Résumé Système Embarqué II TE04

Principe de l'orienté objet en C.

En C les objets ne sont pas supportés. Pour créer un objet il faut faire une structure dans laquelle on va mettre les attributs et les méthodes de l'objet. Les méthodes devront avoir en paramètres la référence de la structure (objet).

Utilité de la macro *container_of* et son implémentation.

Container_of permet d'avoir la référence sur une classe dérivée afin d'avoir accès à ses attributs et méthodes.

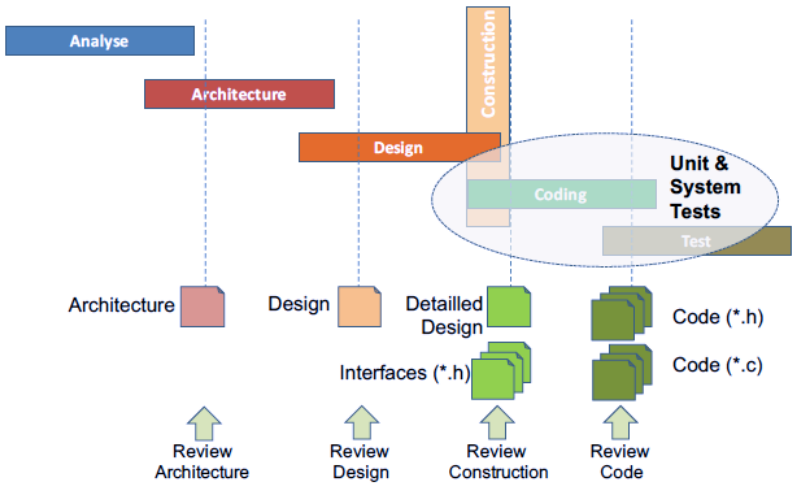
```
#define container_of (ptr, type, member) \
((char*)(((type*)(ptr))–offset_of(type, member)))
```

Méthode pour débugger une application chargée sur une cible à partir d'une machine hôte (2 interfaces).

Pour faire du « remote debuging » il faut utiliser le « gdb » et le « gdb_server ».

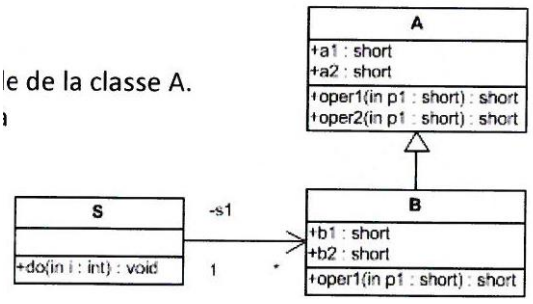
Objectif des revues de construction, ou elles se situent (phase) et quel type de document est examiné.

Les revues de construction se font à la fin de la phase de design et au début de la phase de codage. On fait une révision des .h et de la documentation faite en phase de design afin d'éviter de perdre du temps avec des erreurs en phase de codage.



Déclaration classe A, B et S en C orienté objet, avec la méthode « *oper1* » de la classe B qui surcharge celle de la classe A (a) et la fonction « *oper1* » de la classe B retourne le produit de « *p1 * a2 * b2* » (b).

le de la classe A.



a)

```
Struct A {
short a1 ;
short a2 ;
short (*oper1)(struct A* oref, short p1) ;
short (*oper2)(struct A* oref, short p1) ;
}
Struct B {
short b1 ;
short b2 ;
struct A m_base ;
}
struct S {
struct B* group ;
void (*do)(struct S* oref, int :) ;
}
```

b)

```
short operations (struct A* oref, short p1) {
struct B* B_ref = container_of(oref, struct B, m_base) ;
return (p1* oref -> a2 * B_ref -> b2) ;
}
```

Principe de fonctionnement d'un Makefile.

