# Task 2.1

1. **Static scheduling**. Workload is divided into chunks depending on the iteration loop and is distributed among the process in a round-robin fashion. It’s best used if the workload can be divided evenly among the threads

**If each iteration in a loop can be done in the same time. (no load-imbalance)**

**Dynamic** **scheduling**. A chunk-sized workload is given immediately to a thread that finishes first. It’s best used if the workload cannot be divided unevenly.

**When a loop has load-imbalance (load-imbalance is very big, e.g. Fibonacci)**

**Guided scheduling.** Similar to Dynamic Scheduling, Guided scheduling start by dividing the workload into small chunks, which later becomes bigger after each iteration/finished thread

**Similar to Dynamic, but the load-imbalance is not very big. (need to check between dynamic and guided to determine the best scheduling to use)**

1. 2 main disadvantages:

* Computation-wise, more expensive than static scheduling (synchronization chunks, overhead)
* Higher computation overhead (more cache misses, hinder cache locality)

1. **Mutual Exclusion** ensures that only 1 process can access a part of a shared memory. Others have to wait until the process is out of the critical region

Mutual Exclusion in OpenMP can be applied by using #pragma omp critical

**Event Synchronization** uses signal that can be sent across multiple processes/threads (e.g. Master *directives*, barriers, ordered sections, etc.)

* + **Barrier** holds all processes until they are in the same step (barred), from which then the processes can continue executing (#pragma omp barrier)
  + **Ordered sections** define a portion of a code inside a paralleled code that can only be executed serially (sequential order) ((#pragma omp ordered)
  + **Single** -> the code enveloped in a single directive will be executed by a random thread chosen by the master.
  + **Atomic.** Difference with *critical* is the number of set of instructions that can be included inside the directive.
  + **Master** directives
  + **Nowait** -> overrides implicit/explicit barrier
  + **Reduction**

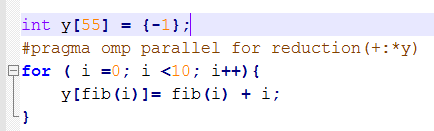
# Task 2.2

Private:

Shared:

# Task 2.3

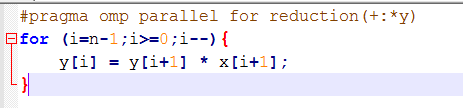
Data race occurs when accessing fib(0) and fib(1)



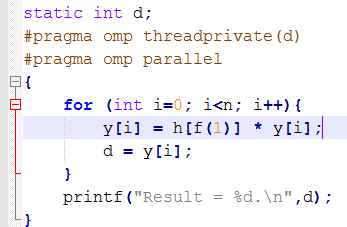
Removes loop 0 – 1. Loop starts with 2. Define **y[0] = 0** outside parallel region

# Task 2.4

1. **Dependence**: Each process depends on the result of the next process.



1. **Dependence**: Each process accessing the same variable



1. **Dependence**: variable depends on variable . Calculation in Variable will affect variable on the other process.

