INF214 — Cheat Sheet / Glossary

At-Most-Once Property

An attribute of an assignment statement `x = e` in which **either:**

- `x` is not read by another process and `e` contains at most one reference to a variable changed by another process, **or**
- `x` is not written by another process and `e` contains no references to variables changed by other processes.

Such an assignment statement will appear to execute as an atomic action.

Atomic Action

A sequence of one of more statement that appears to execute as a single, indivisible action.

Fine-grained atomic action

The one that can be implemented directly by a single machine instruction.

Coarse-grained atomic action

The one that is implemented using critical section protocols.

Barrier

A synchronization point that all processes must reach before any are allowed to proceed.

Busy Waiting

An implementation of synchronization in which a process repeatedly executes a loop waiting for a Boolean condition `B` to be true. This is often programmed as `while (!B) skip;`. When a process is busy waiting, it is also said to be **spinning.**

Condition Synchronization

A type of synchronization that involves delaying a process until some Boolean condition `B` is true. This is implemented by having one process wait for an event that is signaled by another process.

Conditional Atomic Action

An atomic action that must delay until the state satisfied some Boolean condition `B`. This is programmed as ⟨`await (B) S;`⟩.

Context Switch

The act of switching a processor from executing one process to executing another. This is called a context switch because the state of each process is called its context. A context switch is performed in a kernel by a routine that is called a dispatcher or scheduler.

Critical Section

A sequence of statements that must be executed with mutual exclusion with respect to critical sections in other processes that reference the same shared variables. Critical section protocols are used to implement coarse-grained atomic actions.

Deadlock

A state in which two or more processes are waiting for each other, in a so-called deadly embrace.

Fairness

An attribute of a scheduler or algorithm that guarantees that every delayed process gets a chance to proceed.
► See also: **Scheduling Policy.**

History

The sequence of states, or actions, resulting from one executing of a program. Sometimes a history is also called a **trace.**

Independent Statements

Two statements in different processes that do not write into the same variables and that do not read variables written by the other. Independent statements will not interfere with each other if they are executed in parallel.

Interference

The result of two processes reading and writing shared variables in an unpredictable order and hence with unpredictable results.
► See also: **Non-interference.**

(Kernel)

A collection of data structures and primitive operations—uninterruptible procedures—that manages processes, schedules them on processors, and implements high-level communication and synchronization operations such as semaphores or message passing.

Livelock

A situation in which a process is spinning while waiting for a condition that will never become true. Livelock is the busy-waiting analog of deadlock.

Liveness Property

A property of a program that asserts that something good will eventually happen—namely, that the program eventually reaches a good state. Termination and eventual entry into a critical section are examples of liveness properties.

Lock

A variable that is used to protect a critical section. A lock is **set** when some process is executing in a critical section; otherwise it is **clear.**

Mutual exclusion

A type of synchronization that ensures that statements in different processes cannot execute at the same time.
► See also: **Critical Section.**

Non-interference

A relation between an atomic action `a` in one process and a critical assertion `C` in another process. Execution of `a` does not interfere with `C` if it leaves `C` true, assuming that `C` is already true.

Partial Correctness

A property of a program that computes the desired result, assuming the program terminates.

Postcondition

An assertion that is true when statement `S` finishes execution.

Precondition

An assertion that is true when statement `S` starts execution.

Race Condition

A situation in a shared-variable concurrent program in which one process writes a variable that a second process reads, but the first process continues execution—namely, races ahead—and changes the variable again before the second process sees the result of the first change. This usually leads to an incorrectly synchronized program.

Safety Property

A property of a program that asserts that nothing bad will ever happen—namely, that the program never enters a bad state. Partial correctness, mutual exclusion, and absence of deadlock are examples of safety properties.

Scheduling Policy

A policy that determines which action gets to execute next—namely, the order in which processes execute.

Unconditionally fair scheduling policy

If unconditional atomic actions eventually get to execute.

Weakly fair scheduling policy

If conditional atomic actions eventually get to execute if the delay condition becomes true and remains true.

Strongly fair scheduling policy

If conditional atomic actions eventually get to execute if the delay condition is infinitely often true.

Spin Lock

A Boolean variable that is used in conjunction with busy waiting to protect a critical section. A process that wants to enter a critical section spins until the lock is clear.

State of a Program

The value of every program variable at a point in time.

Synchronization

An interaction between processes that controls the order in which the processes execute.
► See also: **Mutual Exclusion;** ► **Condition Synchronization.**

Total Correctness

A property of a program that computes the desired result and terminates.

Triple

A programming logic formula having the form {`P`} `S` {`Q`}, where `P` and `Q` are predicates and `S` is a statement list. If execution of `S` starts in a state satisfying `P` and if `S` terminates, then the final state will satisfy `Q`.

Unconditional Atomic Action

An atomic action that does not have a delay condition. This is programmed as ⟨`S;`⟩ and might be implemented as a single machine instruction.