CS 370

Generator and Consumer, Synchronization

Assignment Review

- You are supposed to implement a solution to the Generator and Consumer problem, using a circular FIFO buffer.
- There will be at least two Consumers and at least two Generators.
- **Generators**: are supposed to generate a certain number of prime numbers. The prime number must be chosen randomly between 3-th prime and 31-th prime (both inclusive = [3, 31]). It also keeps track of the sum of all the prime numbers produced.
- **Consumers**: are supposed to consume the elements, produced by the Generators. Each consumer will keep the sum of the elements which have consumed.
- Both, Generators and all Consumers, are supposed to report the prime numbers generated/consumed along with the index and timestamp with nanosecond resolution.

Which files are required?

- Coordinator.java
- Generator.java
- Consumer.java
- Buffer.java
- Makefile
- README.txt

Coordinator.java

- Set the buffer size, total number of items, number of generators, and number of consumers randomly (The ranges are specified in the HW5 description).
- It creates one instance of the buffer, creates required number of threads of generators, creates required number of threads of consumers, and then waits for all of them to finish.
- Once all threads terminate, we get the sum of prime numbers by each of the Generators and the prime numbers generated by each of the consumers.
- Essentially, all the generated elements must be consumed. However, they may be out of order.

Generator.java

- The Generators will produce the total number of elements which is chosen randomly by the second argument (= Seed).
- The buffer which is created by Coordinator is passed as the first argument.
- The number of items assigned to each generator is passed as the second argument and is the same as the total number of item / number of generators if it is divisible. If it isn't perfectly divisible, then the last generator will take the remains.
- An identification is passed as the third argument (begin with 1 and increment) to identify each Generator.
- A seed 'PrimeSeed' is used to the fourth argument and is for generating a random prime number (details in the next slide).
- Insert the random prime number into the buffer.
- A generator cannot insert an element into the buffer when the buffer is full.
- If the number is inserted successfully, it is added a member variable.

Generator.java

1. Import 'java.util.Random' at the beginning of this file.
2. Get the fourth argument which is a prime seed and make an instance of the Random class using the seed.
3. Save the instance into a member variable.
4. When each generator need to produce an item, generate a random number *N* between 3 and 31 (both inclusive).
5. Use the *N* to find *N*-th prime number.
6. Insert the *N*-th prime number into the buffer.

Consumer.java

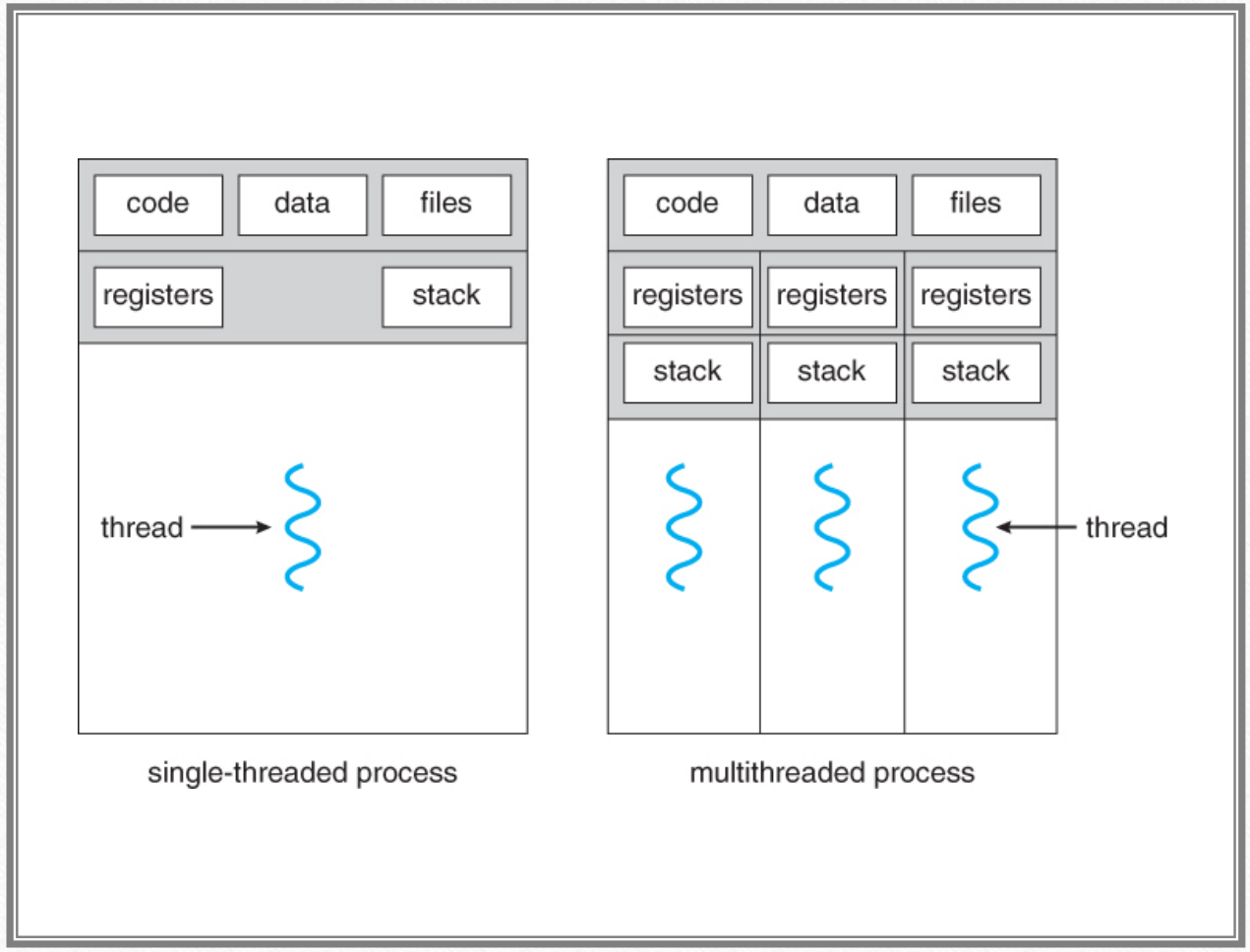
- A consumer consumes an element from the buffer.
- Each consumer will consume a ratio of the total elements (number of elements / number of consumers) if it is evenly divisible.
  - If not, the last consumer will take the remains.
- A consumer cannot consume an element when the buffer is empty.
- Once the consumer consumes an element from the buffer successfully, it is added into a member variable of the consumer.

