Alexandria University
Faculty of Engineering
Comp. & Comm. Engineering
CC373: Operating Systems

جامعة الاسكندرية
كلية الهندسة
برنامج هندسة الحاسب والاتصالات
مادة نظم التشغيل

Lab5

Part1 : Producer/Consumer Problem

Objectives

- Learn how to use semaphores for mutual exclusion and synchronization.
- Learn how to use shared memory between processes.

Introduction

Each student is required to implement the Bounded-Buffer Producer/Consumer problem as learnt in the lectures. You may have up to 20 producers. Each Producer would declare the live price of a commodity (e.g., GOLD, SILVER, CRUDEOIL, COTTON, …). One Consumer would show a dashboard for the prices of all commodities. Producers and Consumer would run indefinitely sharing the prices through shared memory.

Producers

Each producer is supposed to continuously declare the price of one commodity. For simplicity, we assume that each commodity price follows a normal distribution with parameters (μ, σ^2). Therefore, producers will generate a new random price, share it with the consumer, and then sleep for an interval before producing the next price.

All producers are processes running the same codebase. Producers are to be run concurrently, either in separate terminals, or in the background. While running a producer, you will specify the following command line arguments:

- Commodity name (e.g., GOLD – Assume no more than 10 characters.)
- Commodity Price Mean; μ – a double value.
- Commodity Price Standard Deviation; σ – a double value.
- Length of the sleep interval in milliseconds; T – an integer.
- Bounded-Buffer Size (number of entries); N – an integer.

Therefore, the command "./producer NATURALGAS 7.1 0.5 200 40" would run a producer that declares the current price of Natural Gas every 200ms according to a normal distribution with parameters (mean=0.5 and variance=0.25). The size of the shared bounded buffer is 40 entries.

Let the producer log what he does by writing to `stderr`. For instance, a producer may write:

```
[12/31/2022 21:45:20.356] GOLD: generating a new value 1820
[12/31/2022 21:45:20.822] GOLD: trying to get mutex on shared buffer
[12/31/2022 21:45:21.128] GOLD: placing 1820 on shared buffer
[12/31/2022 21:45:22.596] GOLD: sleeping for 200 ms
[12/31/2022 21:45:22.635] GOLD: generating a new value 1830
[12/31/2022 21:45:22.984] GOLD: trying to get mutex on shared buffer
[12/31/2022 21:45:25.353] GOLD: placing 1830 on shared buffer
[12/31/2022 21:45:26.142] GOLD: sleeping for 200 ms
```

You may get the current time in nanosec resolution through the `clock_gettime()` system call.

Consumer

The consumer is to print the current price of each commodity, along the average of the current and past 4 readings. An Up/Down arrow to show whether the current Price (AvgPrice) got increased or decreased from the prior one. Until you receive current prices, use `0.00` for the current price of any commodity.

For simplicity, let's limit the commodities to GOLD, SILVER, CRUDEOIL, NATURALGAS, ALUMINIUM, COPPER, NICKEL, LEAD, ZINC, MENTHAOIL, and COTTON. Show the commodities in alphabetical order. While running the consumer, you will specify the following command line argument:

Bounded-Buffer Size (number of entries); N – an integer.

Below is an example output.

```
---+-------------------------------------------------------   +
  | Currency     |  Price   | AvgPrice |
---+-------------------------------------------------------   +
  | ALUMINIUM    |    0.00  |   0.00   |
  | COPPER       |    0.00  |   0.00   |
  |                   ...                |
  | GOLD         | 1810.31↑ | 1815.25↑ |
  |                   ...                |
  | SILVER       |   22.36↓ |  22.80↓  |
  | ZINC         |    0.00  |   0.00   |
---+-------------------------------------------------------   +
```

Required System Calls

You are to use System V IPC for interprocess communication. It provides facility for

Shared Memory; see this example.

Semaphores; see this example.

Hints

You may use the standard C++ Random library to generate normally distributed random variables. An example can be found in its reference.

