Systèmes Embarqués 1 & 2: Travail écrit no 4.

Classe : T-2/I-2

Nom : **Prénom :** **Date : 08.06.2015**

Problème n° 1 (programmation orienté-objet)

1. Décrivez succinctement le principe d'orienté-objet en langage C.

méthodes — attributs

objet → structure

les objets sont stockés dans des structures, où les attributs en représentent l'état.

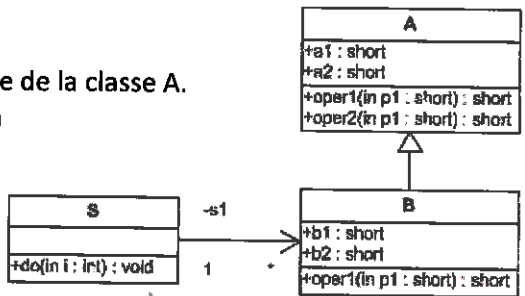
+ passer la référence de l'objet aux méthodes

2. Décrivez succinctement l'utilité de la macro `container_of` et donnez son implémentation.

Permet de récupérer les héritantes à partir de la struct mère.

implémentation: elle utilise la macro d'offset de, mais comment?

3. Pour le diagramme de classes ci-contre :
   a. Déclarez les classes A, B et S en langage C orienté-objet.
      Remarque : la méthode «`oper1`» de la classe B surcharge celle de la classe A.
   b. Implémentez la fonction «`oper1`» de la classe B de manière à ce qu'elle retourne le produit de « `p1 * a2 * b2`»



a)
```
struct A {
    short a1;
    short a2;
    short (*oper1)(struct A*, short);
    short (*oper2)(struct A*, short);
};
```

```
struct S {
    struct B** s1;
    void (*do)(struct S*, int);
};
```

```
struct B {
    short b1;
    short b2;
    struct A* b_base;
    short (*oper1)(struct B*, short);
};
```

b)
```
public short oper1(struct A* aref, short p1) {
    struct B* bref =
        container_of(aref, B, b_base);
    return p1 * aref->a1 * bref->b2;
}
```

Systèmes Embarqués 1 & 2: Travail écrit no 4.

Problème n° 2 (Toolchain)

1. Expliquez à l'aide d'un exemple le principe de fonctionnement d'un Makefile.

Le makefile permet de build automatiquement de fichiers .c, des libraries, ... en fonction d'une configuration définie

Exemple: - build que une partie de l'app car tout n'est pas implémenté.
↳ limiter la variable SRCS à certains .c files.

2. Concevez un Makefile pour la génération d'une application composée de 3 fichiers (`my_app.c`, `file1.c`, `file2.c`). Le programme exécutable sera appelé «`my_app`». Le compilateur «`gcc`» sera utilisé pour la génération de l'application avec les flags de compilation «`-Wall -Wextra -O1 -std=c11`». Le Makefile devra également permettre d'effacer tous les fichiers générés.
Il est impératif d'utiliser des variables pour spécifier les flags de compilation et les fichiers sources.

Makefile:

```
EXEC = my_app
CC = gcc
LDFLAGS = -Wall -Wextra -O1 -std=c11
CFLAGS = $(LDFLAGS) -g -c
SRCS = my_app.c file1.c file2.c
OBJS = $(SRCS:.o=.c)
all: $(EXEC)
%.o : %.c
    $(CC) $(CFLAGS) -o $*.o $<
$(EXEC): $(OBJS)
    $(CC) $(CFLAGS) -o $@ $^

clean:
    rm -f $(EXEC)
    rm -f *.o
.PHONY
clean all
```

3. Décrivez succinctement la méthode à mettre en œuvre pour débugger une application chargée sur une cible à partir d'une machine hôte. Citez une ou deux interfaces permettant de connecter la machine hôte à la cible pour de telles opérations.

gdb → outils GNU de debug

interfaces: - ~~OOD~~ eclipse, ...

Systèmes Embarqués 1 & 2: Travail écrit no 4.

Problème n° 3 (Vérification)

1. Décrivez l'objectif des revues de construction, indiquez où il se situe (quelle phase) dans le processus de développement logiciel et quels types de documents sont examinés

Elles se situent à chaque phases (Analyse, architecture, design, coding, tests).

On examine les .h après le design.
On examine aussi les .h et .c après le coding.

objectif: garder un développement structuré et ne pas coder ça double

2. Décrivez succinctement le principe des tests unitaires et citez une méthode permettant d'en garantir l'efficacité.

Principe: Test automatiquement le retour d'une fonction en fonction de l'input donné.

Efficace si:
- Test valeur OK positive (1; 35; 150)
- Test valeur OK négative (-1; -35; -150)
- Test +MAX_VAL
- Test -MAX_VAL
- Test cas critiques (retours particuliers)
- Test NAN
- Test 0

3. Implémentez un test unitaire permettant de valider/vérifier le bon fonctionnement de la fonction ci-dessous (2 tests positifs et 2 tests négatifs).

```
/**
 * 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);
```

```
CU_ASSERT (log10(NAN) == NAN);
CU_ASSERT (log10(1) == 0);
CU_ASSERT (log10(-1) == NAN);
CU_ASSERT (log10(-35) == NAN);
CU_ASSERT (log10(-MAX_VALUE) == NAN);
CU_ASSERT (log10(0) == -HUGE_VAL);
CU_ASSERT (log10(10) == 1);
CU_ASSERT (log10(100) == 2);
```

```
CU_ASSERT (log10(MAX_VA
== log10(MAX_VAL
```

Systèmes Embarqués 1 & 2: Travail écrit no 4.

