**SCREENSHOTS**

**A screenshot of a computer program

Description automatically generated**

**QUESTIONS**

**1) What type of semaphore is this?**

This program doesn't explicitly use semaphores in the traditional sense. However, it involves synchronization mechanisms like **nanosleep()** to manage timing in the main loop, and **initscr()**, **cbreak()**, and **nodelay()** from the **curses** library help manage screen updates and user input without blocking. If you were to introduce semaphores for synchronization in a more complex version of this program, they would likely be **binary semaphores** to control access to the critical sections of the terminal output and the input processing.

**2) Identify the critical section**

The critical section in this program is the part where the screen is updated or where the program writes to the terminal using **mvprintw()** and **refresh()**. If this program were expanded to handle multiple processes or threads, you'd need synchronization to ensure that only one process or thread updates the terminal at a time. This would prevent data corruption or overlapping output in the terminal window.

**3) How many processes are being generated by this program? How many of them are expected to wait?**

This program only generates a single process, which is the main process that runs the program. The terminal input and screen output are handled within this single process. Since the **main\_loop()** runs continuously until the user presses the space bar, the program doesn't spawn additional processes.

No processes are expected to wait because the **nodelay()** function from the **curses** library makes sure the program doesn't block waiting for user input. Instead, it keeps checking for input without pausing, ensuring the program remains responsive. So, there’s no waiting involved in this case.