make est un utilitaire permettant de décrire les dépendances d'une application, de compiler les fichiers sources et de linker les fichiers objets nécessaires à sa génération.

```
EXEC=exec
CC=gcc
CFLAGS=-g -c -Wall -Wextra -O1 -std=c11
LDFLAGS=-lmymathlib -L.

$(EXEC): main.o libmymathlib.a
-> $(CC) main.o $(LDFLAGS) -o $(EXEC)

main.o: main.c mymathlib.h
-> $(CC) $(CFLAGS) -o main.o main.c
```

Makefile pour la génération d'une application composée de 3 fichiers (*my_app.c, file1.c, file2.c*). Le programme exécutable est applé « my_app ». Le compilateur « gcc » est utilisé pour la génération de l'application avec les flags de compilation « -Wall –Wextra -01 –std=c11 ». Le Makefile permet également d'effacer tous les fichiers générés.

```
EXEC = my_app
CC = gcc
CFLAGS = Wall –Wextra -01 –std=c11
SRCS = file1.c file2.c my_app.c
OBJS = $(SRCS :.c=0)

all : $(EXEC)
.c .0
        $((CC)$(CFLAGS) -0 $* .0$<
$(EXEC) : $(OBJS)
        $(CC) $(LDFLAGS) -0 $@ $^
clean : $(EXEC)
rm –f $(EXEC)
rm –f *.0
.PHONY all clean
```

.PHONY -> Indique les cibles factices et force ainsi leurs reconstruction.

Utilitaires :

- **gcc** : Compilateur
- **as** : Assembleur
- **ld** : Linker
- **ar** : Création des bibliothèques
- **strip** : Elimine les symboles d'un code objet

Tests unitaires

Les tests unitaires permettent de valider le bon fonctionnement des différentes composantes d'un logiciel avec un programme logiciel réalisé par le développeur.

- Développement de logiciel à la place d'un protocole de tests fastidieux.
- Les batteries de tests peuvent être automatisées et répétées à volonté.
- Meilleure couverture des tests réalisés.
- Plus grande efficacité / productivité.
- Méthode TDD peut être mise en œuvre.

On utilise le « cgov » pour vérifier que toutes les lignes de code ont été vérifiées.

```
// function to be tested
int max (int i1, int i2) {
if (i1 > i2)
return i1 ;
else
return i2 ;
}

// simple tests of ”max”
void test_max() {
CU_ASSERT(max(0,2) == 2);
CU_ASSERT(max(-1,2) == 2);
CU_ASSERT(max(2,2) == 2);
}
```

```
/**
 * This function returns the base 10 logarithm of x.
 *    - if x is NAN: NAN is returned
 *    - if x is 1: 0 is returned
 *    - if x is negative: NAN is returned
 *    - if x is 0: -HUGE_VAL is returned
 */
double log10 (double x);
```

```
void test_log10() {
        CU_ASSERT (log10(NAN)==NAN) ;
        CU_ASSERT (log10(1)==0) ;
        CU_ASSERT (log10(-5)==NAN) ;
        CU_ASSERT (log10(0)==-Huge_VAL) ;
}
```

En-tête de code source, ici pour le fichier « fibonacci.c » qui calcule et affiche la suite de Fibonacci.

Une en-tête sert à décrire brièvement à quoi sert le code, qui l'a codé, quand, etc… Pour une méthode on indiquera les paramètres et la valeur de retour ainsi qu'une brève description.

```
/**
* @file fibonacci.c
* @brief permet de calculer la suite de Fibonacci
* @date 14.06.2016
* @author Lambert Michaël
*/
```

Gestionnaires de code source (SVN & Git).

