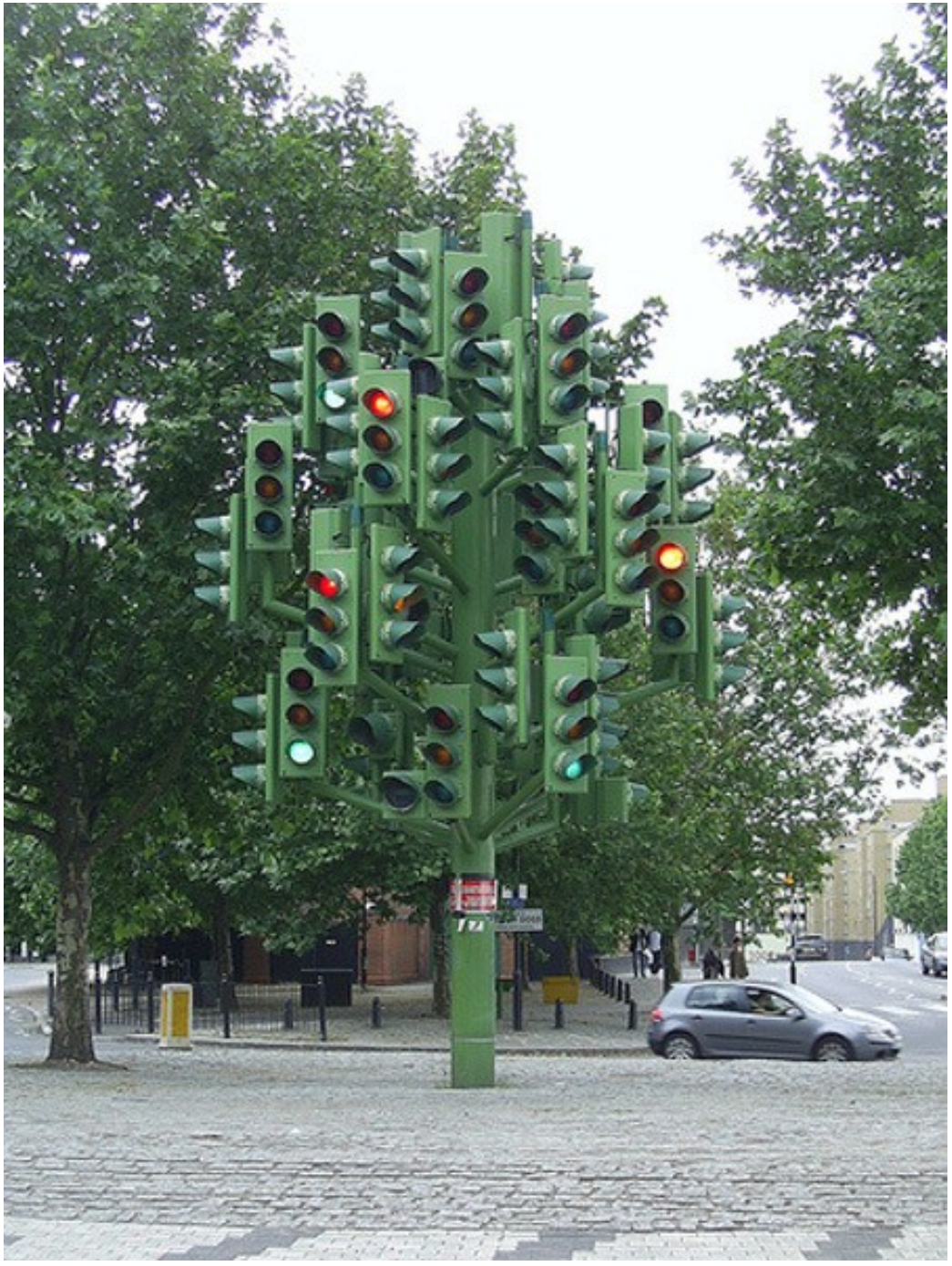
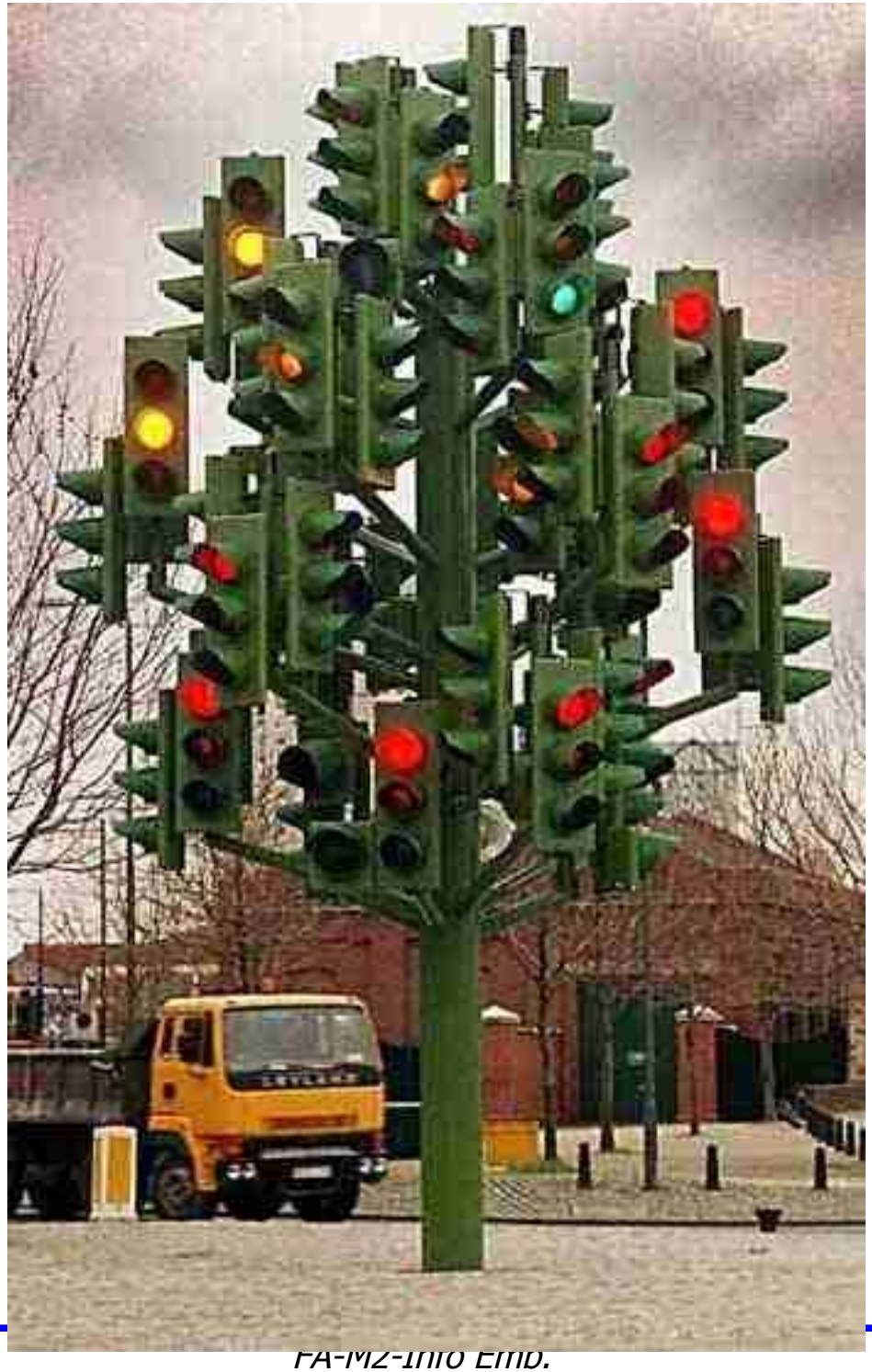
Informatique Embarquée
M2 / 2011 - 2012

