Cours "Informatique Embarquée"

François Armand

M2 SRI/Crypto

Exercice N° 3, 14 Octobre 2011

A rendre avant Jeudi 27/10/2011 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 bug 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: (*3 points*)

Celui-ci devrait être facile.

```
main (int argc, char **argv)
{
    int i;
    if (argc ==1) i = atoi(argv[1]);
    if (i = 0) printf("0 is not valid!\n");
}
```

Code 2: (*2 points*)

Celui-là est à peine plus compliqué.

```
#define IS_VALID_NUM(x) (((x)>='0' && (x)<='9')?1:0)

void ma_fonction(char* chaine)
{
    int lg = 0;
    while(IS_VALID_NUM(*chaine++)) {
        lg++;
    }
    printf("Cette chaine commence par %d digits\n", lg);
}
```

Code 3: (*2 points*)

Tout ceux qui ont effectué le TP N° 2 devraient trouver ces lignes familières. Se comportent-elles de la même manière sur une machine 32 bits et sur une machine 64 bits?

```
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: (*1 point*)

Un classique!

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

main() {
  pthread_t *th;
  int res;
  res = pthread_create(th, NULL, myfunc, NULL);
  ...blah...blah...
}
```

Code 5: (*2 points*)

```
#define MAX_LEN 32

main(int argc, char* argv[1]) {
  char *my_str;
  my_str= malloc(MAX_LEN);

  if ((argv[1] != NULL) & (my_str != NULL))
     strcpy(my_str, argv[1]);
}
```

Code 6: (*2 points*)

```
int res, i;

  for (i =0; i < 20; i++){
    res = fork();
    ....blah blah...
  }
```

Expliquez, ce qui se passe avec le code ci-dessus. Y a-t-il une différence de comportement suivant que l'utilisateur exécutant ce programme est le super utilisateur de la machine ou un utilisateur quelconque? Si nécessaire, ré-écrivez le code, pour obtenir le résultat souhaité.

Code 7: (*2 points*)

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

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

  for (int i =0; i < 20; i++){
    res = pthread_create(...., NULL, my_func, NULL);
}
```

Code 8: (*4 points*)

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 exemplaire, 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: (*2 points*)

Pour l'extrait N°9, dites comment se comporte le programme. Proposez une modification si nécessaire.

```
void * my_func(void* arg)
{
   pthread_exit(0);
}

int main (int argc, char **argv; char **envp)
{
  pthread_t th;
  int res;

  res= pthread_create (&th, NULL, my_func(), NULL);
  printf("pthread_create %d\n", res);
}
```

Code 10: (*4 points*)

Cet extrait ne fonctionne pas toujours comme attendu!

1. Est-il possible que ce soit parce que « les signaux ne sont pas fiables »?
2. Quelle commande super-utilisateur sur un Linux récent faudrait-il entrer pour qu'il s'exécute toujours comme espéré?
3. En l'absence de la possibilité offerte par la réponse à la 2ème question ci-dessus (10.2), quelle solution apporteriez vous à ce problème?

```
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 11: (*2 points*)

Que fait ce programme? Faut-il le modifier? Si oui comment?

```
int main (int argc, char **argv; char **envp)
{
  int i;
  unsigned long delta, total;
  struct timeval t1, t2;

  total = 0;
  gettimeofday(&t1, NULL);

  for (i=0; i < 5; i++) {
    my_func();
    gettimeofay(&t2,NULL);
    delta = (t2.tv_sec - t1.tv_sec) * 1000000;
    delta += (t2.tv_usec - t1.tv_usec);
    printf("L'iteration %d a dure %d\n", i, delta);
    total += delta;
  }
  printf("Une iteration dure en moyenne %d\n", total/i);
}
```

Code 12: (*2 points*)

Ce programme se comporte-t-il de la même manière sur une machine 32 bits et sur un machine 64 bits? Si nécessaire, identifiez et corrigez la source de cette éventuelle différence.

```
int main (int argc, char **argv; char **envp)
{
  unsigned long delta;
  int i;
  struct timeval *t1
  struct timeval *t2;

  start = malloc(sizeof(struct timeval *));
  end   = malloc(sizeof(struct timeval *));

  if ((start == NULL) || (end == NULL)) exit(1);

  gettimeofay(&t1,NULL);
  for (i=0; i < 5; i++) {
    my_func();
  }
  gettimeofay(&t2,NULL);
  delta = (t2.tv_sec - t1.tv_sec) * 1000000;
  delta += (t2.tv_usec - t1.tv_usec);

  printf("my_func %d dure %d microsecs\n", i, delta/i);
}
```

Code 13: (*2 points*)

Pour cet extrait, dites comment se comporte le programme. Proposez une modification si nécessaire.

```
void * my_func(void* arg)
{
   printf("pthread created! Ouah trop fort!\n", res);
   pthread_exit(0);
}

int main (int argc, char **argv; char **envp)
{
  pthread_t th;
  int res;

  res= pthread_create (&th, NULL, my_func, NULL);
  return 0;
}
```