Name: Phạm Đức Đạt

ID: ITITIU20184

1. **Threads: Introductory: Single Thread**

* **Q1. What happens?**

When you press the "Start Thread" button for the RED thread, this thread begins execution. As this is a simulation of asynchronous thread scheduling, the RED thread will attempt to increase the color intensity of randomly chosen pixels to red on the drawing canvas. Since only one thread is running, the entire CPU effort will be dedicated to this task, and you will observe the pixels on the canvas gradually turning red.

* **Q2. Why were the pixels not colored evenly (why did some take longer to be colored red than others)?**

Even though only one thread is running, the uneven distribution of colored pixels can be attributed to several factors, including how the operating system schedules the thread and manages tasks. In asynchronous scheduling, the CPU may randomly choose pixels to intensify in color, meaning some pixels might be selected multiple times before others are selected at all. Additionally, the simulation of CPU and memory operations can also affect the speed at which each pixel is processed, leading to uneven coloring across the canvas.

1. **Threads: Introductory: Two Threads**

* **Q1. What do you think will happen if two threads run at the same time?**

When two threads run simultaneously, each modifying the color intensity of random pixels, you would see a dynamic interplay of color changes on the canvas. Since the RED thread increases the red intensity while decreasing the green and blue, and the GREEN thread increases the green intensity while decreasing the red and blue, the resulting color changes will depend on which thread accesses and modifies a pixel's color values first. This could result in a mix of different colors across the canvas, including shades where both red and green have been increased before either thread could decrease the other's intensity, potentially leading to various shades like yellow, brown, or muted green depending on the exact timing and intensity changes.

* **Q2. What happens?**

When you start the RED thread first, it begins to increase the red intensity and decrease the green and blue intensities of randomly chosen pixels. After a short delay, when the GREEN thread starts, it begins to increase the green intensity and decrease the red and blue intensities. The overlapping actions of these threads on the canvas will lead to complex color dynamics. Some pixels may turn shades of orange or brown as red and green intensities compete, and areas dominated by a particular thread will show a stronger tint of that thread's color.

* **Q3. What happens?**

After stopping the RED thread, the GREEN thread will continue to run alone. Since the RED thread is no longer actively increasing red intensity or decreasing green intensity, the GREEN thread will have a more pronounced effect on the canvas. You should observe a gradual increase in green-dominated areas as the green thread modifies more pixels without the interference of the red thread. Pixels previously influenced by the red thread might start turning greener, and overall, the canvas will start showing more green shades as the influence of the red thread diminishes.

1. **Threads: Introductory: Three Threads**

* **Q1. What do you think will happen if three threads run at the same time? What colour will the drawing canvas appear to be?**

With three threads (RED, GREEN, and BLUE) running simultaneously and each thread trying to increase its respective color's intensity while decreasing the others, the canvas is likely to exhibit a dynamic interplay of colors. If each thread has equal access to CPU time and they effectively counteract each other's color adjustments, the canvas might trend toward white or gray, as these are the combinations when RGB values are balanced or uniformly high.

* **Q2. What happens?**

When you start all three threads in quick succession, each begins to modify the color intensities of random pixels, trying to amplify its color while suppressing the others. The visual result on the canvas will likely be a rapidly changing mosaic of colors, as the threads contend for dominance in their respective color channels.

* **Q3. What happens?**

After stopping the RED thread, the GREEN and BLUE threads continue running. With red no longer being increased or actively suppressing green and blue, you might observe an increase in areas where green and blue are more prominent, potentially shifting the overall hue of the canvas towards cyan, which is a mix of green and blue.

* **Q4. What happens?**

Once you stop the GREEN thread as well, only the BLUE thread remains active. This will likely result in a progressive dominance of blue on the canvas, as there are no more competing threads to increase red or green or to decrease the blue intensity.

* **Q5. Consideration of CPU time:**

In terms of CPU time, when all three threads are running simultaneously, the CPU's time is divided among them. If the scheduling is fair and each thread gets an equal share, they should each have a roughly equal opportunity to modify the pixels. However, the actual allocation can vary based on the scheduling algorithm used by the system, potentially causing slight variations in how often each thread can execute.

* **Q6. Scheduling algorithm:**

The scheduling algorithm that best describes what you see when all three threads are running might depend on the observed behavior. If the threads appear to be running in a smooth, cyclic manner where each gets a turn regularly, it could suggest a Round Robin (RR) approach, especially since this method is known for handling time-sharing systems effectively. Round Robin assigns a fixed time slice to each thread and cycles through them continuously, which supports the observed even distribution of CPU time among the threads. The other options like First-Come, First-Served (FCFS), Shortest Job First (SJF), or Shortest Remaining Job Next (SRJN) are less likely if the threads seem to be treated equally without consideration of their task length or remaining time.

