Université Paris Diderot – Paris 7 **Programmation réseau**
L3 Informatique **Anne 2010-2011, 2ème semestre**

TP n°2

Des bases pour la programmation réseau en C

1 Bien compiler et déboguer en C

Exercice 1 [Lire les warning pour mieux programmer] Nous considérons le programme suivant :

```
#include <stdio.h>

int main() {
  int a;
  a=1;
  printf("%d\n", a);
}
```

Compiler ce programme dans un premier temps avec `gcc` puis en utilisant `gcc -Wall`. Que constatez-vous ? Comment remédier à cela ?

Remarque : Dorénavant, vous devrez compiler toujours en utilisant l'option `-Wall` et supprimer tous les warnings obtenus.

Exercice 2 [Débogage post-mortem] Nous considérons le programme suivant :

```
#include <stdio.h>

int main() {
  int *p=NULL;
  printf("%d\n",*p);
  return 0;
}
```

Compilez ce programme et exécutez-le. Que se passe-t-il ? Pourquoi ?
Afin de détecter d'où vient l'erreur, vous pouvez analyser l'exécution de votre programme "post-mortem" en utilisant `gdb`. Pour cela, vous devez faire les opérations suivantes :

1. Autoriser votre système à créer des `core dump` pour stocker l'état de la mémoire au moment où le processus a crashé. Il faut faire `ulimit -c unlimited`.
2. Compiler en donnant l'option `-g` de façon à produire les fichiers utiles au débogage.

3. Lorsque vous exécutez votre programme, un fichier nommé `core` sera alors créé.
4. Lancer `gdb` avec `a.out` et `core` comme arguments. Que vous affiche-t-il ? Tapez `bt` (pour *backtrace*) que vous affiche-t-il ?
5. En tapant `start` et en tapant ensuite `step`, `gdb` vous permet de regarder la trace de l'exécution pas à pas.

Utilisez `gdb` pour voir où le programme suivant fait une erreur.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _liste{
  int val;
  struct _liste *next;
} * liste;

int main() {
  liste t=NULL;
  liste h=malloc(sizeof(struct _liste));
  h->val=2;
  h->next=t;
  t=h;
  h=malloc(sizeof(struct _liste));
  h->val=3;
  h->next=t;
  t=h;
  printf("%d\n",t->val);
  t=t->next;
  printf("%d\n",t->val);
  t=t->next;
  printf("%d\n",t->val);
  return 0;
}
```

Exercice 3 [Débogage par simulation] Compilez et exécutez le programme suivant :

```
#include <stdio.h>

int main() {
  int a;
  printf("%d\n",a);
  return 0;
}
```

Que se passe-t-il ? Pourquoi ? De façon à trouver les erreurs vous pouvez utiliser l'outil `valgrind` (avec l'option `--leak-check=full`) en lui donnant comme argument l'exécutable. Utilisez `valgrind` sur les programmes de l'exercice 2. (Compilez

les avec l'option `-g`).Pour finir, utilisez `valgrind` sur les deux programmes suivants et déduisez-en les erreurs de programmation.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _liste{
  int val;
  struct _liste *next;
} * liste;

int main() {
  liste t=NULL;
  liste h=malloc(sizeof(struct _liste));
  h->val=2;
  h->next=t;
  t=h;
  h=malloc(sizeof(struct _liste));
  h->val=3;
  h->next=t;
  h=t;
  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct _liste{
  int val;
  struct _liste *next;
} * liste;

int main() {
  liste t=NULL;
  liste h=malloc(sizeof(struct _liste));
  h->val=2;
  h->next=t;
  free(h);
  free(h);
  return 0;
}
```

2 Programmation de processus linux

Exercice 4 [Création de processus]

- Écrivez un programme qui crée un processus fils. Chaque processus affichera si il est le père ou le fils ainsi que l'identité du fils (pour le père) ou du père (pour le fils).
- Écrivez un programme similaire à celui précédemment, mais cette fois-ci faîtes-en sorte que le père boucle toujours. Regardez ce qu'il advient du processus fils avec la commande `ps`.

Indication : il faudra vous servir des fonctions `fork`, `getpid`, `getppid`, `exit` *et* `wait`. *Pour savoir ce qu'elles font, consultez les pages* `man`.

Exercice 5 [Copie de l'espace mémoire] Écrivez un programme qui manipule une variable entière `i` et qui crée un processus fils. Le père incrémentera deux fois de 1 la variable et entre les deux incréments dormira 2 secondes, et le fils incrémentera deux fois de 100 la variable et entre les deux incréments dormira 2 secondes. Après chaque incrément, chaque processus affichera la valeur de la variable `i`. Que constatez-vous ?

Exercice 6 [Écriture dans un fichier] Écrire un programme qui crée un processus fils. Le père et le fils doivent ensuite écrire tous les deux dans le même fichier, le père écrira "Je suis le pere" et le fils "Je suis le fils".
Indication : pour écrire dans un fichier, il faudra vous servir des fonctions `fopen`, `fclose` *et* `fprintf`. *Là encore, pour savoir ce qu'elles font et comment les utiliser, consultez les pages* `man`.