Logiciel de Base

Ensimag 1A Apprentissage

Examen — Juin 2012

Consignes :

- Durée : 2h.
- Tous documents autorisés.
- Le barème est donné à titre indicatif.
- On attend des réponses courtes et pertinentes, inutile de recopier le cours.
- Les parties sont indépendantes les unes des autres. La plupart des questions du problème sont également indépendantes. Pensez à lire le sujet en entier avant de commencer à répondre.

Consignes relatives à l'écriture de code C et assembleur Pentium :

- Pour chaque question, une partie des points sera affectée à la clarté du code et au respect des consignes ci-dessous.
- Pour les questions portant sur la traduction d'une fonction C en assembleur, on demande d'indiquer en commentaire chaque ligne du programme C original avant d'écrire les instructions assembleur correspondantes.
- Pour améliorer la lisibilité du code assembleur, il est conseillé d'utiliser des constantes (i.e. déclarations du type x=42) pour les déplacements relatifs à `%ebp` (i.e. paramètres des fonctions et variables locales). Par exemple, si une variable locale s'appelle `var` en langage C, on y fera référence avec `var(%ebp)`.
- Sauf indication contraire dans l'énoncé, on demande de traduire le code C en assembleur de façon systématique, sans chercher à faire la moindre optimisation : en particulier, **on stockera les variables locales dans la pile** (pas dans des registres), comme le fait le compilateur C par défaut.
- On respectera les conventions de gestions des registres Intel vues en cours, c'est à dire :
  - `%eax`, `%ecx` et `%edx` sont des registres scratch ;
  - `%ebx`, `%esi` et `%edi` ne sont pas des registres scratch.

1 Exercices sur le langage d'assemblage et GDB

On considère le programme assembleur suivant :

```
        param_y=8
        param_x=12
swap:
        pushl %ebp
```

```
        movl %esp, %ebp

        movl param_y(%ebp), %eax
        movl (%eax), %esi
        movl param_x(%ebp), %ecx
        movl (%ecx), %edi
within_swap:
        movl %esi, (%ecx)
        movl %edi, (%eax)

        leave
        ret

        .data
x:      .int 42
y:      .int 54
fmt:    .asciz "x=%d, y=%d\n"
        .text
        .globl main
main:
        pushl %ebp
        movl %esp, %ebp

        pushl x
        pushl y
        pushl $fmt
        call printf
        addl $12, %esp

        pushl $x
        pushl $y
before_call_swap:
        call swap
        addl $8, %esp

        pushl x
        pushl y
        pushl $fmt
        call printf
        addl $12, %esp

        leave
        ret
```

On assemble ce programme et on l'exécute dans GDB. Une trace incomplète est donnée ci-dessous :

```
(gdb) break before_call_swap
Breakpoint 1 at 0x80483fd: file swap.S, line 37.
(gdb) run
Starting program: swap
Breakpoint 1, before_call_swap () at swap.S:37
# Afficher 16 mots, en hexa, à partir du pointeur de pile %esp:
(gdb) x/16x $esp
```

```
0xbfffe4d0: 0x080495f0 0x________ 0xbfffe558 0x40036ca6
0xbfffe4e0: 0x00000001 0xbfffe584 0xbfffe58c 0x4001fbd8
0xbfffe4f0: 0xbfffe540 0xffffffff 0x4001bff4 0x08048234
0xbfffe500: 0x00000001 0xbfffe540 0x4000d966 0x4001cab0
# Afficher le contenu du registre %esp, en hexa
(gdb) p /x $esp
$1 = 0x________
(gdb) p /x $ebp
$2 = 0x________
(gdb) break within_swap
Breakpoint 2 at 0x80483d1: file swap.S, line 12.
# Continuer jusqu'au breakpoint suivant
(gdb) continue
Breakpoint 2, within_swap () at swap.S:12
(gdb) x/16x $esp
0xbfffe4c8: 0xbfffe4d8 0x08048402 0x080495f0 0x080495ec
0xbfffe4d8: 0xbfffe558 0x40036ca6 0x00000001 0xbfffe584
0xbfffe4e8: 0xbfffe58c 0x4001fbd8 0xbfffe540 0xffffffff
0xbfffe4f8: 0x4001bff4 0x08048234 0x00000001 0xbfffe540
(gdb) p /x $eax
$3 = 0x________
(gdb) p /x $ecx
$4 = 0x80495ec
(gdb) p $esi
$5 = ________
(gdb) p $edi
$6 = ________
```