1. **Threads: Introductory: Asynchronous vs Synchronous Thread Scheduling**

* **Q1. Can you clearly see the difference between the two modes of scheduling in terms of the thread behaviors?**
* Asynchronous Thread Scheduling: In this mode, threads are scheduled by the operating system independently of each other, much like separate processes. Threads may execute concurrently if there are multiple processors or interleaved on a single processor. The transitions between threads can be very rapid and frequent, allowing for fine-grained interleaving, which mimics true parallelism.
* Synchronous Thread Scheduling: Here, the application itself controls when threads are run, and typically, the threads do not actually run at the same time but rather take turns executing. This results in more predictable and coarse-grained scheduling where one thread completes a section of work before another thread starts. This can be simpler to manage as the application has full control over the thread execution order.
* **How to explain to a friend:** Imagine you and two other friends are painting a wall, and you have different colors. In asynchronous scheduling, each of you continuously switches between painting small sections randomly, depending on who the supervisor calls next. In synchronous scheduling, each person paints a complete layer of the wall one at a time while the others wait their turn.
* **Q2. In which mode is it easiest to see that there are separate threads running?**

In synchronous scheduling, it's easier to observe that there are separate threads running. This is because each thread takes its turn in a visible sequence, making it clear when one thread stops and another starts. In asynchronous scheduling, the transitions between threads can be so seamless and frequent that it might appear as if the operations are blended together, which can make it harder to distinguish individual thread activities without detailed monitoring tools.

* **Q3. Which mode would be suitable for an application in which each thread dealt with a different query, such as a web-server? Why?**

Asynchronous thread scheduling would be more suitable for an application like a web server where each thread handles a different query. This mode is preferable because:

* Responsiveness: It allows the server to handle multiple requests concurrently or in quick succession, improving responsiveness and throughput.
* Resource Utilization: Asynchronous scheduling can make better use of CPU resources by filling idle time caused by I/O wait or other delays in one thread with processing from another thread.
* Scalability: This mode is more adaptable to varying loads, which is typical in web server environments where request rates can fluctuate significantly.

Asynchronous scheduling supports the dynamic and unpredictable nature of web server requests, where different threads might be handling different types of requests or data and require the ability to operate independently without waiting for other threads to complete their execution cycles.

1. **Threads: Introductory: Thread Priority**
2. **Asynchronous Thread Scheduling**

* **Q1. What is the visual effect of increasing a thread’s priority?**

Increasing a thread's priority in asynchronous scheduling should visually manifest as that thread getting more CPU time compared to others. This would mean it colors more pixels in its color faster than other threads can adjust or modify them.

* **Q2. What actually causes this visual effect (in terms of the thread scheduling)?**

The visual effect is caused by the operating system's scheduler allocating more processing time to the thread with a higher priority. This means the higher priority thread is scheduled more frequently or for longer time slices, allowing it more opportunities to execute its task (coloring the pixels).

* **Q3. What are the effects of changing the priority settings?**

All to HIGHEST: No noticeable difference in the distribution of execution time among the threads, as all are treated equally by the scheduler.

All to LOWEST: Similarly, no difference as all threads are equally prioritized at the lowest level.

One HIGH, others LOW: The thread with high priority will dominate the coloring activity, significantly impacting the canvas more than the lower-priority threads.

* **Q4. Do the priority settings have a relative, absolute, or relative and absolute effect?**

The effect of priority settings in asynchronous scheduling is both relative and absolute. It's absolute in that a higher priority absolutely gets more CPU time than a lower one. It's relative because the effect of a priority change depends on the priorities of other running threads.

1. **Synchronous Thread Scheduling**

* **Q5. What is the visual effect of increasing a thread’s priority? Why is this?**

In synchronous scheduling, increasing a thread's priority using the application's internal mechanism can influence the order or frequency of thread execution within the process but won't change the overall CPU time allocated to the process by the OS.

* **Q6. What is the visual effect of using externally imposed priority?**

Using externally imposed priority, you might not see a significant visual effect since all thread operations still occur within the same process time slice allocated by the OS.

* **Q7. What actually causes this visual effect (in terms of the thread scheduling)?**

In synchronous scheduling, any visual effects from priority changes are due to the application's logic in deciding the order of thread execution, not the OS's scheduling.

* **Q8. What is the effect of changing priority settings under synchronous scheduling?**

All to HIGHEST: No effect since all threads are equally prioritized.

All to LOWEST: Similarly, no effect.

One HIGH, others LOW: The high-priority thread might run more frequently or earlier, according to the application’s scheduling logic, but overall CPU time remains the same.

* **Q9. Do the priority settings have a relative, absolute, or relative and absolute effect?**

In synchronous scheduling, the effect of priority settings is mostly relative, depending on how the application’s logic handles these priorities. It’s not absolute since the OS doesn’t differentiate between the threads.

These experiments highlight how thread priority influences scheduling and execution in different threading models, impacting how tasks (like pixel coloring) are performed in a multi-threaded environment.