Microprocesseurs 1 & 2: Travail écrit no 4.

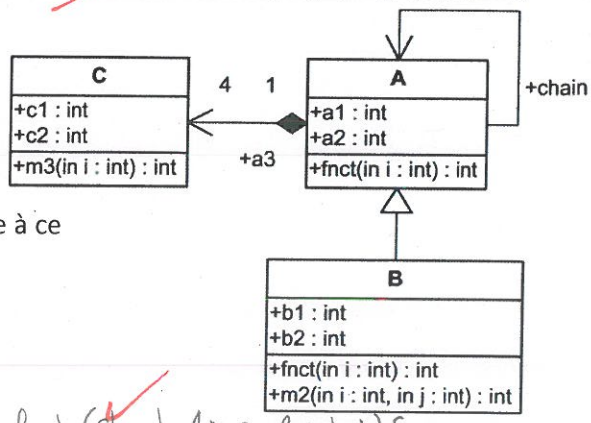
Nom : Thalmann **Prénom :** Cédric 5.2

Classe : T/2
Date : 11.06.2012

7

Problème n° 1 (programmation orienté-objet)

1. Pour le diagramme de classes ci-contre :
   a. Déclarez les classes A, B et C en langage C orienté-objet.
   Remarque : la classe B surcharge la fonction «fnct» de la classe A.
   b. Implémentez la fonction «fnct» de la classe B de manière à ce qu'elle retourne la somme de « i + B::b1 + A::a2 »



a)

```
struct A {
    int a1;
    int a2;
    int (*fnct)(struct A* oref, int i);
    (struct A*)chain;
    (struct C*) a3[4];
}
struct B {
    int b1;
    int b2;
    int (*fnct)(struct B* oref, int i);
    int (*m2)(struct B* oref, int i, int j);
    struct A m_base;
}
struct C {
    int c1;
    int c2;
    int(*m3)(struct C* oref, int i);
}
```

b)

```
int fnct(struct A* oref, int i) {
    B_oref = container_of(oref, struct B, m_base);
    return i + B_oref->b1 + oref->a2;
}
```

2. Implémentez les macros «offset_of» et «container_of» permettant d'obtenir la référence sur l'objet dérivé à partir de la référence sur la classe de base.

container_of(ptr, type, member) = (. . .) - offset_of(type, member);

voir théorie

offset_of(type, member) = ((char*) (&(type -member))

! Je ne sais plus exactement comment ° cela fonctionne !

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

En C, le principe OO n'existe pas.
Nous pouvons néanmoins le faire avec des structures.
Elles permettent de rassembler des attributs et des fonctions dans un même objet (=structure).
Passer référence(adresse) de la struct à toutes les méthodes + Hérit

Microprocesseurs 1 & 2: Travail écrit no 4.

8

Problème n° 2 (Toolchain)

1. Concevez un Makefile pour la génération de l'application « `exec` », laquelle est composée de 3 fichiers (file1.c, file2.c et file3.c). Pour la génération de l'application on utilisera le compileur GNU « `gcc` » avec les flags «`-g -Wall -Wextra -O2 -std=c99 -MD`». Pour rappel, le flag «`-MD`» permet de générer les dépendances. Le Makefile devra également permettre d'effacer les fichiers générés pour une cible donnée. Il est impératif d'utiliser des variables pour spécifier les flags de compilation et les fichiers sources. La génération des codes objets sera fera à l'aide d'une règle.

Makefile :

2

```
EXEC = exec
SRCS = file1.c file2.c file3.c
CC = gcc
CFLAGS = -g -Wall -Wextra -O2 -std=c99 -MD -c
OBJS = $(SRCS:.c=.o)        DEPS = $(OBJS:.o=.d)

.c.o:
    $(CC) $(CFLAGS) $< -o $@
-include $(DEPS)
```

0,5

```
all: $(EXEC)
    $(CC) $(CFLAG) $(LDFLAGS) $^ -o $@
```

1,1

```
clean:
    rm -Rf $(OBJS) $(EXEC) $DEPS

.PHONY clean all
```

2. Indiquez la fonction des 2 de ces 4 utilitaires suivants :
   a. gcov : permet de voir la converture de nos tests sur un code (quelles instructions ne sont pas employées par exemple)
   b. objdump :
2
   c. strip :
   d. gprof : permet de faire du profiling de code

3. Indiquez en une phrase la méthode pour débugger une application fonctionnant sur une cible à partir d'une machine hôte.

Nous pouvons utiliser l'utilitaire gdb. + gdb sur
1 + gdbserver sur la cible

Microprocesseurs 1 & 2: Travail écrit no 4.

8

Problème n° 3 (Vérification)

1. Citez 2 techniques/méthodes permettant de valider des applications logicielles dans les différentes phases de leur développement

2

reviews / tests unitaires & tests système

↓

early revies
revues de construction

2. Décrivez une technique/méthode permettant de garantir qu'un composant logiciel a été correctement et si possible complètement vérifier. Citez un utilitaire de la chaîne d'outils GNU permettant de mettre d'utiliser cette méthode/technique ainsi que la façon de le mettre en œuvre.

2

Pour vérifier un code il faut faire des tests unitaires et utiliser l'utilitaire gcov afin de voir s'ils ont testé toutes les lignes de code.

pour test unitaire en C: CUnit

3. Décrivez succinctement le concept de revues de construction

0.1

A la fin de chaque phase d'un ~~projet~~, il faut faire des revues de constructions.

→ Analyse spec, Architecture spec, Design spec, Code spec, tests spec.

