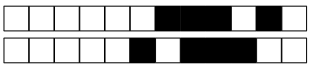


+58/1/28+

ESIR SYS1 CC2 2022–2023
12 janvier 2023

Nom et Prénom

y y
numéro d'étudiant : 99999991

Durée : 1h, Notation : sur 20 points

---

1 Première partie : QCM (10 points)

Instructions :

Cochez clairement la case de la réponse que vous pensez être juste. Il y a **une seule réponse** juste par question.

Barème :

+0.5 pour chaque réponse correcte
$-0.5/m$ pour chaque réponse fausse (où m+1 est le nombre de réponses possibles)

Question 1 Soit le programme assembleur `nasm` suivant (le caractère ASCII 10 opère un retour à la ligne sous Unix.) :

```
SECTION     .data
message:    db 'HELLO WORLD!',10 ; 10 is the ASCI code for '\n'
len:        equ $-message

SECTION    .text
  GLOBAL    _start
_start:
  mov       rsi, 0
loop:
  cmp       BYTE [message+rsi] , '!'
  je        end
  add       BYTE [message+rsi] , 'a'-'A' ; computed by nasm
  inc       rsi
  jmp       loop
end:
  mov       rsi,message
  mov       rdx,len
  mov       rax, 1              ; system call for write
  mov       rdi, 1              ; file handle 1 is stdout
  syscall                       ; invoke operating system to do the write

  mov       rax, 60             ; system call for exit
  mov       rdi, 0              ; exit code 0, xor rdi, rdi is used (faster, shorter)
  syscall                       ; invoke operating system to exit
```

Quelle sortie ce programme produit–il ?

- ☐ `hello worldA`
- ☐ `hello world!`
- ☐ `hello@world!`

Question 2 Quelle fonction C permet–elle de libérer de la mémoire allouée sur le tas (*heap*) ?

- ☐ une telle fonction n'existe pas.
- ☐ delete
- ☐ free

Question 3 Sur combien de bits fonctionne le registre `ecx` dans les architectures x86-64 ?

- ☐ 16
- ☐ 32
- ☐ 64

Question 4 Dans un processeur utilisant la technologie de *pipelining*, l'unité de chargement d'instructioin (Instruction Fetch Unit) et l'unité de décodage d'instruction (Instruction Decode Unit) d'un même coeur peuvent–elles être actives simultanément ?

- ☐ non
- ☐ oui

Question 5 Si l'on modifie l'implantation d'une bibliothèque dynamique (ou partagée, *shared librairies*), faut–il nécessairement recompiler tous les exécutables qui utilisent cette bibliothèque pour leur permettre d'utiliser la nouvelle version de la bibliothèque ?

- ☐ oui
- ☐ non

Question 6 L'on souhaite écrire la chaîne `"Hello World!"` dans le fichier `mon_fichier.txt`. Le code suivant d'un `main` de programme C est-il correct pour réaliser cette opération ? (Des `#include` on été omis par concision.)

```
int main(int argc, char** argv) {
  fprintf("mon_fichier.txt", "Hello World!");
  return 0;
}
```

- ☐ non
- ☐ oui

Question 7 Le bytecode java est directement exécutable par un processeur.

- ☐ faux
- ☐ vrai

Question 8 On considère les instructions assembleur x86-64 suivantes :

```
push rbp
mov  rbp, rsp
```

À quoi ces deux lignes peuvent–elles correspondre ?

- ☐ à l'invocation d'une fonction
- ☐ au prologue d'une fonction
- ☐ à l'épilogue d'une fonction

Question 9 Le code C suivant produit–il un fichier binaire ou un fichier texte ? (Des `#include` on été omis par concision.)

```
int main( int argc, char** argv) {
  unsigned int temperatures[] = {8,10,17,5,20};
  FILE * f = fopen("temps","w");
  fwrite(temperatures,sizeof(unsigned int), 5, f);
  fclose(f);
  return 0;
}
```

- ☐ un fichier binaire
- ☐ un fichier texte
- ☐ Ce code ne produit pas de fichier.

Question 10 Laquelle des assertions suivantes est-elle vraie ?

- ☐ Un cache L2 est plus rapide qu'un cache L1.
- ☐ Un cache L1 est plus rapide qu'un cache L2.

Question 11 Où sont allouées les variables automatiques ?

- ☐ sur le segment BSS
- ☐ sur la pile
- ☐ sur le tas

Question 12 Un code assembleur est directement exécutable par un processeur.

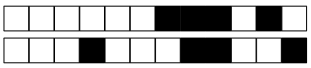
- ☐ faux
- ☐ vrai

Question 13 Lorsque l'on ouvre un fichier en C, quel mode permet d'ajouter (*append*) des information à un fichier pré–existant sans l'écraser ?

- ☐ `"w+"`
- ☐ `"a"`
- ☐ `"r+"`
- ☐ `"w"`

Question 14 Si l'exécutable `a.out` utilise la librairie partagée (*shared library*) `libfoo.so`, trouvera–t–on le contenu de `libfoo.so` dans le fichier `a.out` ?