BUGS (C)





Citations *(source wikipédia)*

- « Ce n'est pas un bug, c'est une fonctionnalité non documentée ! »
- « Tout programme non trivial possède au moins un bug. »
  - Ça ne veut pas dire que les autres n'en ont pas!
- « L'erreur est humaine, mais un vrai désastre nécessite un ordinateur. »
- « Quand un logiciel n'a plus aucun bug, il est habituellement désuet. »

Échauffement

- Quelle sont les tailles de ces structures?
  - Sur machine 32 bits et sur machines 64 bits?

```
struct s1 {
   int i; char c; int i2;
};
struct s2 {
   long i; char c;
};
```

Échauffement

```
struct s1 {
   int i; char c; int i2;
};
```

- 12 en 32 bits, 16 en 64 bits

```
struct s2 {
   long i; char c;
};
```

- 8 en 32 bits, 16 en 64 bits

Échauffement

```
struct s1 {
   int i; char c; int i2;
} __attribute__((__packed__));
```

- 9 en 32 bits, 13 en 64 bits

```
struct s2 {
   long i; char c;
}  __attribute__((__packed__));
```

- 5 en 32 bits, 9 en 64 bits

Code N°1

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

Code N°1

```
main (int argc, char **argv)
{
    int i = 0;
    if (argc ==1>=2) i =
                        atoistrtol (argv[1]);
    if (i == 0)
                printf("0 is not valid!\n");
}
```

argc == 2

atoi(argv[0])

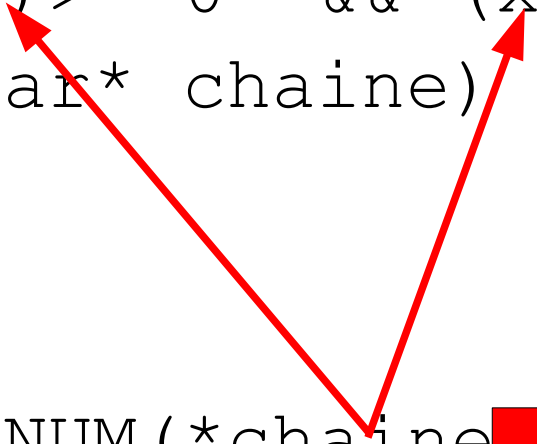
Code N°2

```
#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);
}
```

