Introduction à la programmation concurrente

Section critique: Où et quand?

Yann Thoma, Fiorenzo Gamba

Février 2019

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Introduction

Introduction

- Quand est-il nécessaire de garantir l'exclusion mutuelle sur l'accès à une variable globale?
- Quelques exemples choisis
  - **Nous ne nous intéressons pas à la séquentialité des instructions**, mais
  - A ne pas corrompre des données ou le fonctionnement

Exemples

Exemple (1)

```
static int var;
```

```
void TacheA()
{
    int a;
    ...

    var=a;

    ...
}
```

```
void TacheB()
{
    int b;
    ...

    var=b;

    ...
}
```

Exemple (1)

```
static int var;
```

```
void TacheA()
{
    int a;
    ...

    var=a;

    ...
}
```

```
void TacheB()
{
    int b;
    ...

    var=b;

    ...
}
```

Pas de souci: opération atomique

Exemple (2)

```
static int var;
```

```
void taskA()
{
    int a;
    ...

    var=a;

    ...
}
```

```
void taskB()
{
    int b;
    ...

    b=var;

    ...
}
```

Exemple (3)

```
typedef struct {
    int a;
    int b;
} struct_t;
static struct_t var;
```

```
void TacheA()
{
    int a;
    int b;
    ...

    var.a=a;
    var.b=b;

    ...
}
```

```
void TacheB()
{
    int a;
    int b;
    ...

    var.a=a;
    var.b=b;

    ...
}
```

Exemple (3)

```
typedef struct {
    int a;
    int b;
} struct_t;
static struct_t var;
```

```
void TacheA()
{
    int a;
    int b;
    ...
    DebutSC();
    var.a=a;
    var.b=b;
    FinSC();
    ...
}
```

```
void TacheB()
{
    int a;
    int b;
    ...
    DebutSC();
    var.a=a;
    var.b=b;
    FinSC();
    ...
}
```

Problème: les assignations pourraient s'entrelacer et donc la structure dans un état non-voulu

Exemple (4)

```
typedef struct {
    int a;
    int b;
} struct_t;
static struct_t var;
```

```
void taskA()
{
    int a;
    int b;
    ...

    var.a=a;
    var.b=b;

    ...
}
```

```
void taskB()
{
    int a;
    int b;
    ...

    a=var.a;
    b=var.b;

    ...
}
```

Exemple (4)

```
typedef struct {
    int a;
    int b;
} struct_t;
static struct_t var;
```

```
void taskA()
{
    int a;
    int b;
    ...
    DebutSC();
    var.a=a;
    var.b=b;
    FinSC();
    ...
}
```

```
void taskB()
{
    int a;
    int b;
    ...
    DebutSC();
    a=var.a;
    b=var.b;
    FinSC();
    ...
}
```

Exemple (5)

```
static int var;
```

```
void TacheA()
{
    int a;
    ...

    var++;

    ...
}
```

```
void TacheB()
{
    int b;
    ...

    var++;

    ...
}
```

Exemple (5)

```
static int var;
```

```
void TacheA()
{
    int a;
    ...
    DebutSC();
    var++;
    FinSC();
    ...
}
```

```
void TacheB()
{
    int b;
    ...
    DebutSC();
    var++;
    FinSC();
    ...
}
```

Problème: l'incrémentation n'est pas une opération atomique

Exemple (6)

```
static int var;
```

```
void taskA()
{
    int a;
    ...

    var++;

    ...
}
```

```
void taskB()
{
    int b;
    ...

    b=var;

    ...
}
```

Exemple (7)

```
static int var = 0;
```

```
void TacheA()
{
    int a;
    ...

    var++;

    ...
}
```

```
void TacheB()
{
    int b;
    ...

    var++;

    ...
}
```

```
void TacheC()
{
    int c;
    ...

    c=var;

    ...
}
```

Exemple (7)

```
static int var = 0;
```

```
void TacheA()
{
    int a;
    ...
    DebutSC();
    var++;
    FinSC();
    ...
}
```

```
void TacheB()
{
    int b;
    ...
    DebutSC();
    var++;
    FinSC();
    ...
}
```

```
void TacheC()
{
    int c;
    ...

    c=var;

    ...
}
```

Il faut protéger les incrémentations mais la lecture depuis la mémoire est atomique ici

Exemple (8)

```
static int var;
```

```
void taskA()
{
    int a;
    ...

    var++;

    ...
}
```

```
void taskB()
{
    int b;

    b=var;

    if (b>10) {

        var=0;

    }
    else {

        var++;

    }
```

```
void taskC()
{
    int c;
    ...

    c=var;

    ...
}
```

Exemple (8)

```
static int var;
```

```
void taskA()
{
    int a;
    ...
    DebutSC();
    var++;
    FinSC();
    ...
}
```

```
void taskB()
{
    int b;
    DebutSC();
    b=var;

    if (b>10) {

        var=0;

    }
    else {

        var++;

    }
```

```
void taskC()
{
    int c;
    ...

    c=var;

    ...
}
```

Exemple (9)

```
static int var;
```

```
void TacheA()
{
    int a;
    ...

    var++;

    ...
}
```

```
void TacheB()
{
    int b;
    ...

    var=b;

    ...
}
```

Exemple (9)

```
static int var;
```

```
void TacheA()
{
    int a;
    ...
    DebutSC();
    var++;
    FinSC();
    ...
}
```

```
void TacheB()
{
    int b;
    ...
    DebutSC();
    var=b;
    FinSC();
    ...
}
```

En assignation, il faut protéger car l'assignation en mémoire pourrait être écrasée par l'incrémentation

Exemple (10)

```
static float var;
```

```
void taskA()
{
    float a;
    ...

    var=a;

    ...
}
```

```
void taskB()
{
    float b;
    ...

    var=b;

    ...
}
```

Exemple (10)

```
static float var;
```

```
void taskA()
{
    float a;
    ...

    var=a;

    ...
}
```

```
void taskB()
{
    float b;
    ...

    var=b;

    ...
}
```

Dépend de l'architecture

Exemple (11)

```
static double var;
```

```
void TacheA()
{
    double a;
    ...

    var=a;

    ...
}
```

```
void TacheB()
{
    double b;
    ...

    var=b;

    ...
}
```

Exemple (11)

```
static double var;
```

```
void TacheA()
{
    double a;
    ...
    DebutSC();
    var=a;
    FinSC();
    ...
}
```

```
void TacheB()
{
    double b;
    ...
    DebutSC();
    var=b;
    FinSC();
    ...
}
```

Dépend de l'architecture de la CPU, il faut donc protéger les accès

Exemple (12)

```
std::vector<bool> vec;
```

```
void TacheA()
{
    bool a;
    ...

    vec[0] = a;

    ...
}
```

```
void TacheB()
{
    bool b;
    ...

    vec[0] = b;

    ...
}
```

Exemple (12)

```
std::vector<bool> vec;
```

```
void TacheA()
{
    bool a;
    ...
    DebutSC();
    vec[o] = a;
    FinSC();
    ...
}
```

```
void TacheB()
{
    bool b;
    ...
    DebutSC();
    vec[o] = b;
    FinSC();
    ...
}
```

Problème: `std::vector<bool>` est dépendant de l'implémentation. L'espace mémoire peut être optimisé, ce qui implique qu'une écriture n'est plus forcément atomique (récupération de l'élément, application d'un masque, écriture en mémoire).

Conclusion

- Dans le doute...
- Protégez tous les accès concurrents!