**Starvation and Livelock**

Starvation and livelock are much less common a problem than deadlock, but are still problems that every designer of concurrent software is likely to encounter.

**Starvation**

*Starvation* describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

**Livelock**

A thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then *livelock* may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked — they are simply too busy responding to each other to resume work. This is comparable to two people attempting to pass each other in a corridor: Alphonse moves to his left to let Gaston pass, while Gaston moves to his right to let Alphonse pass. Seeing that they are still blocking each other, Alphone moves to his right, while Gaston moves to his left. They're still blocking each other, so...

Starvation (computer science)

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Starvation_(computer_science)#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Starvation_(computer_science)#searchInput)

*For other uses, see*[*Starvation (disambiguation)*](https://en.wikipedia.org/wiki/Starvation_(disambiguation))*.*

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **resource starvation** is a problem encountered in [concurrent computing](https://en.wikipedia.org/wiki/Concurrent_computing) where a [process](https://en.wikipedia.org/wiki/Computer_process) is perpetually denied necessary [resources](https://en.wikipedia.org/wiki/Resource_(computer_science)) to process its work.[[1]](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_note-1) Starvation may be caused by errors in a scheduling or [mutual exclusion](https://en.wikipedia.org/wiki/Mutual_exclusion) algorithm, but can also be caused by [resource leaks](https://en.wikipedia.org/wiki/Resource_leak), and can be intentionally caused via a [denial-of-service attack](https://en.wikipedia.org/wiki/Denial-of-service_attack) such as a [fork bomb](https://en.wikipedia.org/wiki/Fork_bomb).

When starvation is impossible in a [concurrent algorithm](https://en.wikipedia.org/wiki/Concurrent_algorithm), the algorithm is called **starvation-free**, **lockout-freed**[[2]](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_note-2) or said to have **finite bypass**.[[3]](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_note-raynal-3) This property is an instance of [liveness](https://en.wikipedia.org/wiki/Liveness), and is one of the two requirements for any mutual exclusion algorithm; the other being [correctness](https://en.wikipedia.org/wiki/Correctness_(computer_science)). The name "finite bypass" means that any process (concurrent part) of the algorithm is bypassed at most a finite number times before being allowed access to the [shared resource](https://en.wikipedia.org/wiki/Shared_resource).[[3]](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_note-raynal-3)

Scheduling[[edit](https://en.wikipedia.org/w/index.php?title=Starvation_(computer_science)&action=edit&section=1)]

Starvation is usually caused by an overly simplistic [scheduling algorithm](https://en.wikipedia.org/wiki/Scheduling_algorithm). For example, if a (poorly designed) multi-tasking system always switches between the first two tasks while a third never gets to run, then the third task is being starved of [CPU time](https://en.wikipedia.org/wiki/CPU_time). The scheduling algorithm, which is part of the [kernel](https://en.wikipedia.org/wiki/Kernel_(computer_science)), is supposed to allocate resources equitably; that is, the algorithm should allocate resources so that no process perpetually lacks necessary resources.

Many operating system schedulers employ the concept of process priority. A high priority process A will run before a low priority process B. If the high priority process (process A) blocks and never yields, the low priority process (B) will (in some systems) never be scheduled—it will experience starvation. If there is an even higher priority process X, which is dependent on a result from process B, then process X might never finish, even though it is the most important process in the system. This condition is called a [priority inversion](https://en.wikipedia.org/wiki/Priority_inversion). Modern scheduling algorithms normally contain code to guarantee that all processes will receive a minimum amount of each important resource (most often CPU time) in order to prevent any process from being subjected to starvation.

In computer networks, especially wireless networks, [scheduling algorithms](https://en.wikipedia.org/wiki/Scheduling_algorithm) may suffer from scheduling starvation. An example is [maximum throughput scheduling](https://en.wikipedia.org/wiki/Maximum_throughput_scheduling).

