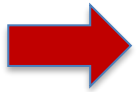
# `std::atomic_flag`

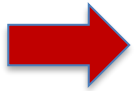
The `std::atomic_flag` has a very simple interface:

- Is the only lock-free data structure.



All other atomics for integral types, pointers, and user-defined atomics can internally use a lock.

- Is the building block for higher abstractions.



Spinlock

# `std::atomic_flag` Interface

| Member Function | Description |
|---|---|
| `atomicFlag.clear()` | Clears the atomic flag. |
| `atomicFlag.test_and_set()` | Sets the atomic flag and returns the old value. |
| `atomicFlag.test()` (C++20) | Returns the value of the flag. |
| `atomFlag.notify_one()` (C++20) | Notifies one thread waiting on the atomic flag. |
| `atomFlag.notify_all()` (C++20) | Notifies all threads waiting on the atomic flag. |
| `atomFlag.wait(val)` (C++20) | Waits for a notification and blocks as long as `atomFlag == val` holds. |

- The default constructor initializes the value.

# std::atomic_flag

## Spinlock

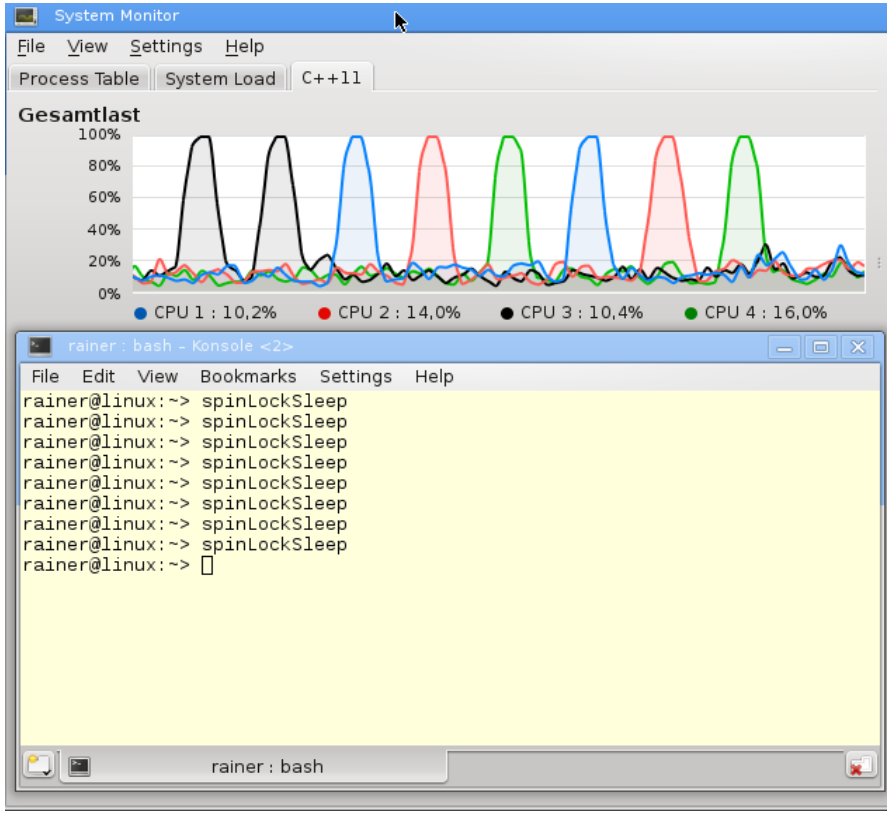
```
class Spinlock{
 std::atomic_flag  flag;
public:
  Spinlock():flag(ATOMIC_FLAG_INIT){}

  void lock(){
    while(flag.test_and_set());
  }

  void unlock(){
    flag.clear();
  }
};
```

## Lock a resource

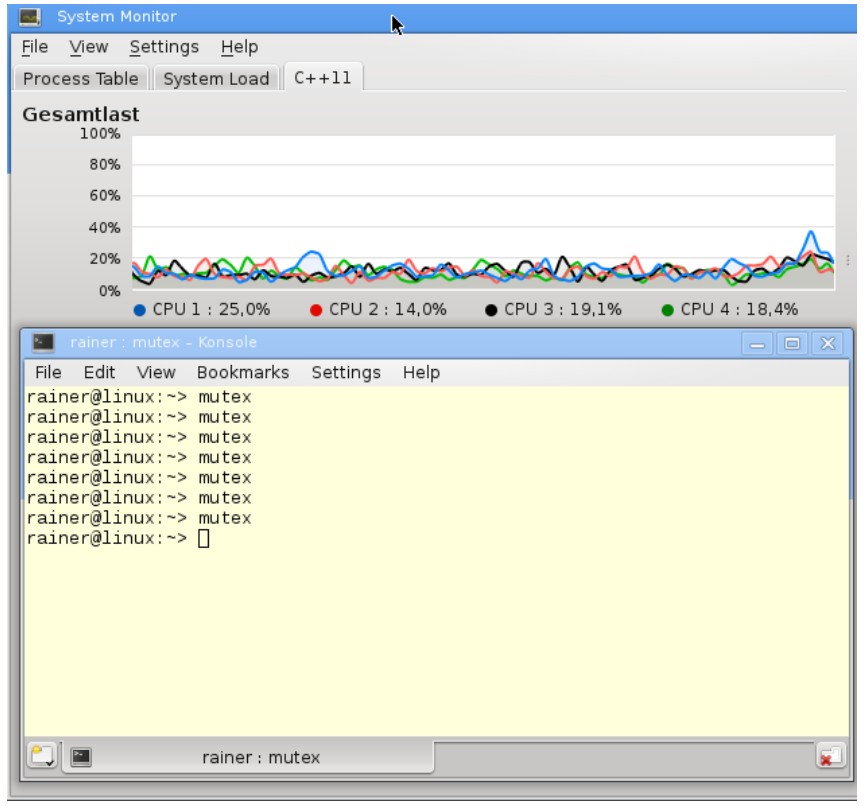
```
Spinlock spin;
// Mutex spin;
void workOnResource(){
  spin.lock();
  sleep_for(seconds(2));
  spin.unlock();
}
int main({
  thread t(workOnResource);
  thread t2(workOnResource);
  t.join();
  t2.join();
}
```

# std::atomic_flag

## Spinlock




## Mutex




spinLock.cpp