Analyse des API et de la structure du logiciel en début de phase d'implémentation

4. Implémentez un test unitaire permettant de valider/vérifier deux résultats positifs et un résultat négatif pour la fonction « `strrchr ()` » de la librairie standard C (selon description ci-dessous).

```
/** string scanning operation
 *
 * The strrchr() function shall locate the last occurrence of c (converted to a char)
 * in the string pointed to by s. The terminating null byte is considered to be part
 * of the string.
 *
 * @return Upon successful completion, strrchr() shall return a pointer to the byte
 *         or a null pointer if c does not occur in the string.
 */
char *strrchr(const char *s, int c);
```

char* myChar = "bonjour"; 0 1 2 3 4 5 6

CU_ASSERT(strrchr (&myChar , 'r'), ~~*myChar[6]~~); //devrait être juste

CU_ASSERT(strrchr (&myChar, 'c'), ~~null~~); // devrait être juste

3

CU_ASSERT(strrchr (&myChar , 'c'), ~~*myChar[1]~~); //devrait être faut

Microprocesseurs 1 & 2: Travail écrit no 4.

9

Problème n° 4 (Documentation)

1. Citez 4 outils permettant de simplifier le développement de logiciels et d'améliorer sa qualité

2,1

- pour réviosionner le code : SVN, Git, ... ✓
- pour tracer les bugs : bracker ✓
- pour le code : doxygen ✓
- ~~"autre" : wiki, fichier texte (ASCII)~~

2. Citez les 3 niveaux principaux de la documentation du logiciel (*public cible*)

3

- utilisateur : pour les utilisateurs finaux (mode d'emploi, ...)
- vérificateur : doc sur les tests réalisés
- développeur : doc sur code, design, architecture, ...

3. Décrivez succinctement l'utilité d'un SCM (Source Code Management Tool) tel que GIT ou SVN

1,5

pour révisionner le code : gérer les versions, revenir en arrière, ✓
gérer le codage par un team, ... ✓
+ création des branches (maintenance, dev. para.)

4. Indiquez une manière de structurer le logiciel et sa documentation afin de simplifier son développement et sa maintenance

2

Platform — Domain
└ Component
└ Module

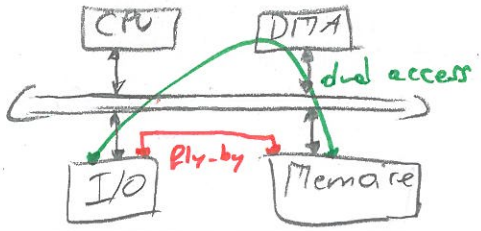
Bibliothèque — Domain
└ Component
└ Module

Produits — Domain
└ Component
└ Module

Microprocesseurs 1 & 2: Travail écrit no 4.

10

Problème n° 5 (Mémoire cache et MMU)

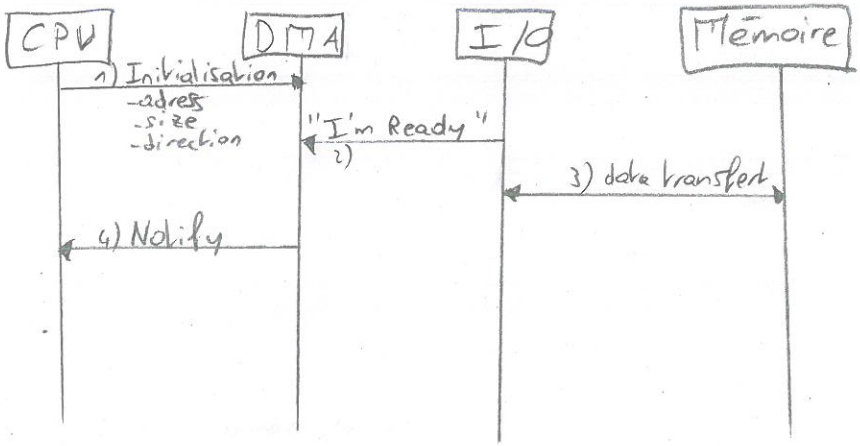


1. Décrivez succinctement le principe d'un DMA.

Le contrôleur DMA sert à décharger le CPU lors de gros ou de fréquents transfert de données entre les entrées-sorties (I/O) et la mémoire.

2

2. Indiquez à l'aide d'un graphique les 4 phases principales d'un transfert DMA



3

3. Décrivez succinctement la fonction de la mémoire cache et citez les deux principes qui sont à son origine.

la mémoire cache (mémoire de petite taille mais très rapide) sert à stocker les données souvent utilisées afin de diminuer le temps d'attente.

2

Principes à son origine:

1) proximité spatiale

2) proximité temporelle

4. Citez deux algorithmes de remplacement de ligne dans la mémoire cache

1) FIFO: la première stockée dans la cache sera la première remplacée lorsqu'elle sera pleine

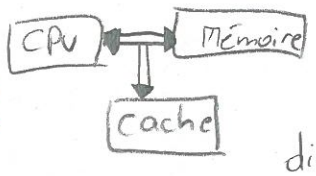
1

2) Aléatoire: de manière aléatoire

principale

5. Décrivez succinctement les deux algorithmes d'écriture des données dans la mémoire ~~cache~~

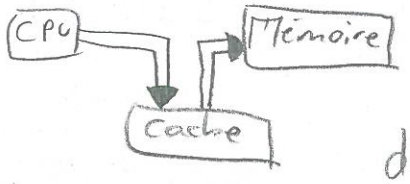
1)



2

directe

2)



différée