Starvation is normally caused by [deadlock](https://en.wikipedia.org/wiki/Deadlock) in that it causes a process to freeze. Two or more processes become deadlocked when each of them is doing nothing while waiting for a resource occupied by another program in the same set. On the other hand, a process is in starvation when it is waiting for a resource that is continuously given to other processes. Starvation-freedom is a stronger guarantee than the absence of deadlock: a mutual exclusion algorithm that must choose to allow one of two processes into a [critical section](https://en.wikipedia.org/wiki/Critical_section) and picks one arbitrarily is deadlock-free, but not starvation-free.[[3]](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_note-raynal-3)

A possible solution to starvation is to use a scheduling algorithm with priority queue that also uses the [aging](https://en.wikipedia.org/wiki/Aging_(scheduling)) technique. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.[[4]](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_note-4)

See also[[edit](https://en.wikipedia.org/w/index.php?title=Starvation_(computer_science)&action=edit&section=2)]

* [Dining philosophers problem](https://en.wikipedia.org/wiki/Dining_philosophers_problem)

References[[edit](https://en.wikipedia.org/w/index.php?title=Starvation_(computer_science)&action=edit&section=3)]

* 1. [**^**](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_ref-1) [*Tanenbaum, Andrew*](https://en.wikipedia.org/wiki/Andrew_Tanenbaum)*(2001).*[*Modern Operating Systems*](https://archive.org/details/modernoperatings00tane)*. Prentice Hall. pp.*[*184–185*](https://archive.org/details/modernoperatings00tane/page/184)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-13-092641-8*](https://en.wikipedia.org/wiki/Special:BookSources/0-13-092641-8)*.*
  2. [**^**](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_ref-2) [*Herlihy, Maurice*](https://en.wikipedia.org/wiki/Maurice_Herlihy)*; [Shavit, Nir](https://en.wikipedia.org/wiki/Nir_Shavit" \o "Nir Shavit) (2012). The Art of Multiprocessor Programming. Elsevier. p. 24.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*9780123977953*](https://en.wikipedia.org/wiki/Special:BookSources/9780123977953)*.*
  3. ^ [Jump up to:***a***](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_ref-raynal_3-0) [***b***](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_ref-raynal_3-1) [***c***](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_ref-raynal_3-2) *[Raynal, Michel](https://en.wikipedia.org/wiki/Michel_Raynal" \o "Michel Raynal) (2012). Concurrent Programming: Algorithms, Principles, and Foundations. Springer Science & Business Media. p. 10–11.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*3642320279*](https://en.wikipedia.org/wiki/Special:BookSources/3642320279)*.*
  4. [**^**](https://en.wikipedia.org/wiki/Starvation_(computer_science)#cite_ref-4) *Galvin, Peter (2010). Operating System Concepts. Wiley India Edition. p. 193.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*978-81-265-2051-0*](https://en.wikipedia.org/wiki/Special:BookSources/978-81-265-2051-0)*.*