10

Problème n° 4 (Documentation & DMA)

1. On constate que les codes sources ont souvent un en-tête sous la forme d'un commentaire. Expliquez à quoi sert cet en-tête et indiquez les informations données par en tel en-tête. Rédigez un en-tête pour le fichier «fibonacci.c» qui calcule et affiche la suite de Fibonacci.

l'en-tête permet de générer la doc doxygen avec des paramètres (auteurs, dat

```
/*  Copyright blablabla
 *  @author Gabriel Magne
 *  @date 23.06.15
 *  @purpose Compute Fibonacci suite
 *
 *
 */
```

3

2. Décrivez succinctement l'utilité d'outils tel que Git ou Subversion

Permet de gérer les versions d'une application, en gardant un historique de qui les a effectuées (et quand).

2

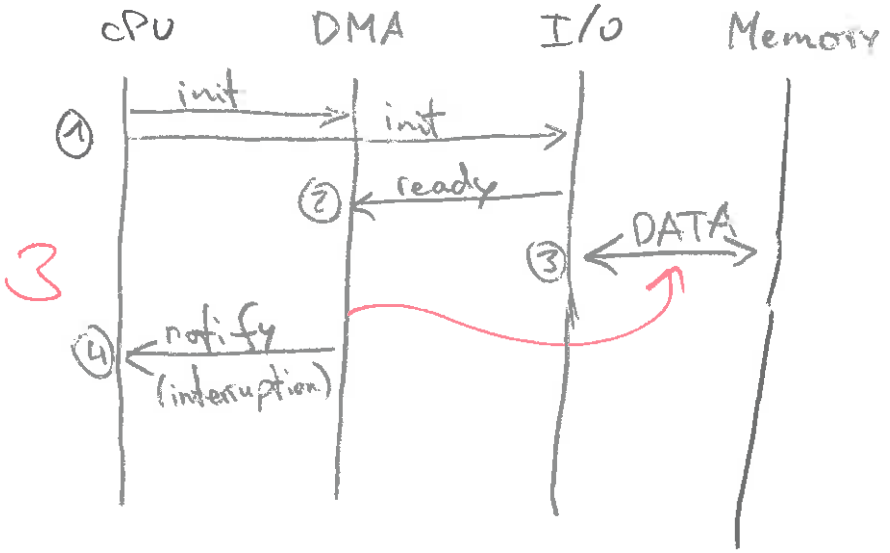
3. Donnez la définition de l'abréviation DMA et décrivez succinctement sa fonction dans un système à µP

Direct Memory Access ✓

2 permet à un/des I/O d'écrire directement dans la mémoire

→ cela économise du tps de calcul au ✓ CPU

4. Expliquez à l'aide d'une figure les phases principales d'un transfert DMA entre un périphérique d'entrée/sortie et la mémoire principale



3

Systèmes Embarqués 1 & 2: Travail écrit no 4.

10

Problème n° 5 (Mémoire cache et MMU)

1. L'utilisation de la mémoire cache s'est popularisée sur les µP modernes. Décrivez succinctement son utilité et indiquez les 3 types d'architecture des mémoires caches.

3 types : Accès direct ✓
Complètement associative
N-K associations ✓

3

Utilité : augmente les perfs globales du CPU ✓ pas

2. Citez les deux principes qui sont à l'origine des mémoires caches et donnez un exemple.

11

temporel ✓ : si on accède à qqch, on a beaucoup de chance d'y ré-accéder directement après.

ex

localité ✓ : si on accède à qqch, on a beaucoup de chance d'accéder à qqch qui le suit directement après.

3. Donnez la définition de l'abréviation MMU et décrivez succinctement sa fonction dans un système à µP

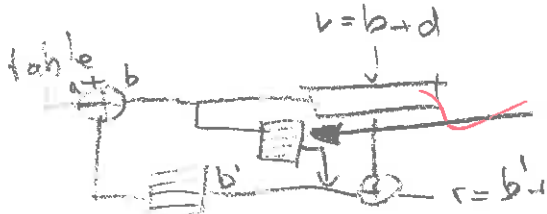
Memory Management Unit ✓

2

Traduit des adresses virtuelles (cache) en adresses physiques (memory) ✓

4. Décrivez succinctement la fonction de la TLB (Translation Lookaside Buffer)

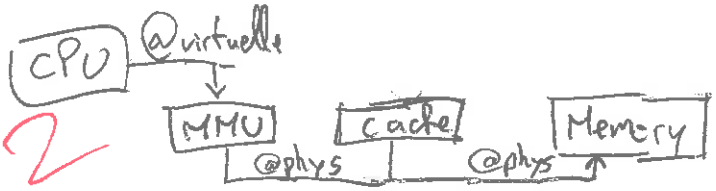
4



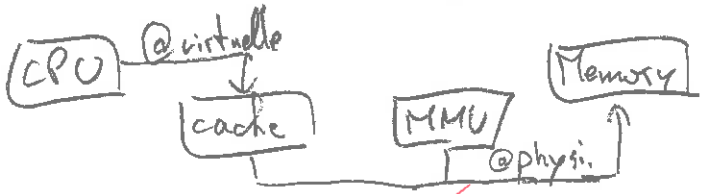
raccourci lors de la traduction car MMU sur le CPU.

5. On parle de cache physique et virtuelle. Expliquez succinctement la différence entre ces 2 types.

2



Physique : ↗ ✓



Virtuelle : ↗ ✓