Question 1 (1.5 points) *6 valeurs ont été remplacées par des* `________`*. Donnez ces 6 valeurs, avec pour chacune une explication d'une ou deux phrases.*

```
(gdb) break before_call_swap
Breakpoint 1 at 0x80483fd: file swap.S, line 37.
(gdb) run
Starting program: swap

Breakpoint 1, before_call_swap () at swap.S:37
37 call swap
Current language:  auto
The current source language is "auto; currently asm".
(gdb) x/16x $esp
0xbfffe4d0: 0x080495f0 0x080495ec 0xbfffe558 0x40036ca6
0xbfffe4e0: 0x00000001 0xbfffe584 0xbfffe58c 0x4001fbd8
0xbfffe4f0: 0xbfffe540 0xffffffff 0x4001bff4 0x08048234
```

```
0xbfffe500: 0x00000001 0xbfffe540 0x4000d966 0x4001cab0
(gdb) p /x $esp
$1 = 0xbfffe4d0
(gdb) p /x $ebp
$2 = 0xbfffe4d8
(gdb) break within_swap
Breakpoint 2 at 0x80483d1: file swap.S, line 12.
(gdb) continue
Continuing.

Breakpoint 2, within_swap () at swap.S:12
12 movl %esi, (%ecx)
(gdb) x/16x $esp
0xbfffe4c8: 0xbfffe4d8 0x08048402 0x080495f0 0x080495ec
0xbfffe4d8: 0xbfffe558 0x40036ca6 0x00000001 0xbfffe584
0xbfffe4e8: 0xbfffe58c 0x4001fbd8 0xbfffe540 0xffffffff
0xbfffe4f8: 0x4001bff4 0x08048234 0x00000001 0xbfffe540
(gdb) p /x $eax
$3 = 0x80495f0
(gdb) p /x $ecx
$4 = 0x80495ec
(gdb) p $esi
$5 = 54
(gdb) p $edi
$6 = 42
```

Question 2 (1 point) *Dessinez la pile au moment où le programme se trouve sur le point d'arrêt* `within_swap`. *Représentez également les cases mémoires des variables* `x` *et* `y`, *et les registres concernés, en représentant les pointeurs par des flèches.*

2 Implémentation d'une fonction simple en assembleur

Soit la fonction C suivante :

```
void ecrire_peut_etre(int valeur, int *destination) {
        if (destination != NULL)
                *destination = valeur;
}
```

Question 3 (1 point) *Traduire cette fonction en assembleur.*

```
.globl ecrire_peut_etre
        param_valeur=8
```

```
        param_destination=12
ecrire_peut_etre:
        pushl %ebp
        movl %esp, %ebp
        ## if (destination != NULL)
        movl param_destination(%ebp), %eax
        cmpl $0, %eax
        je fin_if
        ##         *destination = valeur;
        movl param_valeur(%ebp), %eax
        movl param_destination(%ebp), %ecx
        movl %eax, (%ecx)

fin_if:
        leave
        ret
```

3 Liste chaînée

Soit une liste chaînée définie par :

```
struct cellule {
        int val;
        struct cellule *suiv;
};
```

Question 4 (1 point) *Écrire en C une fonction* `chercher_42(...)` *la plus simple possible (une fonction plus compliqué ne donnera pas la note maximum) qui renvoie vrai si la liste contient la valeur 42.*

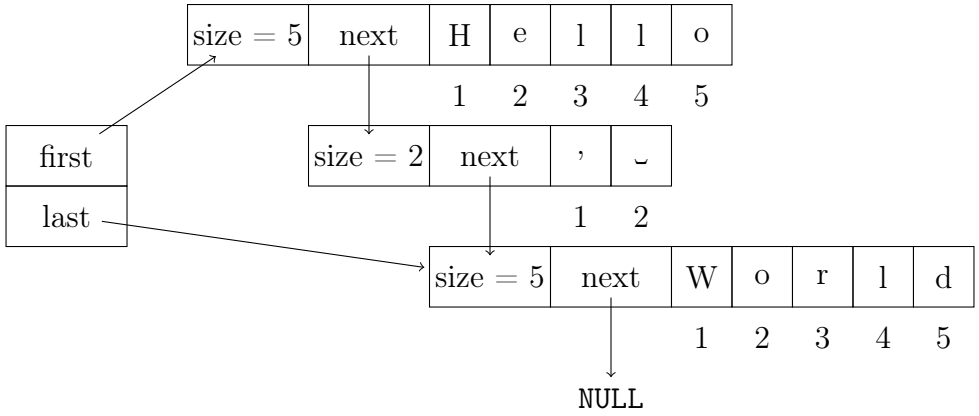
```
// Passage par valeur :
// on modifie les elements, pas les pointeurs.
int incrementer_liste(struct cellule *l)
{
        while (l != NULL) {
                if (l->val == 42)
                        return 1;
                l = l->suiv;
        }
        return 0;
}
```