|  |  |
| --- | --- |
| [Stub icon](https://en.wikipedia.org/wiki/File:LampFlowchart.svg) | *This*[*computer science*](https://en.wikipedia.org/wiki/Computer_science)*article is a*[*stub*](https://en.wikipedia.org/wiki/Wikipedia:Stub)*. You can help Wikipedia by*[*expanding it*](https://en.wikipedia.org/w/index.php?title=Starvation_(computer_science)&action=edit)*.* |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Java - Thread Starvation and Fairness [Last Updated: Feb 7, 2017] | Top of Form   |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | |  |  | |  |   Bottom of Form  [Previous Page](https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/thread-wait-notify.html)[Next Page](https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/thread-livelock.html) |
| main  In multithreaded application starvation is a situation when a thread is constantly ignored to gain possession of the intrinsic lock in favor of other threads.  [From Oracle reference docs:](https://docs.oracle.com/javase/tutorial/essential/concurrency/starvelive.html)   Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.  Let's understand that with an example. We are going to create 5 threads and display progress of each thread on Swing JProgressBar:  public class StarvationDemo {  private static Object sharedObj = new Object();  public static void main (String[] args) {  JFrame frame = createFrame();  frame.setLayout(new FlowLayout(FlowLayout.LEFT));  for (int i = 0; i < 5; i++) {  ProgressThread progressThread = new ProgressThread();  frame.add(progressThread.getProgressComponent());  progressThread.start();  }  frame.setLocationRelativeTo(null);  frame.setVisible(true);  }  private static JFrame createFrame () {  JFrame frame = new JFrame("Starvation Demo");  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  frame.setSize(new Dimension(300, 200));  return frame;  }  private static class ProgressThread extends Thread {  JProgressBar progressBar;  ProgressThread () {  progressBar = new JProgressBar();  progressBar.setString(this.getName());  progressBar.setStringPainted(true);  }  JComponent getProgressComponent () {  return progressBar;  }  @Override  public void run () {  int c = 0;  while (true) {  synchronized (sharedObj) {  if (c == 100) {  c = 0;  }  progressBar.setValue(++c);  try {  //sleep the thread to simulate long running task  Thread.sleep(100);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  }  Output:    As seen above, it's evident that all threads are not given the equal chance to have equal CPU time. Some threads are starving for long time to acquire the shared lock.   Fairness Fairness is the situation when all threads are given equal opportunity for intrinsic lock acquisition.  As we saw in above example that the long running thread has more chances to aquire the lock again consecutively.  In general we cannot predict how underlying thread scheduler (built in O.S.) chooses the next thread for the lock acquisition.  From developer perspective, the code should not hold the lock for a long time to make a thread greedy.  We can fix the above code by using wait() method which releases the lock but goes to waiting state i.e. the scheduler cannot choose it for the lock acquisition again. That way other threads are given the equal opportunity to run.  public class FairnessDemo {  private static Object sharedObj = new Object();  public static void main (String[] args) {  JFrame frame = createFrame();  frame.setLayout(new FlowLayout(FlowLayout.LEFT));  for (int i = 0; i < 5; i++) {  ProgressThread progressThread = new ProgressThread();  frame.add(progressThread.getProgressComponent());  progressThread.start();  }  frame.setLocationRelativeTo(null);  frame.setVisible(true);  }  private static JFrame createFrame () {  JFrame frame = new JFrame("Fairness Demo");  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  frame.setSize(new Dimension(300, 200));  return frame;  }  private static class ProgressThread extends Thread {  JProgressBar progressBar;  ProgressThread () {  progressBar = new JProgressBar();  progressBar.setString(this.getName());  progressBar.setStringPainted(true);  }  JComponent getProgressComponent () {  return progressBar;  }  @Override  public void run () {  int c = 0;  while (true) {  synchronized (sharedObj) {  if (c == 100) {  c = 0;  }  progressBar.setValue(++c);  try {  //simulate long running task with wait..  // releasing the lock for long running task gives  //fair chances to run other threads  sharedObj.wait(100);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  }  Output:   Example Project Dependencies and Technologies Used:   * JDK 1.8 * Maven 3.0.4 | Core Java Tutorials  * [Java 14 Features](https://www.logicbig.com/tutorials/core-java-tutorial/java-14-changes.html) * [Java 13 Features](https://www.logicbig.com/tutorials/core-java-tutorial/java-13-changes.html) * [Java 12 Features](https://www.logicbig.com/tutorials/core-java-tutorial/java-12-changes.html) * [Java 11 Features](https://www.logicbig.com/tutorials/core-java-tutorial/java-11-changes.html) * [Java 10 Features](https://www.logicbig.com/tutorials/core-java-tutorial/java-10-changes.html) * [Java 9 Module System](https://www.logicbig.com/tutorials/core-java-tutorial/modules.html) * [Java 9 Misc Features](https://www.logicbig.com/tutorials/core-java-tutorial/java-9-changes.html) * [Java 9 JShell](https://www.logicbig.com/tutorials/core-java-tutorial/jshell.html)  Recent Tutorials  * [Spring Boot Primefaces Integration](https://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-primefaces-integration.html) * [Spring Boot JSF Integration](https://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-jsf-integration.html) * [Reactor - Creating an instance using Flux.just() Mono.just()](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/creating-flux-or-mono-with-just.html) * [Spring Boot - Testing With @MockBean](https://www.logicbig.com/tutorials/spring-framework/spring-boot/testing-with-mocking-beans.html) * [Java - How to delete old files under a folder if number of files are over a specified limit?](https://www.logicbig.com/how-to/java-io/delete-old-files.html) * [Mockito - BDD Style Verification using then() - should()](https://www.logicbig.com/tutorials/unit-testing/mockito/bdd-style-behavior-verification.html) * [Mockito - BDD Style Stubbing with given() - willReturn()](https://www.logicbig.com/tutorials/unit-testing/mockito/bdd-style-stubbing-with-given-will-return.html) * [Cassandra - Mapping Java Objects using Mapper](https://www.logicbig.com/tutorials/misc/nosql/cassandra/object-mapper-getting-started.html) * [Java - How to convert Calendar to LocalDateTime?](https://www.logicbig.com/how-to/java-8-date-time-api/calendar-to-local-date-time.html) * [Mockito - verifyNoMoreInteractions() and verifyNoInteractions()](https://www.logicbig.com/tutorials/unit-testing/mockito/verify-no-more-interactions.html) * [Mockito - Verifying Multiple Number of Method Invocations](https://www.logicbig.com/tutorials/unit-testing/mockito/verifying-varing-number-of-invocations.html) * [Mockito - Verify Method Calls With Argument Matchers](https://www.logicbig.com/tutorials/unit-testing/mockito/verify-method-calls-with-argument-matchers.html) * [Kafka - ConsumerRebalanceListener Example](https://www.logicbig.com/tutorials/misc/kafka/consumer-rebalance-listener-example.html) * [Kafka - Understanding Partition Rebalancing](https://www.logicbig.com/tutorials/misc/kafka/kafka-partition-rebalancing.html) * [Kafka Manual Commit - commitSync() Example](https://www.logicbig.com/tutorials/misc/kafka/kafka-manual-commit-sync-example.html) * [Kafka Manual Commit - CommitAsync With Callback and Specified Offset](https://www.logicbig.com/tutorials/misc/kafka/kafka-manual-commit-async-with-specific-offset-and-callback.html) * [Kafka Manual Commit - commitAsync With Callback Example](https://www.logicbig.com/tutorials/misc/kafka/kafka-manual-commit-async-with-callback-example.html) * [Java - Introduction to Java 8 Date and Time API](https://www.logicbig.com/tutorials/core-java-tutorial/date-time-api/intro-to-java-date-time-api.html) * [Reactor - Understanding Flux/Mono's retryWhen()](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/understanding-retry-when.html) * [Reactor - Retrying Flux/Mono Sequence](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/reactor-retry-examples.html) * [Spring - Injecting Prototype Bean Using Java 8 Functions](https://www.logicbig.com/tutorials/spring-framework/spring-core/injecting-prototype-bean-using-java-8-functions.html) * [Mockito - Argument Matchers Example](https://www.logicbig.com/tutorials/unit-testing/mockito/mockito-argument-matchers-examples.html) * [Mockito - Argument Matchers](https://www.logicbig.com/tutorials/unit-testing/mockito/mockito-argument-matchers.html) * [Kafka Manual Commit - CommitAsync() Example](https://www.logicbig.com/tutorials/misc/kafka/kafka-manual-commit-async-example.html) * [Java - How to Indent multiline String?](https://www.logicbig.com/how-to/java-string/java-indent-string.html) * [Mockito - Verifying Method Calls](https://www.logicbig.com/tutorials/unit-testing/mockito/verify-method-calls.html) * [Spring MVC - Post Request With Simple Form Submission](https://www.logicbig.com/tutorials/spring-framework/spring-web-mvc/spring-mvc-post-request-with-simple-form-submission.html) * [Java - Parsing String To Numeric Primitives](https://www.logicbig.com/how-to/java-numbers/parse-string-to-primitive-numbers.html) * [Mockito - Stubbing methods with exceptions](https://www.logicbig.com/tutorials/unit-testing/mockito/stubbing-exceptions.html) * [Java - Avoiding possible NullPointerException with method call chain](https://www.logicbig.com/how-to/java/avoid-null-pointer-with-method-chain.html) * [Java - Autoboxing And Unboxing, How to avoid NullPointerException with unboxing?](https://www.logicbig.com/tutorials/core-java-tutorial/java-language/autoboxing-and-unboxing.html) * [Kafka - Auto Committing Offsets](https://www.logicbig.com/tutorials/misc/kafka/auto-committing-offsets.html) * [Kafka - Understanding Offset Commits](https://www.logicbig.com/tutorials/misc/kafka/committing-offsets.html) * [Kafka - Publishing records With null keys and no assigned partitions](https://www.logicbig.com/tutorials/misc/kafka/records-with-null-key-and-no-assigned-partitions.html) * [Java Collections - How to find frequency of each element in a collection?](https://www.logicbig.com/how-to/java-collections/collection-element-frequency.html) * [How to convert java.util.Map To Java Bean?](https://www.logicbig.com/how-to/java-beans/map-to-java-bean.html) * [Java - How to repeat a string n number of times?](https://www.logicbig.com/how-to/java-string/string-repeat-n-times.html) * [Git - Merging Branches](https://www.logicbig.com/tutorials/misc/git/merging-branches.html) * [Java - How to convert Iterator To List?](https://www.logicbig.com/how-to/java-collections/iterator-to-list.html) * [Spring Boot - Unit Testing Application Arguments](https://www.logicbig.com/tutorials/spring-framework/spring-boot/testing-application-arguments.html) * [How to find the longest and the shortest String in Java?](https://www.logicbig.com/how-to/java-string/longest-and-shortest-string.html) * [How to find first and last element of Java 8 stream?](https://www.logicbig.com/how-to/java-8-streams/find-first-and-last-element.html) * [Mockito - Stubbing consecutive method calls](https://www.logicbig.com/tutorials/unit-testing/mockito/stubbing-consecutive-method-calls.html) * [Mockito - Stubbing a method's return Value](https://www.logicbig.com/tutorials/unit-testing/mockito/subbing-method-return-values.html) * [Kafka - Using Keys For Partition Assignment](https://www.logicbig.com/tutorials/misc/kafka/using-keys-for-partition-assignment.html) * [Spring Boot - Using @TestConfiguration In static nested class](https://www.logicbig.com/tutorials/spring-framework/spring-boot/test-configuration-in-nested-class.html) * [Spring Boot - Using @TestConfiguration to define beans for tests](https://www.logicbig.com/tutorials/spring-framework/spring-boot/test-configuration.html) * [Java Collections - Why Arrays.asList() does not work for primitive arrays?](https://www.logicbig.com/how-to/java-collections/arrays-as-list-for-primitives.html) * [Java Collections - Only put Map key/value if the specified key does not exist](https://www.logicbig.com/how-to/java-collections/map-put-if-absent-example.html) * [Getting Started with Mockito](https://www.logicbig.com/tutorials/unit-testing/mockito/mockito-getting-started.html) * [How to connect a Database server in Intellij Community Edition?](https://www.logicbig.com/how-to/intellij/intellij-community-edition-connecting-database.html) * [Java HashMap - Understanding equals() and hashCode() methods](https://www.logicbig.com/tutorials/core-java-tutorial/java-collections/hash-map-equal-and-hash-code.html) * [Java IO - How to write lines To a file and read lines from a files?](https://www.logicbig.com/how-to/java-io/java-read-write-lines-to-files.html) * [Java Collections - How to find distinct elements count in collections and arrays?](https://www.logicbig.com/how-to/java-collections/distinct-elements-count.html) * [Spring Boot - Web Application Testing With Embedded Server And TestRestTemplate](https://www.logicbig.com/tutorials/spring-framework/spring-boot/web-application-testing-with-embedded-server-and-test-rest-template.html) * [Java - How to find Available Runtime Memory?](https://www.logicbig.com/how-to/java/java-runtime-memory.html) * [Kafka - Understanding Consumer Group with examples](https://www.logicbig.com/tutorials/misc/kafka/automatic-partition-assignment-to-consumers.html) * [Spring Boot - Unit Testing Web Application With Embedded Server](https://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-web-application-testing-with-embedded-server.html) * [Java - Different ways to Set Nested Field Value By Reflection](https://www.logicbig.com/how-to/reflection/setting-nested-field-value.html) * [Java - Different ways to Set Field Value by Reflection](https://www.logicbig.com/how-to/reflection/setting-field-value.html) * [Installing Python 2.7 on windows](https://www.logicbig.com/how-to/python/python-27-install.html) * [Installing Cassandra And Intro To CQLSH](https://www.logicbig.com/how-to/cassandra/cassandra-install-and-intro-to-cqlsh.html) * [Cassandra - Getting Started with