1. Process là gì?

* Process (tiến trình) là một chương trình đang chạy trong bộ nhớ
* Mỗi process có:
  + Bộ nhớ riêng (heap, stack)
  + Không gian địa chỉ riêng
  + Tài nguyên(CPU, RAM, file handle…)
* Process độc lập với nhau, muốn giao tiếp phải dùng IPC (Inter)

1. Thread là gì?

* Thread (luồng) là đơn vị thực thi nhỏ nhất trong một process
* Một process có thể chứa nhiều thread cùng chạy song song và chia sẻ chung bộ nhớ của process đó
* Ví dụ: Trong Chrome (1 process), có nhiều thread:
  + 1 thread tải trang web,
  + 1 thread hiển thị giao diện.
  + 1 thread chạy JavaScript

1. Các tạo 1 thread: có 2 cách chính

* Kế thừa từ lớp Thread
  + Khi lớp của bạn không cần kế thứ từ 1 lớp khác
  + Khi chỉ cần tạo 1 một đối tượng và gọi phương thức strat()
* Cài đặt interface Runnable
  + Khi lớp đã kế thừa từ một lớp khác vì java cho phép một lớp triển khai nhiều interface
  + Tách biệt hóa mã chạy trong Thread và việc tạo và quản lý Thread
  + Khi cần chia sẻ tài nguyên giữa các Thread vì bạn có thể chia sẻ cùng một đối tượng Runnable giữa nhiều Thread
* Dùng Callable + Future (trả về kết quả, có exception)
* Khác biệt:
  + Thread: đơn giản nhưng không tái sử dụng được vì Java chỉ hỗ trợ kế thừa 1 lớp
  + Runnable: linh hoạt hơn, tách logic công việc ra khỏi thread, dễ tái sử dụng
  + Callable: giống Runnable nhưng có trả về kết quả và nem exception

1. Thế nào là multi-thread?

* Multi-thread là khi 1 process có nhiều thread cùng chạy song song
* JVM hỗ trợ multi-thread để tận dụng đa lõi CPU
* Ưu điểm:
  + Tận dụng tài nguyên CPU
  + Chương trình chạy nhanh hơn(song song)
  + Dễ tách biệt logic
* Nhược điểm:
  + Lập trình phức tạp (dễ deadlock, race condition)
  + Tốn tài nguyên khi tạo quá nhiều thread
  + Khó debug

1. Làm thế nào để biết được 1 thread/multi-thread đã hoàn thành?
2. Có giới hạn việc tạo ra bao nhiêu thread trong 1 ứng dụng Java không?

* Không có con số cố định trong java
* Giới hạn phụ thuộc vào:
  + Bộ nhớ Heap/stack
  + Cấu hình OS
  + Kích thước stack mỗi thread
* Thông thường:
  + Một ứng ụn có thể tạo hàng nghìn thread, nhưng nếu quá nhiều -> OutOfMemoryError: unable to create new native thread
  + Giải pháp dùng Thread Pool để tái sử dụng thread thay vì tạo mới liên tục

1. Tóm lại:

* Process = chương trình chạy, có bộ nhớ riêng
* Thread = đơn vị thực thi trong process, chia sẻ bộ nhớ
* Có 3 cách tạo thread (Thread, Runnable, Callble)
* Multi-thread = nhiều luồng chạy song song. Ưu: nhanh hơn, Nhược: dễ lỗi
* Kiểm tra thread xong hay chưa -> join() hoặc Future
* Số lượng thread phụ thuộc RAM/OS nên dùng Thread Pool

1. Lợi ích của Thread

* Hiệu năng: sử dụng Thread có thể giúp tận dụng tối đa khả năng đa nhân của bộ xử lý, do đó cải thiện hiệu năng của ứng dụng
* Phản hồi nhanh: một ứng dụng đa luồng cho phép người dùng tiếp tục sử dụng các tính năng khác của ứng dụng trong khi một số tác vụ khác đang chạy ở nền.
* Tận dụng tối đa tài nguyên; Đa luồng giúp chia sẻ tài nguyên hiệu quả giữa các tác vụ, giảm thiểu thời gian chờ đợi và tận dụng tối đa tài nguyên của hệ thống.
* Đơn giản hóa mã nguồn: Chia nhỏ chương trình thành các Thread độc lập giúp mã nguồn dễ hiểu hơn, dễ bảo trì hơn.