4.5

4 Problème : Implémentation d'une structure de donnée « Unrolled Linked List »

On a souvent le choix entre utiliser un tableau et utiliser une liste chaînée. Un parcours de tableau est plus efficace (mémoire contiguë, donc meilleure utilisation du cache), mais les listes chaînées ont aussi des avantages : concaténation peu coûteuse, ajout en tête ou en queue en O(1) (si on garde un pointeur sur le dernier élément).

Cet exercice propose d'implémenter un compromis entre les deux, appelé « unrolled linked lists » dans la littérature. Le principe est d'avoir une liste chaînée de tableaux, comme illustré ci-dessous :



Nous représentons ici des chaînes de caractères : chaque élément de la liste contient une suite de `char` (le `'\0'` final n'est pas nécessaire ici), un champ `size` (nécessaire pour connaître le nombre de caractères puisqu'on n'a pas de `'\0'` final), et un pointeur sur l'élément suivant de la liste. Pour que l'insertion en queue et la concaténation soient efficaces, on garde un pointeur sur le premier élément et sur le dernier. La définition en C est donc la suivante :

```
// data structure used internally
struct unrolled_list_elem {
        size_t size;
        struct unrolled_list_elem *next;
        char content[];
};

// data structure used by the user
struct unrolled_list {
        struct unrolled_list_elem *first;
        struct unrolled_list_elem *last;
};
```

Au cours de cet exercice, nous utiliserons la fonction `memcpy`, donc on rappelle la spécification en annexe. On y donne également un exemple de programme principal utilisant cette structure et un cas particulier d'utilisation de `printf` est aussi décrit.

On suppose les constantes suivantes définies :

```
offset_first = 0
offset_last = 4
```

```
offset_size = 0
offset_next = 4
offset_content = 8
```

4.1 Parcours de la structure

On considère la fonction C suivante :

```
size_t unrolled_list_XXX(struct unrolled_list *r) {
        struct unrolled_list_elem *cur;
        size_t XXX = 0;
        for (cur = r->first; cur != NULL; cur = cur->next) {
                XXX += cur->size;
        }
        return XXX;
}
```

Question 5 (1 point) *Que fait cette fonction ?*

Question 6 (1.5 points) *Traduire cette fonction en assembleur.*

On considère maintenant la fonction assembleur suivante :

```
        .section .rodata
fmt_star_s:     .asciz "%.*s"
        .text
        /* ... */
        .globl unrolled_list_YYYY
loc_cur = -4
arg_r = 8
unrolled_list_YYYY:
        pushl %ebp
        movl %esp, %ebp
        /* ... */
        subl $4, %esp
        /* ... */
        movl arg_r(%ebp), %eax
        movl offset_first(%eax), %eax
        movl %eax, loc_cur(%ebp)
        /* ... */
debut_for_YYYY:
        cmpl $0, loc_cur(%ebp)
        je end_for_YYYY
        /* ... */
        movl loc_cur(%ebp), %eax
        addl $offset_content, %eax
        pushl %eax
        movl loc_cur(%ebp), %eax
        pushl offset_size(%eax)
        pushl $fmt_star_s
        call printf
        addl $12, %esp
        /* ... */
```

```
        movl loc_cur(%ebp), %eax
        movl offset_next(%eax), %eax
        movl %eax, loc_cur(%ebp)
        /* ... */
        jmp debut_for_YYYY
end_for_YYYY:
        leave
        ret
```

Pour vous aider, on a gardé des commentaires vides `/* ... */` correspondant aux lignes de code C traduites en assembleur.

Question 7 (1.5 points) *Traduire cette fonction en C.*

Fonction `unrolled_list_print` dans `unrolled_list_asm.S`

Question 8 (0.5 point) *Que fait cette fonction ?*

Elle affiche la chaîne.