Java](https://www.logicbig.com/tutorials/misc/nosql/cassandra/getting-started.html) * [Java 14 - Switch Expressions And Statements Examples](https://www.logicbig.com/tutorials/core-java-tutorial/java-14-changes/switch-expressions-and-statements.html) * [Kafka - Manually Assign Partition To A Consumer](https://www.logicbig.com/tutorials/misc/kafka/manually-assign-partition-to-a-consumer.html) * [Kafka - Understanding Topic Partitions](https://www.logicbig.com/tutorials/misc/kafka/understanding-kafka-topic-partitions.html) * [Spring Boot - Unit Testing Web Application With Mock MVC](https://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-web-application-testing-with-mock-mvc.html) * [Kafka - Introduction to Kafka Admin API](https://www.logicbig.com/tutorials/misc/kafka/admin-api-getting-started.html) * [Java 14 - Helpful NullPointerException](https://www.logicbig.com/tutorials/core-java-tutorial/java-14-changes/null-pointer-exception-enhancement.html) * [Java 14 - Records](https://www.logicbig.com/tutorials/core-java-tutorial/java-14-changes/records.html) * [Java 14 - Pattern Matching for instanceof](https://www.logicbig.com/tutorials/core-java-tutorial/java-14-changes/instance-of-pattern-matching.html) * [Spring Boot - Application Unit Testing with @SpringBootTest](https://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-application-testing.html) * [Installing and Running Kafka](https://www.logicbig.com/how-to/kafka/kafka-install.html) * [Kafka - Getting Started](https://www.logicbig.com/tutorials/misc/kafka/getting-started.html) * [Spring Boot - Different Ways To Pass Application Properties](https://www.logicbig.com/tutorials/spring-framework/spring-boot/different-ways-to-pass-application-properties.html) * [Reactor - Transforming into Publishers and Delaying any Error with flatMapDelayError() and flatMapSequentialDelayError()](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/flat-map-delay-error.html) * [Reactor - Transforming into Publisher and Maintaining the source order with flatMapSequential()](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/flat-map-sequential.html) * [Reactor - Transforming into Publishers and then flattening by using flatMap operation](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/flat-map.html) * [Reactor - Using transform Operation](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/transform-operation.html) * [Reactor - Mapping items](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/map-operation.html) * [Reactor - Getting Started](https://www.logicbig.com/tutorials/misc/reactive-programming/reactor/getting-started.html) * [Java 13 - Text Blocks (JEP 355)](https://www.logicbig.com/tutorials/core-java-tutorial/java-13-changes/text-block-examples.html) * [Spring Cloud - Hystrix CircuitBreaker, Thread Local Context Propagation](https://www.logicbig.com/tutorials/spring-framework/spring-cloud/hystrix-thread-local-context-propagation.html) * [Spring Cloud - Circuit Breaker Hystrix Event Listener](https://www.logicbig.com/tutorials/spring-framework/spring-cloud/circuit-breaker-hystrix-event-listener.html) |
| |  |  | | --- | --- | | ui-buttonui-button Thread Starvation Fairness Example | [Select All](javascript:selectAllCode(1);) [Download](https://www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/thread-starvation/thread-starvation-fairness-example.zip) |  * java-thread-starvation   + src     - main       * java         + com   logicbig  example  **FairnessDemo.java**  **StarvationDemo.java**   * + **pom.xml**   package com.logicbig.example;  import javax.swing.\*;  import java.awt.\*;  public class StarvationDemo {  private static Object sharedObj = new Object();  public static void main (String[] args) {  JFrame frame = createFrame();  frame.setLayout(new FlowLayout(FlowLayout.LEFT));  for (int i = 0; i < 5; i++) {  ProgressThread progressThread = new ProgressThread();  frame.add(progressThread.getProgressComponent());  progressThread.start();  }  frame.setLocationRelativeTo(null);  frame.setVisible(true);  }  private static JFrame createFrame () {  JFrame frame = new JFrame("Starvation Demo");  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  frame.setSize(new Dimension(300, 200));  return frame;  }  private static class ProgressThread extends Thread {  JProgressBar progressBar;  ProgressThread () {  progressBar = new JProgressBar();  progressBar.setString(this.getName());  progressBar.setStringPainted(true);  }  JComponent getProgressComponent () {  return progressBar;  }  @Override  public void run () {  int c = 0;  while (true) {  synchronized (sharedObj) {  if (c == 100) {  c = 0;  }  progressBar.setValue(++c);  try {  //sleep the thread to simulate long running task  Thread.sleep(100);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  } | |
|  | |

