# Cours "Informatique Embarquée"

**François Armand**

## M2 SRI/Crypto

## Exercice N° 4, 14 Novembre 2014

## A rendre avant le 20/11/2014 23H59

**armand@informatique.univ-paris-diderot.fr**

### Trouvez les bugs!

Une grande partie des systèmes embarqués sont codés en C. Les extraits de code C ci-après contiennent quelques erreurs ou dangers potentiels. Identifiez-les, expliquez pourquoi il s'agit de bugs ou de dangers et **proposez une correction**.

Attention, chaque extrait de code peut contenir plusieurs erreurs. **Il ne s'agit pas de déterminer des erreurs de compilation** puisque vous n'avez à votre disposition qu'un extrait de code, mais plutôt des erreurs ou dangers à l'exécution.

**Code 1:**

Celui-ci devrait être facile.

```
void main (int argc, char **argv)
{
    long i;

    if (argc = 1) printf("Veuillez saisir un argument");
    i += strtol(argv[1], NULL, 10);
    printf("i = %d\n", i);
}
```

**Code 2:**

Celui-là est à peine plus compliqué.

```
#define IS_VALID_CHAR(x) (((x)>='0' && (x)<='9') || \
                          ((x)>='A' && (x)<='F')  ?1:0)

void ma_fonction(char* chaine)
{
    int lg;
    while(IS_VALID_NUM(*chaine++)) {
        lg++;
    }
    printf("%s commence par %d digits hexa\n",
           lg, chaine);
}
```

**Code 3:**

Cet extrait de code se comporte-t-il de la même manière sur une machine 32 bits et sur une machine 64 bits? Détaillez.

```
int res;
long start_usecs, stop_usecs, delta2_usecs;
struct timeval t1, t2;

res = gettimeofday(&t1, NULL);
my_func();
res = gettimeofday(&t2, NULL);

stop_usecs= (t2.tv_sec * 1000000) + t2.tv_usec;
start_usecs= (t1.tv_sec * 1000000) + t1.tv_usec;
delta2_usecs = stop_usecs - start_usecs;
```

**Code 4:**

Un classique!

```
void * myfunc(void *myarg) {
   printf("Myfunc\n");
}

main() {
  pthread_t *th;
  int res;
  res = pthread_create(th, NULL, myfunc, NULL);
  if (res = 0) printf("Thread create OK!\n");
}
```

**Code 5:**

```
char * save_str(char * buf, unsigned int lg)
{
  char *my_str;
  my_str= malloc(lg);

  strcpy(my_str, buf);
  return(my_str);
}
```

**Code 6:**

```
int res, i;
  // Let's create 20 processes
  for (i =0; i < 20; i++){
    res = fork();
    printf("Yippie!\n");
  }
```

**Code 7:**

Une variation sur le thème précédent! Corriger l'extrait de programme si nécessaire.

```
void * my_func(void* arg)
{
int res;
int i;

  // Let's create 20 threads
  for (i; i < 20; i++){
    res = pthread_create(...., NULL, my_func, NULL);
}
```

**Code 8:**

L'extrait N°8 est un programme complet qui compile normalement sans erreur et sans warning (option -Wall). Comme d'habitude, c'est un programme qui n'est pas d'une utilité foudroyante, mais c'est encore une fois mon programme :-)

Le problème ici est en fait dépendant des options utilisées pour la compilation. Quand vous aurez trouvé la manifestation de ce problème, vous proposerez une correction qui permette à ce programme de se comporter normalement. Vous expliquerez aussi pourquoi les options de compilations influent sur le comportement de ce programme avant votre correction.

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

int check;
struct sigaction act;

void wakeup(int signb)
{
  check++;
  if (check == 11){
    printf("received 11 SIGALRM, Erreur, Stop!\n");
    exit(1);
  }
  if (alarm(1) < 0) {
    perror("cannot set alarm");
    exit(1);
  }
}

int main(int argc, char** argv)
{
  int cur;

  act.sa_handler=wakeup;
  if ( sigaction(SIGALRM, &act, NULL) <0) {
    perror("sigaction failed");
    exit(1);
  }
  if (alarm(1) < 0) {
    perror("cannot set alarm");
    exit(1);
  }
  for(cur=check; cur!=10;){
    if (check > cur) {
      printf("Recu %d ticks\n", check-cur);
      cur = check;
    }
  }
  return 0;
}
```

**Code 9: Gestion de signal**

```
void sig_handler(int sig){
  gettimeofday(&end, NULL);
}

int main(void){
  int rc;
  struct sigaction action;

  action.sa_handler = sig_handler;
  sigemptyset(&(action.sa_mask));
  action.sa_flags = 0;

  rc = sigaction(SIGCHLD, &action, NULL);

  switch(fork()){
    case 0:
      return EXIT_SUCCESS;
    case -1:
      return EXIT_FAILURE;
    default:
      pause();
  }
}
```

**Code 10:**

```
void my_struct_init(struct my_struct * pt_s, int val) {
  pt_s = malloc(sizeof(struct my_struct));
  pt_s->value = val;
  pt_s->time  = time(NULL);
  if (pt_s == NULL) perror("malloc failed!");
}
```

**Code 11:**

```
char * colimacon(unsigned int rows, unsigned int cols)
{
  char * pt;
  pt = malloc(sizeof((unsigned int) * rows * cols));
  if (pt == NULL) {
     perror("malloc failed!");
     return NULL;
  }
  fill_colimacon(pt); // remplit et rien d'autre
  free(pt);
  printf("colimacon OK");
}

main()
{
  char * tab;
  int i, j;
  tab = colimacon(4, 5);
  if (tab == NULL) exit(1);
  for (i = 0; i < 4; i++) {
      for (j = 0; j < 5; j++) {
          printf("%d\n", tab[i][j]);
      }
  }
}
```

**Code 12:**

```
char * colimacon(unsigned int rows, unsigned int cols)
{
  char * pt;
  pt = malloc(sizeof((unsigned int) * rows * cols));
  if (pt == NULL) {
     perror("malloc failed!");
     return NULL;
  }
  fill_colimacon(pt); // remplit et rien d'autre

}

main()
{
  char * tab;
  int i, res;
  static int ok, ko;
  for (i = 1; i < 10000; i++) {
     tab = colimacon(4*i, 5*i);
     res = check_colimacon(tab);
     (res == 0)? ok++ : ko++;
  }
  printf("Passed %d, Failed %d\n", ok, ko);
  return pt;
}
```

**Code 13 :**

```
char * printStringInBuffer (char * fmt, va_list pa)
{
 char * res;
 res = (char *)calloc(MAX_STRING_LENGTH, sizeof(char));
 if (res == NULL)

   return NULL;
 if (vsprintf(res, fmt, pa) >= MAX_STRING_LENGTH) {
   res[MAX_STRING_LENGTH-1] ='\0';
   return NULL;
 }
   return res;
}
```