Question 9 (2 points) *Écrire une version optimisée de cette fonction, toujours en assembleur :*

- *La nouvelle fonction ne devra pas avoir de variable locale, mais devra utiliser uniquement les registres.*
- *On remplacera l'utilisation de l'instruction* `addl $offset_content, %eax` *par mode d'adressage approprié (ce qui fait gagner une instruction). L'instruction* `leal` *(load effective address) peut aider (l'instruction* `leal <mode-d'adressage>, %eax` *met l'adresse de la case mémoire désignée par* `<mode-d'adressage>` *dans* `%eax`*).*

Fonction `unrolled_list_print_opt` dans `unrolled_list_asm.S`

4.2 Constructeur

On cherche à pouvoir écrire le constructeur de la structure de données de la manière suivante :

```
// Cree une ``unrolled list'' contenant les caracteres
// de la chaine C ``content'' (terminee par un '\0').
struct unrolled_list *unrolled_list_new(char *content) {
        struct unrolled_list_elem *elem
                = unrolled_list_elem(content);
        return unrolled_list_wrap(elem);
}
```

Question 10 (1 point) *Traduire cette fonction en assembleur.*

Question 11 (1.5 points) *Que doivent faire les fonctions* `unrolled_list_elem` *et* `unrolled_list_wrap` *? Donner une implémentation de ces fonctions en C.*

4.3 Destructeur

Question 12 (1.5 points) *Implémenter en C une fonction* `unrolled_list_free` *qui libère toute la mémoire utilisée par une* `struct unrolled_list` *(en utilisant la fonction C* `free` *autant de fois que nécessaire).*

4.4 Ajout en fin

Question 13 (1.5 points) *Écrire en C une fonction efficace*

```
void unrolled_list_append(struct unrolled_list *dest, char *to_append);
```

qui ajoute la chaîne C `to_append` *(terminée par un* `'\0'`*) à la "unrolled list"* `dest` *(en modifiant cette dernière). Quelle est la complexité algorithmique de cette fonction ? (i.e. combien d'opérations sont nécessaires en fonction de la taille des entrées ?)*

4.5 Optimisation

Une "unrolled list" obtenue par concaténations successives via `unrolled_list_append` de petites chaînes risque d'être inefficace, puisqu'elle sera constituée d'une multitude de `struct unrolled_list_elem` de petite taille. On va maintenant écrire une fonction

```
struct unrolled_list *unrolled_list_pack(struct unrolled_list *r);
```

qui, sans modifier la liste `r` passée en paramètre, va construire et renvoyer une `struct unrolled_list *` ne contenant qu'un seul `struct unrolled_list_elem`, avec le même contenu que `r`.

Question 14 (2 points) *Implémentez en C la fonction* `unrolled_list_pack`*. Il peut être utile d'appeler une des fonctions déjà définies ci-dessus.*

Question 15 (3 points) *(Attention, question difficile). Traduire cette fonction en assembleur.*

```
17
21.5
```

A Annexes

A.1 Fonction memcpy

```
NAME
       memcpy - copy memory area

SYNOPSIS
       #include <string.h>

       void *memcpy(void *dest, const void *src, size_t n);

DESCRIPTION
       The memcpy() function copies n bytes from memory area src to
       memory area dest.  The memory areas should not overlap.  Use
       memmove(3) if the memory areas do overlap.

RETURN VALUE
       The memcpy() function returns a pointer to dest.
```

A.2 Programme d'exemple

```
/* Programme de test qui affiche :

hello, world!
len = 14
hello, world!
hello, world!
 */

#include "unrolled_list.h"
#include <stdio.h>

int main() {
        struct unrolled_list *r = unrolled_list_new("hello");
        unrolled_list_append(r, ", ");
        unrolled_list_append(r, "world!\n");
        unrolled_list_print(r);
        printf("len = %d\n", unrolled_list_len(r));
        struct unrolled_list *r2 = unrolled_list_pack(r);

        unrolled_list_print(r2);
        // Version optimisee de la meme fonction.
        unrolled_list_print_opt(r);

        unrolled_list_free(r);
        unrolled_list_free(r2);
        return 0;
}
```

A.3 Affichage d'un nombre fixe de caractères avec `printf`

L'utilisation classique de printf pour afficher une chaîne est la suivante :

```
char *null_termined_string = /* ... */;
/* Affiche les caractères depuis l'adresse
   "null_terminated_string". S'arrête juste
   avant le premier caractère nul */
printf("%s", null_terminated_string);
```

On pourra dans cet exercice utiliser une variante, qui permet de spécifier le nombre d'octets à afficher comme argument de `printf`, et donc d'afficher une séquence de caractères non terminée par le traditionnel caractère nul :

```
char *any_string = /* ... */;
/* Affiche "size" caractères à partir de
   l'adresse "any_string" */
printf("%.*s", size, any_string);
```