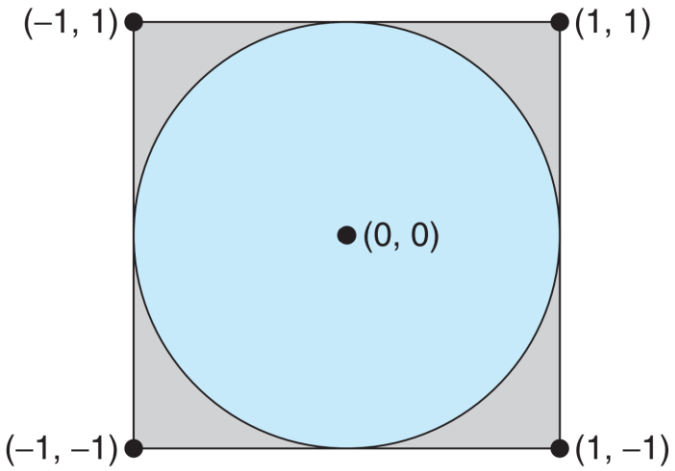

Figure 4.25 Monte Carlo technique for calculating π.

Write a multithreaded version of this algorithm in C++ using POSIX threads that creates a separate thread to generate a number of random points. The thread will count the number of points that occur within the circle and store that result in a global variable. When this thread has exited, the parent thread will calculate and output the estimated value of π. It is worth experimenting with the number of random points generated. As a general rule, the greater the number of points, the closer the approximation to π.
You can use **rand()** function to generate random numbers. Also, use **srand()** to seed the random generator so that you generate different random numbers every time you run the program.
Comment your code and submit it in a file named **computepi.cpp.**

<u>Sample outputs:</u>
$./computepi
Syntax program num

$./computepi 0
Please provide a positive number of points.

$./computepi 1000
Total count of points inside the circle = 759
Value of PI using Monte Carlo technique with 1000 random points = 3.036

$./computepi 1000000
Total count of points inside the circle = 777423
Value of PI using Monte Carlo technique with 1000000 random points = 3.10969

$./computepi 1000000
Total count of points inside the circle = 777423
Value of PI using Monte Carlo technique with 1000000 random points = 3.11016

3. (40) Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
*P*1	2	2
*P*2	1	1
*P*3	9	4
*P*4	4	2
*P*5	6	3

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 2).
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?
c. What is the waiting time of each process for each of these scheduling algorithms?
d. Which of the algorithms results in the minimum average waiting time (over all processes)?

4. (20) Consider two processes, *P*1 and *P*2, where *P*1 has a period of $p1 = 50$ and CPU burst $t1 = 30$. For *P*2, the corresponding values are $p2 = 75$, and $t2 = 25$.

a) Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 5.21–Figure 5.24.
b) Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

Submissions:

- We will be using an electronic homework submission on Canvas to make sure that all students submit their programming tasks on time. You will submit both the program source code file(s) to the **Homework 2 dropbox** on Canvas by the due date and time.
- Make sure you submit the following files for this homework (do not submit a zip file):
    - **collatz.cpp**
    - **computepi.cpp**
    - **hw2sol.pdf** (which contains answers to questions # 3 and 4)
        - Please write your answers in a word document and convert them into the pdf. You can easily use tables and shapes in the word document to design the Gantt charts.
        - If you decide to use paper and scan it to make the pdf document, make sure it is scanned clearly (with clear answers and without any blur) otherwise you may not get the points.
- Program submissions will be checked using a code plagiarism tool against other solutions, including those found on the Internet, so please ensure that all work submitted is your own. Any student determined to have cheated will receive an **'F'** in the course and will be reported for an academic integrity violation.
- Your programs will be tested on our Linux CSE machines. So, make sure you test your programs on the CSE machines fully before you submit on canvas.
- Until you are comfortable working on our Linux CSE machines, as a safety precaution, do not edit your program (using vim or nano) after you have submitted your program where you might accidentally re-save the program, causing the timestamp on your file to be later than the due date. If you want to look (or work on it) after submitting it, make a copy of your submission and work off that copy. Should there be any issues with your submission, this timestamp on your code on the CSE machines will be used to validate when the program is completed.