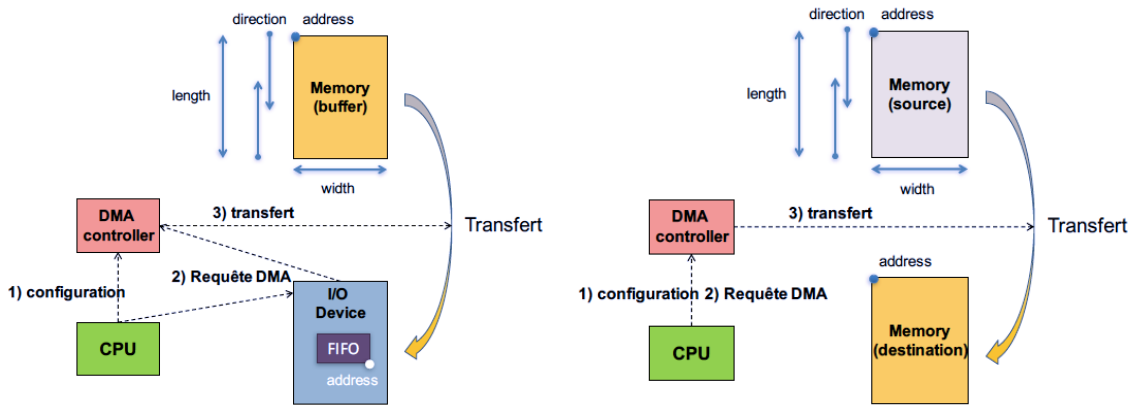
Ces utilitaires permettent de gérer très « simplement » le code source, les modifications apportées au code, des versions différentes (branches), etc…

SVN (Centralisé) : Chaque développeur obtient tout ou partie des sources d'une base de données placée sur un serveur centralisé, qu'il peut modifier et resynchroniser avec le serveur.

Git (Décentralisé) : Chaque développeur dispose de l'entier du dépôt qu'il peut synchroniser avec d'autres développeurs ou avec un dépôt de référence placé sur une machine publique.

DMA (Direct Memory Access)

Le contrôleur DMA permet de décharger le CPU, il prend en charge le transfert de données entre un périphérique d'entrées/sorties et la mémoire principale. Le CPU est déchargé de toutes les tâches hormis la configuration du contrôleur DMA.



Mémoire cache

La mémoire cache permet d'augmenter les performances globales d'un µP.

Localité spatiale

Le principe de localité spatiale suppose que si un emplacement mémoire est référencé à un instant donné, il est alors fort probable qu'un emplacement à proximité soit également référencé peu après.

```
uint16_t in[128];
uint16_t out[128];
for (int i=0 ; i<128 ; i++) {
out[i] = in[i];
}
```

Localité temporelle

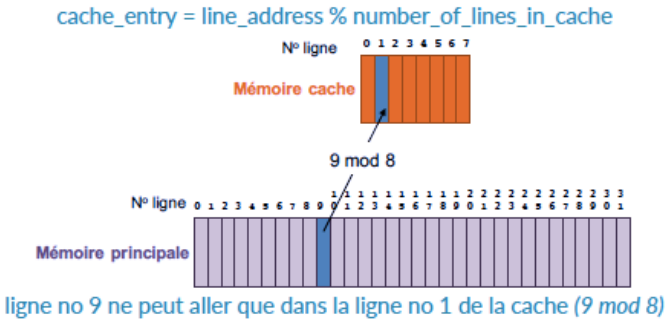
Le principe de localité temporelle suppose que si à un instant donné un emplacement mémoire est référencé, il est alors fort probable que ce même emplacement soit à nouveau référencé peu après.

```
uint8_t msg[128];
uint32_t sum=0 ;
for (int i=0 ; i<128 ; i++) {
sum += msg[i];
}
```

3 types de mémoire cache

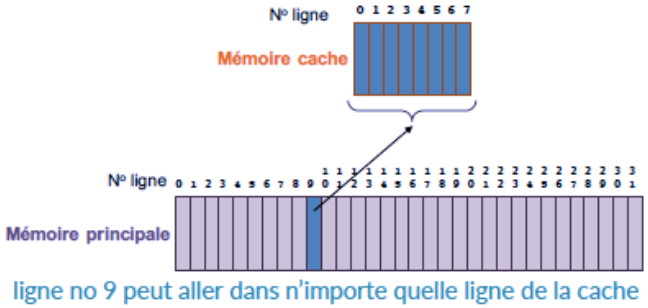
Cache à accès direct (Simple et bon marché mais mauvaise performances)

Une ligne de la mémoire principale ne peut être placée qu'à une seule place dans la mémoire cache, le choix de la place est calculé en prenant le modulo de l'adresse de la ligne en mémoire principale avec le nombre de lignes de la mémoire cache.



