הנחיות:

• הבחינה עם חומר חום.
• אסף ולהחזר פלאפיים בכתובת הבחיים בכתובת הבחיים.
• ח今まで בקורה מוסדרת נקייה ובריאה בקורה.
• בבחינה 3 שאלות, שקר הבחינה שלוש שאלות.
• כתבו ובדפוס נקייה.
• נמקו את כל הש participação.
• אשר مباشرק הקצה פיתרון אלגוריתמי לאיצוי או בעיה, יש להזמין לקהל פיתרון יעיל לכל היזמים במקומ.

בצלחת!
Mostly Concurrent Copying GC

Mostly Concurrent Copying GC refers to a garbage collection (GC) algorithm that aims to minimize the pauses during memory allocation and deallocation. Unlike traditional GC algorithms that pause the application to perform garbage collection, Mostly Concurrent Copying GC allows the application to continue running during the garbage collection process, with minimal interruptions.

The algorithm works by copying the active generation of memory pages to a new generation, while the old pages are released. This process is done concurrently with the application's operations, hence the name 'mostly concurrent'.

In Mostly Concurrent Copying GC, the garbage collector operates in the background, recognizing the start of a pause when the application's memory usage exceeds a certain threshold. During this pause, the collector copies the active generation of memory pages to a new generation, and releases the old generation.

The key advantage of Mostly Concurrent Copying GC is that it allows the application to continue running during the garbage collection process, which is particularly useful in multi-threaded applications where minimizing downtime is crucial.

The implementation of Mostly Concurrent Copying GC requires careful coordination between the application and the garbage collector. The collector must be able to accurately detect the start of pauses and copy the active generation of memory pages without interfering with the application's operations.

Overall, Mostly Concurrent Copying GC is a significant advancement in garbage collection technology, offering a balance between performance and memory management efficiency.
 incr_barrier: when do you invoke it?

3. does the algorithm respond to read-barrier?

4. what pattern do you observe when two threads race?

5. does the algorithm respond to read-barrier?

6. does the algorithm respond to write-barrier?

7. is the problem of race conditions solved?

8. is the algorithm race free?