### Thread Starvation in Java Multi-Threading

In any multi-threaded application, where you have multiple [threads](https://www.netjstech.com/2015/06/lifecycle-of-thread-thread-states-in-java-multithreading.html) vying for the access over the shared resources you may come across a situation where a thread (or a bunch of threads) is unable to gain regular access to shared resources and is unable to make progress. This situation is known as **thread starvation**.

Thread starvation in multi-threading may happen because other “greedy” threads are gaining the lock and access to the shared resource, resulting in a thread (or a bunch of threads) getting starved of access to shared resources and CPU time.

### Thread starvation in Java

Some of the reasons why thread starvation may happen in Java-

* If there is an object providing a [synchronized method](https://www.netjstech.com/2015/06/synchronization-in-java-multithreading-synchronizing-thread.html%20target=) that requires a lots of processing time. If there is a thread invoking this method quite frequently, other threads that also need to access the synchronized method will be blocked.
* If there are some higher [priority threads](https://www.netjstech.com/2015/06/thread-priorities-java-multithreading.html) those threads will get executed first rather than the threads having lower priority.
* If you are using [wait-notify signalling](https://www.netjstech.com/2015/07/inter-thread-communiction-wait-notify-java-multi-thread.html), then theoretically a thread that is waiting to get access to a shared resource may wait indefinitely if always some other thread is notified and get access to the shared resource.

### Thread starvation scenario - Java Example

Let’s see an example in Java where thread starvation scenario is shown.

Here we have a class called **ThreadStarveDemo** which has a synchronized method **displayValues()**. Three [threads are created](https://www.netjstech.com/2015/06/creating-thread-in-java.html) which will share an object of class **ThreadStarveDemo**. There are two classes **Display** and **Display1**.

Two thread objects are created of type Display and one thread object of type Display1 is created. While the two thread objects of type Display are given the maximum priority the thread object of type Display1 is given the minimum priority.

In displayValues() method, [sleep method](https://www.netjstech.com/2015/07/difference-between-sleep-and-wait-java-threading.html) of thread class is used to simulate some delay (processing time) and it is called thrice from maximum priority threads to have a scenario that the method is invoked frequently from thread.

class Display implements Runnable{

private ThreadStarveDemo td;

Display(ThreadStarveDemo td){

this.td = td;

}

@Override

public void run() {

td.displayValues();

System.out.println("Calling again");

td.displayValues();

System.out.println("Calling again");

td.displayValues();

//System.out.println("Calling again");

}

}

/\*\*

\*

\*/

class Display1 implements Runnable{

private ThreadStarveDemo td;

Display1(ThreadStarveDemo td){

this.td = td;

}

@Override

public void run() {

try {

// introducing some delay

Thread.sleep(5000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

td.displayValues();

//System.out.println("Calling again");

}

}

public class ThreadStarveDemo {

synchronized void displayValues(){

System.out.println("In ThreadStarveDemo For thread " +

Thread.currentThread().getName());

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("For thread " + Thread.currentThread().getName());

for(int i = 0; i < 3; i++){

System.out.println("Value of i " + i);

}

}

public static void main(String[] args) {

ThreadStarveDemo td1 = new ThreadStarveDemo();

// Creating 3 threads

Thread thread0 = new Thread(new Display1(td1));

Thread thread1 = new Thread(new Display(td1));

Thread thread2 = new Thread(new Display(td1));

// Setting priorities

thread1.setPriority(Thread.MAX\_PRIORITY);

thread2.setPriority(Thread.MAX\_PRIORITY);

thread0.setPriority(Thread.MIN\_PRIORITY);

thread0.start();

thread1.start();

thread2.start();

}

}

**Output**

In ThreadStarveDemo For thread Thread-2

For thread Thread-2

Value of i 0

Value of i 1

Value of i 2

In ThreadStarveDemo For thread Thread-1

Calling again

For thread Thread-1

Value of i 0

Value of i 1

Value of i 2

Calling again

In ThreadStarveDemo For thread Thread-2

For thread Thread-2

Value of i 0

Value of i 1

Value of i 2

Calling again

In ThreadStarveDemo For thread Thread-1

For thread Thread-1

Value of i 0

Value of i 1

Value of i 2

Calling again

In ThreadStarveDemo For thread Thread-2

For thread Thread-2

Value of i 0

Value of i 1

Value of i 2

In ThreadStarveDemo For thread Thread-0

For thread Thread-0

Value of i 0

Value of i 1

Value of i 2

In ThreadStarveDemo For thread Thread-1

For thread Thread-1

Value of i 0

Value of i 1

Value of i 2

Here it can be seen that Thread-0 has to invoke displayValues() method only once but that invocation is delayed by the threads of maximum priority.

Think of same scenario in a multi-threaded application having lots of threads and a synchronized method called from various points in the application. There is a very good chance that thread starvation may happen.

That's all for this topic **Thread Starvation in Java Multi-Threading**. If you have any doubt or any suggestions to make please drop a comment. Thanks!

[>>>Return to Java Advanced Tutorial Page](https://www.netjstech.com/2015/05/java-advanced-topics.html) | [Return to Java Multi-Threading Tutorial Page>>>](https://www.netjstech.com/p/java-multi.html)