Cache complétement associative (Excellente performances mais coûts très élevés)

Une ligne de la mémoire principale peut être placée à n'importe quel endroit dans la mémoire cache.

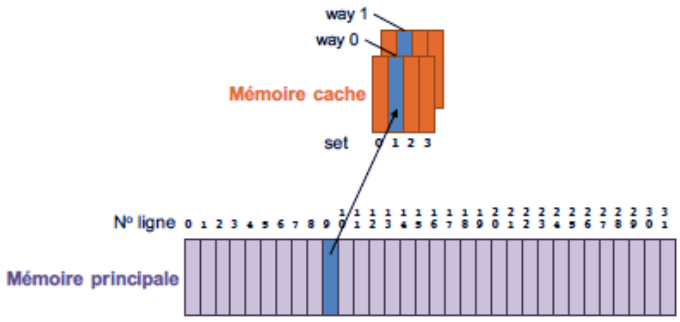


ligne no 9 peut aller dans n'importe quelle ligne de la cache

Cache à N-voies associatives

Ce type de mémoire cache est un compromis des deux précédents :

- La mémoire cache est divisée en ensembles (set)
- Chaque ensemble contient plusieurs lignes, appelées voies (way)



ligne no 9 peut aller soit dans la voie 0 ou 1, mais que dans le set no 1 (ligne no 1)

L'emplacement dans la mémoire cache est calculé en 2 étapes :

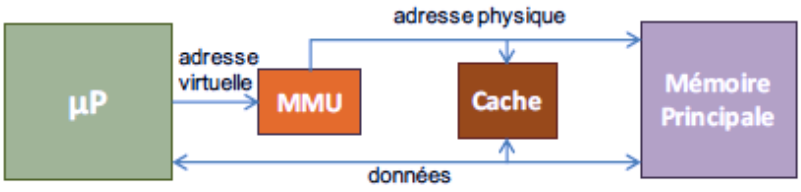
- Accès direct pour le choix de l'ensemble (set)
- Complétement associatif pour le choix de la voie (way)

MMU (Memory Management Unit)

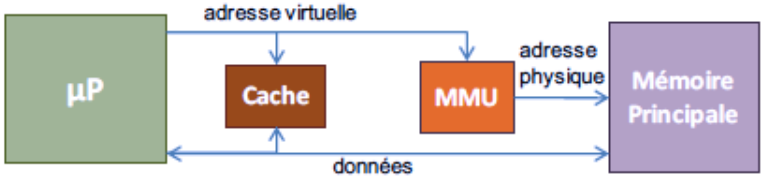
La MMU est chargée de traduire les adresses virtuelles en adresses physiques.

- **Protection des accès à la mémoire physique** : Elle procure une isolation hermétique du processus, en garantissant qu'il ne puisse pas sortir de son espace virtuel et accéder aux données d'autres processus
- **Gestion de la mémoire cache** : Elle décide quelles données peuvent être stockées dans la mémoire cache, car toutes les données traitées par le µP ne peuvent pas l'être.