Buffer.java

- `Buffer.java` contains the circular FIFO buffer that will be used among all the producers and the consumers
- It also has the required functions that is used to insert or remove an element, and it returns the appropriate values.
- It may additionally have other functions such as isFull(), isEmpty(), etc. depending on your implementation.

Synchronization in Java

- Java has inbuilt monitors
  - Allows threads to have mutual exclusion
  - Allows threads the ability to wait (block) for a condition to become true
- Signaling is done using
  - wait()
  - notify() or notifyAll()
- Built in thread class can be extended and used
  - Instantiate and use myThread.start()
  - @Override run() to change what a thread does



Threads

```
public class PhilosopherThread
extends Thread
{
    @Override
    public void run()
    {
        // Thread entry point
    }
}
```

Creating and Starting threads

```
public class PhilosopherThread extends Thread {
    @Override
    public void run()  {
        // Thread entry point
    }
}
```

```
PhilosopherThread Socrates = new PhilosopherThread(table, seat);

Socrates.start(); //begins Socrates thread invokes the run() method
```

Synchronized methods

- A piece of logic marked with synchronized becomes a synchronized block, allowing only one thread to execute at any given time.

```
public synchronized void pickup(int i) throws InterruptedException
{
    //Synchronized code goes in here
}
```

wait(), notify() and notifyAll()

- wait()
  - Causes current thread to wait until another thread invokes the notify() or notifyAll() method
- notify()
  - notify() wakes up one thread waiting for the lock
- notifyAll()
  - The notifyAll() method wakes up all the threads waiting for the lock; the JVM selects one of the threads from the list of threads waiting for the lock and wakes that thread up

Demo

- Demo of DiningPhilosophers from self-expercise in Teams.

CS 370

Raspberry Pi

Topics

- Intro to Raspberry Pi
- Setting up a Raspberry Pi
- Term Project Requirements
- Term Project Expectations
- Helpful Links

Why Raspberry Pi's

- Small and Portable
- Cheap
- Well-Documented
- Versatile
- Support for many peripherals (thanks to Linux)

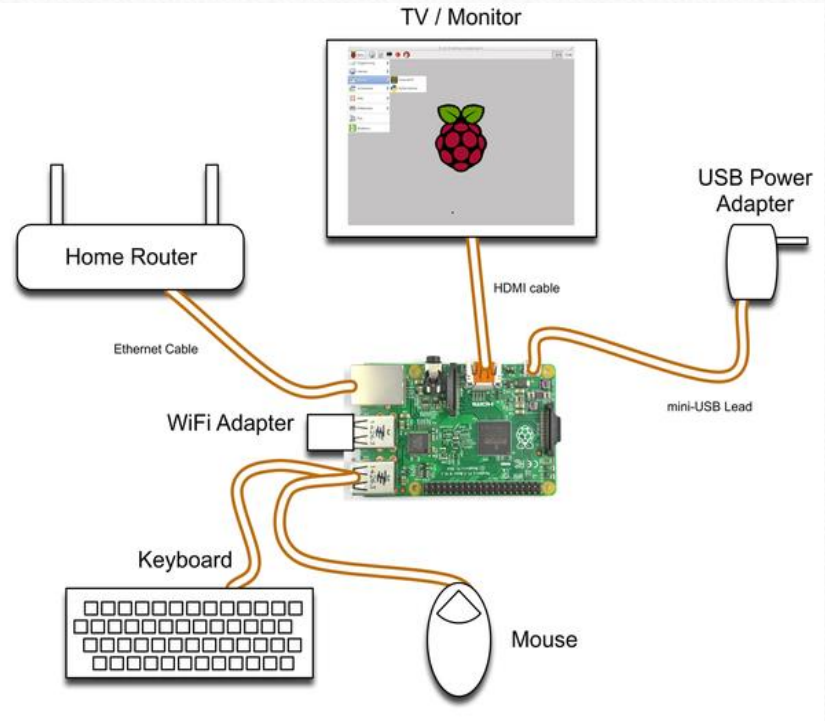
Third Best Selling Computer Brand in the World