You may use escape sequences to clear the screen and place the cursor at the top of the screen. It is platform independent.

```
printf("\e[1;1H\e[2J");
```

You may use escape sequences to move the cursor to a certain position. For instance, the following statement would move the cursor to line 5 and column 10.

```
printf("\033[5;10H");
```

You may use Linux terminal commands to print color text on the terminal. See this example.

Use the format specifier `"7.2lf"` while printing prices.

Part2 : *Conway's Game of Life*

description :

Conway's Game of Life is a cellular automaton devised by mathematician John Conway in 1970. It consists of a grid of cells, each of which can be either alive (1) or dead (0). The evolution of the grid over discrete time steps is determined by a set of simple rules that take into account the states of neighboring cells. Despite its simplicity, the Game of Life can produce complex and interesting patterns over time.

Please watch this video : conway game explained

Barrier :

A barrier is a synchronization mechanism that ensures all threads reach a certain point before any of them can proceed. This is particularly useful in the Game of Life, where the state of the grid must be updated simultaneously by all threads.

Shapes Representation :

We will simulate three specific shapes using the rules of Conway's Game of Life. Each shape represents a different type of behavior: (3 shapes will be given to you in the code)

Still Life: A stable configuration that does not change from one generation to the next (e.g., a square).
Oscillator: A configuration that alternates between different states over time (e.g., a vertical line that becomes a horizontal line).
Spaceship: A pattern that moves across the grid over successive generations (e.g., a glider

Simulation Rules :

1. **Birth Rule:** A dead cell (0) will become alive (1) in the next generation if it has exactly 3 live neighbors.
2. **Survival Rule:** A live cell (1) will remain alive (1) in the next generation if it has 2 or 3 live neighbors.
3. **Death Rule:** A live cell (1) will die (become 0) in the next generation if it has fewer than 2 live neighbors (underpopulation) or more than 3 live neighbors (overpopulation).

Generations

- **Definition:** A generation in Conway's Game of Life refers to a single time step in the evolution of the grid. After applying the birth,death and survival rules to the current grid, the resulting configuration grid becomes the next generation.
- **Process:**
  - The game starts with an initial configuration (the first generation given).
  - Each subsequent generation is calculated based on the state of the current generation.
  - This process continues indefinitely or until a predefined number of generations is reached.

Task

Implement the function ***compute_next_gen*** to evolve the grid based on the rules of Conway's Game of Life.

Thread Responsibilities:

- The grid consists of 20 rows.
- Four threads will be created, with each thread responsible for calculating the state of cells in a subset of 5 rows.

Example:

- Thread 1: Rows 0-4
- Thread 2: Rows 5-9
- Thread 3: Rows 10-14
- Thread 4: Rows 15-19

Barrier Synchronization:

- Implement a barrier to ensure all threads complete their calculations before proceeding to the next generation.
- This synchronization prevents threads from moving on until every thread has finished updating its assigned rows.

Givens :

Constants:

- GRID_SIZE = 20
- GRID[GRID_SIZE][GRID_SIZE]
- NUM_THREAD 4
- GENERATIONS = 32

Functions:

- *initialize_patterns*: This function will set up the initial shapes to be tested in the simulation, no need to call it .
- *printer_grid*: This function will be called at the end of each generation to visualize the current state of the grid , **only one thread will call it after barrier synchronization**

Expected Output

Link

Deliverables

1. The directory containing your code should have four files:
   `producer.cpp` file whose main function accepts the arguments as described above.
   `consumer.cpp` file whose main function accepts the arguments as described above.
   `Makefile` that will build the two files and create two output files `producer` and `consumer`.
   game.c file

2. Step outside the directory containing your code.
3. Name your work directory as id1-id2-id3-lab# (for id: 6000-7000-8000 and lab#5, the directory should be called 6000-7000-8000-lab5)
4. Create a .tar.gz file using
5. `tar cvfz ids...-lab5.tar.gz <path_of_directory>`
6. Append .pdf to the filename to be able to upload your project

Submission Link : submission form

Deadline : December 8th, 2024

Notes

- Languages used: C/C++.
- Students will work in a group of 3
- Groups may talk together on the algorithms or functions being used but are **NOT** allowed to look at anybody's code.
- Revise the academic integrity note found on the class web page.