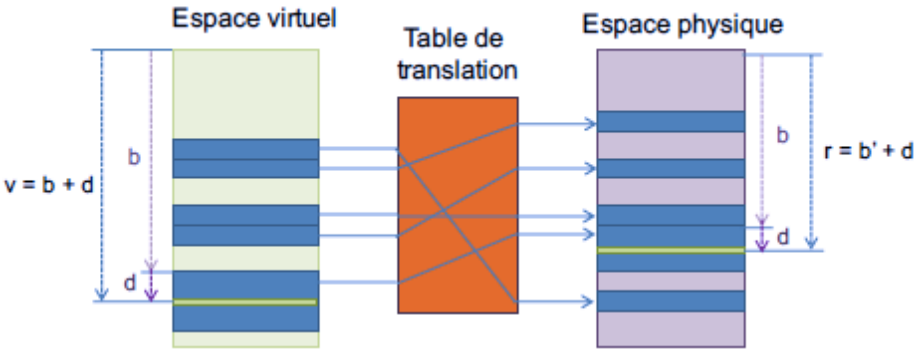
Cache physique :



Cache virtuelle :

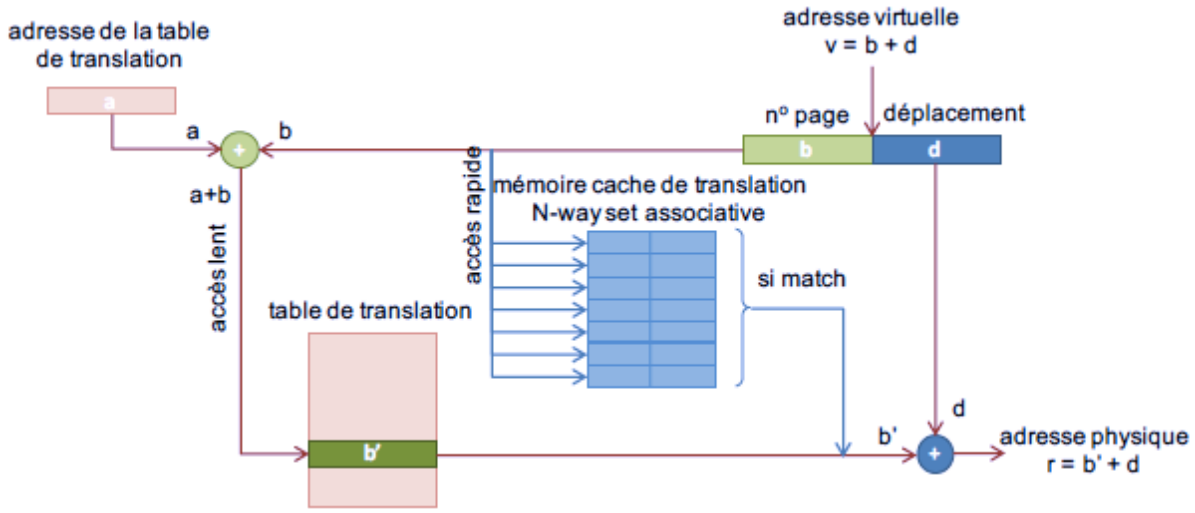


Pour convertir une adresse virtuelle (v = b + d) en une adresse physique (r = b' + d), la MMU utilise une table de translation qui se site dans la mémoire principale.



TLB (Translation Lookaside Buffer)

Afin de diminuer les temps d'accès sur la table de traduction située dans la mémoire principale, une partie de celle-ci est copiée dans une mémoire à l'intérieur de la MMU (du µP).



Mécanisme permettant à une méthode d'une classe dérivée d'obtenir la référence sur son objet (l'objet dérivé) à partir de la référence sur l'objet de la classe de base

Pour obtenir la référence de la classe dérivée, à partie de la référence de la classe de base, il faudra soustraire l'espace mémoire occupé par les éléments déclarés avant de la classe dérivée.

Documentation

Documentation pour utilisateur (user doc)

- Utilisation et intégration des composants et librairies -> doxygen
- Description des interfaces (API) -> doxygen
- Notes de révision (revision notes) -> ascii text file

Documentation pour le vérificateur

- Notes de révision sous une forme très détaillée -> ascii text file

Documentation pour le développeur

- Documentation de l'utilisateur
- Documents de spécifications -> office doc
- Description des modifications apportées au code -> repository
- Commentaires dans le code