1. Vòng đời của Thread trong java

* Mới tạo (new):
  + Mới được khởi tạo nhưng chưa bắt đầu chạy
  + Thread chưa được liên kết với hệ thống phân bổ tài nguyên và chưa được đưa vào hàng đợi CPU
  + Để bắt đầu chạy 1 luồng mới -> gọi strat()
* Chạy (Runnable):
  + Thread đang chờ đợi CPU để thực thi mã của nó
  + Không đảm báo rằng Thread đang thực sự chạy => mà đơn giản là nó đã sẵn sàng để thực thi và đang chờ đợi tài nguyên
* Chờ (Waiting)
  + Trong quá trình chạy => 1 Thread có thể phải chờ dd1 điều kiện nào đó để tiếp tục.
  + Nó không thực thi mã của nó và giải phóng tài nguyên CPU để các Thread khác có thể sử dụng
* Ngủ:
  + Không thực thi mã của nó và giải phóng tài nguyên CPU
  + Ngủ trong 1 khoảng thời gian nhất định => tự động chuyển qua trạng thái chạy
* Kết thúc:
  + Khi hoàn thành công việc hoặc bị dừng bởi một lý do nào đó => chuyển sang dừng
  + Nó không thể tái sử dụng => cần tạo 1 Thread mới nếu thực hiện lại công việc

1. Thứ tự ưu tiên:

* Mỗi thread đều có 1 mức độ ưu tiên -> xác định thứ tự kho phân bổ tài nguyên và thời gian CPU cho cac Thread
* Năm khoảng từ 1 đến 10 -> 1 là thấp nhât và 10 là cao nhất
* Mặc định -> Thread mới có mức ưu tiên bằng Thread cha
* Java cung cấp một số hằng số để đại diện cho các mức ưu tiên thông dụng:
  + Thread.MIN\_PRIORITY: Mức ưu tiên tối thiểu (1).
  + Thread.NORM\_PRIORITY: Mức ưu tiên bình thường (5).
  + Thread.MAX\_PRIORITY: Mức ưu tiên tối đa (10).

1. Đồng bộ hóa trong java

* Là một kỹ thuật được sử dụng để điều khiển việc truy cập đồng thời vào cá tài nguyên chia sẻ trong môi trường đa luồng
* Mục đích: đảm bảo tính nhất quán và an toàn của dữ liệu khi nhiều luồng truy cập vào cùng một thời điểm
* Đồng bộ hóa được thực hiện thông qua việc sử dụng khóa (trên đối tượng)
* Từ khóa synchronized được sử dụng để đảm bảo rằng chỉ có một luồng có thể truy cập vào một phần mã đồng bộ hóa tại một thời điểm. => được áp dụng cho phương thức hoặc khối mã.
* Dùng không đúng cách có thể dẫn đến tình trạng khóa chết => khiến các luồng không tiếp tục thực thi được

1. Một số lỗi trong Thread

* Race condition:
  + Khi nhiều luồng truy cập và thay đổi đồng thời 1 tài nguyên chia sẻ => dẫn đến kết quả không như mong đợi và không đảm vbaor tính nhất quán của dữ liệu
  + Tình trạng này có thể dẫn đến kết quả không chính xác hoặc bất đồng bộ trong chương trình
  + Nguyên nhân:
    - Khi sử dụng các biến dữ liệu không được đồng bộ hóa giữa các Thread
* Thread Deadlock:
  + Xảy ra khi hai hoặc nhiều luồng chờ đợi nhau để giải phóng các khóa mà chúng đang nắm giữ, dẫn đến việc các luồng bị treo vô thời hạn và không thể tiếp tục thực thi
  + Thường xảy ra khi các luồng sử dụng đồng bộ hóa không đúng cách trong môi trường đa luồng
  + Nguyên nhân dẫn đến:
    - Khóa độc quyền: các tài nguyên có thể được sử dụng bởi một luồng tại một thời điểm.
    - Giữ và chờ: một luồng đang giữ một hoặc nhiều tài nguyên và chờ đợi để giành được tài nguyên khác đang được giữ bởi luồng nhau.
    - Không có lệnh hủy bỏ: tài nguyên không thể bị giành lại từ một luồng trừ khi luồng đó giải phóng tài nguyên đó.
    - Vồng chờ đợi: có một chuỗi của luồng đang chờ đợi nhau theo một vòng tròn, trong đó mỗi luồng chờ đợi tài nguyên đang được giữ bởi luồng tiếp theo trong chuỗi

1. f