Raspberry Pi Models

Raspberry Pi 4 Model B+

- 1.5GHz 64-bit quad-core processor
- dual-band wireless LAN
- Bluetooth 5.0/BLE
- Gigabit Ethernet
- Power-over-Ethernet support (with separate PoE HAT)
- 2 x micro-HDMI ports (up to 4kp60 supported)

Raspberry Pi Setup

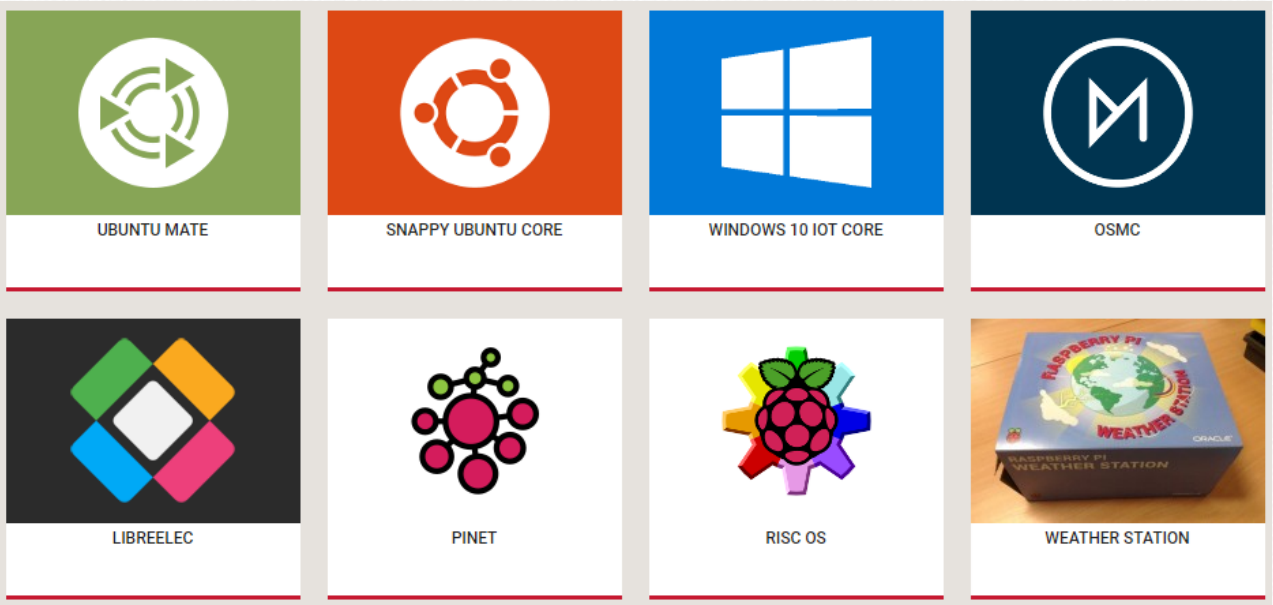


Can connect to monitor, keyboard, mouse

Usable as a normal desktop

Optionally use *ssh* instead of a monitor

Raspberry Pi Operating Systems



Expect most groups to use Raspbian (officially supported OS)

Other options are available - some OS's for specific use cases

Programming Languages

Basically any language will work (Python, C, Java, C++, Javascript, Ruby, Lisp, Rust, R, etc…)

Most projects done in Python or C

GPIO Libraries

Python/C

- RPi.GPIO (Python)
  - RPi.GPIO code samples
- RPIO.GPIO (Python)
- wiringPi (Python/C)
- pigpio (Python/C/Javascript)
- gpiozero (Python)
- bcm2835 (C)

Term Project Requirements

Project must involve:

- A single board computer (Raspberry Pi)
  - With WiFi capability + operating system
- Communication with at least one other computer
  - Another board, desktop, assistant, etc.
- At least one sensing or interacting device
  - Heat sensor, motion detector, camera, motor, controller, etc...

Term Project TODO

- Team Composition and Proposal (done – 5%)
- Progress Report (due on 04/07/2022 - 15%)
- Final Report and Demo
  - Report: 1500 - 2500 words
  - Code
  - 10 - 15 Minute Demo
- Presentation
- Peer Review (5%)

Term Project Expectations

- Originality
  - Several groups with similar projects (temperature sensors, plant waterers, etc...)
  - Come up with a unique selling point
    - Find similar projects online, then do something different
- Thoroughness
  - Think about the evaluations you're performing - design careful experiments and control for variables
  - Try to learn something you couldn't have guessed

Helpful Links

Help Guides

- Setup instructions
- SSH with Raspberry Pi's
- Help videos
- FAQ's
- Embedded Linux wiki

Forums and Tutorials

- Raspberry Pi forums / projects
- Hackaday Projects
- Adafruit Learning Guides
- Raspberry Pi subreddit

Thank You

Questions?