- ☐ oui
- ☐ non



Question 15 On considère le programme C suivant :

```
int i = 6 ;
int main( int argc, char** argv) {
  i = i + 3 ;
} // EndMain
```

À l'exécution de ce programme, ou est stockée la variable `i` ?

- ☐ dans le segment de données du programme (*data segment*)
- ☐ dans le tas (*heap*)
- ☐ dans la pile (*stack*)

Question 16 Comment peut–on compiler le fichier `test.c` avec `gcc` sans réaliser l'édition de lien (*linking*) ?

- ☐ `gcc -c test.c`
- ☐ `gcc test.c`
- ☐ `gcc -o test.c`

Question 17 L'exécutable d'un programme qui utilise des fonctions fournies par une librairie partagée va–t–il contenir des symboles non–définis ? (marqués par un `U` dans la sortie de l'outil `nm` vu en cours)

- ☐ oui
- ☐ non

Question 18 Peut–on désallouer de façon explicite une variable automatique ?

- ☐ oui
- ☐ non

Question 19 Sous Unix, un exécutable peut–il utiliser lors de son exécution une bibliothèque partagée (*shared library*) dont le code a été compilé plus récemment que celui de l'exécutable ?

- ☐ Uniquement s'il s'agit d'un module noyau.
- ☐ Oui, c'est possible.
- ☐ Non, c'est impossible.

Question 20 L'appel suivant est-il possible en assembleur nasm x86-64 ?

```
call foo(10,20)
```

- ☐ oui
- ☐ non

2 Deuxième partie : Questions ouvertes et problèmes (10 points)

Question 21 On considère le programme C suivant (certains `#include` on été retirés pour plus de lisibilité).

```
void foo(int i) {
  printf("%i\n",  i);
  if (i>0) {
    foo(i-1);
  }
}

int main( int argc, char** argv) {
  foo(5);
}
```

L'on compile ce programme avec `gcc`. L'exécution du programme produit la sortie suivante :

```
$ ./foo
5
4
3
2
1
0
```

L'on décompile ensuite l'exécutable obtenu grâce à l'outil `objdump` utilisé en cours. Dans le code décompilé de `main`, la fonction `foo` est appelée de la manière suivante, où `1135` représente l'adresse en hexadécimal du début de `foo` :

```
mov    edi,0x5
call   1135 <foo>
```

Quel type de passage de paramètre le compilateur a-t-il utilisé ? Mentionnez les éléments pertinents du code assembleur et du code source pour étayer votre réponse.

[2 points] ☐A+ ☐A ☐B ☐F

Question 22 (Suite de la question 21.) On s'intéresse maintenant au code décompilé de la fonction `foo` présentée à la question 21. Le code décompilé de `foo` produit par `objdump` est le suivant :

```
0000000000001135 <foo>:
    1135: 55                    push   rbp
    1136: 48 89 e5              mov    rbp,rsp
    1139: 48 83 ec 10           sub    rsp,0x10
    113d: 89 7d fc              mov    DWORD PTR [rbp-0x4],edi
    1140: 8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
    1143: 89 c6                 mov    esi,eax
    1145: 48 8d 3d b8 0e 00 00  lea    rdi,[rip+0xeb8]        # 2004 <_IO_stdin_used+0x4>
    114c: b8 00 00 00 00        mov    eax,0x0
    1151: e8 da fe ff ff        call   1030 <printf@plt>
    1156: 83 7d fc 00           cmp    DWORD PTR [rbp-0x4],0x0
    115a: 7e 0d                 jle    1169 <foo+0x34>
    115c: 8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
    115f: 83 e8 01              sub    eax,0x1
    1162: 89 c7                 mov    edi,eax
    1164: e8 cc ff ff ff        call   1135 <foo>
    1169: 90                    nop
    116a: c9                    leave
    116b: c3                    ret
```

Remarques :

— Dans ce code, `DWORD PTR` indique une zone mémoire stockant une valeur sur 4 octets (*double word*).

— L'instruction `lea rdi,[rip+0xeb8]` (*load effective address*) est ici équivalente à `mov rdi, 0x2004`.

— `0x2004` est l'addresse mémoire de la chaîne de formattage `"%i\n"` utilisée par `printf` dans `foo`.

— Enfin, `nop` (*no operation*) représente une instruction de remplissage, qui ne fait rien.

Sous–questions :

(a) Comment s'appellent les trois premières instructions assembleur de ce code ? À quoi servent–elles ? `[3 points]`

(b) Dans quelle zone de la mémoire la variable `i` est–elle stockée ? Vous justifierez votre réponse en faisant référence au code décompilé. `[2 points]`

(c) Quelles instructions implémentent le bloc `if` présent dans le code source de la fonction `foo` ? Que fait chacune d'entre elles ? `[3 points]`

[8 points] ☐A+ ☐A ☐B+ ☐B ☐C+ ☐C ☐D ☐E ☐F