Ne saute pas un caractère sur 2

Code N°2

```
#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++; chaine++; // pas *chaine++
    }
printf("Cette chaine commence par %d
digits\n", lg);
}
```



Code N°3

```
int res;
long start, stop, delta;
struct timeval t1, t2;
res = gettimeofday(&t1, NULL);
my_func();
res = gettimeofday(&t2, NULL);
stop= (t2.tv_sec * 1000000) + t2.tv_usec;
start= (t1.tv_sec * 1000000) + t1.tv_usec;
delta = stop_usecs - start_usecs;
```

Code N°3

- Taille d'un long (tv_sec)
  - 32 bits => 4
  - 64 bits => 8
- $2^{32} =\sim 4 \times 10^{9}$
- 4×10^{9} / 1000 000 => ~4 000 (1H = 3 600 sec)
- On ne peut pas stocker une durée de plus de ~4000 secondes exprimée en µsec dans un long sur 32 bits.
- Donc pas le nombre de secondes depuis 1/1/70!

Code N°3

```
int res;
long long start, stop, delta;
struct timeval t1, t2;
res = gettimeofday(&t1, NULL);
my_func();
res = gettimeofday(&t2, NULL);
stop= (t2.tv_sec * 1000000) + t2.tv_usec;
start= (t1.tv_sec * 1000000) + t1.tv_usec;
delta = stop_usecs - start_usecs;
```

Code N°3

```
int res;
long start, stop, delta;
struct timeval t1, t2;
res = gettimeofday(&t1, NULL);
my_func();
res = gettimeofday(&t2, NULL);
delta= (t2.tv_sec - t1.tv_sec) * 1000000;
delta += t2.tv_usec - t1.tv_usec;
```

Tant que durée < ~1H...

Code N°4

```
void * myfunc(void *myarg) {
     printf("Myfunc\n");
}
main() {
    ~~pthread_t *th;~~pthread_t th;
     int res;
     res = pthread_create(&th, NULL,
                          myfunc, NULL);
...blah...blah...
}
```

Code N°5

```
#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 N°5

```
#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]);
}
```

Typo inoffensive

Bug volontaire...

Code retour pas testé...
Pas grave, si conditions OK

Code N°5

```
#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]);
}
```

Débordement possible....

```
strncpy(my_str, argv[1], MAX_LEN);
```

ou

```
malloc(strlen(argv[1]) + 1);
```

Code N°6

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

Code N°6

- C'est une « fork bomb ».
- Création de 2^{20}-1 processus
- Utilisateur lambda
  - => limite (ulimit -a) ou moins...
  - EAGAIN à partir d'un certain moment
- Utilisateur root => limite = place dans la « table des processus » (taille calculée dynamiquement par le noyau au démarrage – fonction taille mémoire-)

Code N°7

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

Code N°7

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

« Récursivité » => infinité de threads

Code N°7

Le problème est-il résolu?

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

Code N°7

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

Le problème est-il (bien) résolu?

Code N°8

```
int check;
void wakeup(int signb)
{
    check++;
    if (check == 11){
        printf("received 11 SIGALRM,"
               "Erreur, Stop!\n");
        exit(1);
    }
    alarm(1) ;
}
```

Code N°8 *(suite)*

```
for(cur=check; cur!=10;){
    if (check > cur) {
        printf("Recu %d ticks\n",
               check-cur);
        cur = check;
    }
}
```

Code N°8

```
volatile int check;
void wakeup(int signb)
{
    check++;
    if (check == 11){
        printf("received 11 SIGALRM,"
               "Erreur, Stop!\n");
        exit(1);
    }
    alarm(1) ;
}
```

Code N°9

```
void * my_func(void* arg)
{
    pthread_exit(0);
}
int main (int argc, char **argv)
{
    pthread_t th;
    int res;
    res= pthread_create (&th, NULL,
                         my_func(), NULL);
    printf("pthread_create %d\n", res);
}
```

Code N°9

```
void * my_func(void* arg)
{
    pthread_exit(0);
}
int main (int argc, char **argv)
{
    pthread_t th;
    int res;
    res= pthread_create (&th, NULL,
                         my_func(), NULL);
    printf("pthread_create %d\n", res);
}
```

Indépendamment des « warnings » éventuels de compilation, cette trace apparaitra-t-elle?

Code N°9

```
void * my_func(void* arg)
{
    pthread_exit(0);
}
int main (int argc, char **argv)
{
    pthread_t th;
    int res;
    res= pthread_create (&th, NULL,
                         my_func(), NULL);
    printf("pthread_create %d\n", res);
}
```

Invoque my_func ,à cause de ().
my_func termine la seul thread du processus!
RIP!

Code N°10

```
void sig_handler(int sig){
    gettimeofday(&end, NULL);
}
int main(void){
    rc = sigaction(SIGCHLD, &action, NULL);
    switch(fork()){
        case 0: return EXIT_SUCCESS;
        case -1: return EXIT_FAILURE;
        default: pause();
}
```

Code N°10

- Forcer l'exécution du père avant le fils:

```
# echo 1 > /proc/sys/kernel/sched_child_runs_first
```

- En tant que « root »
  - De toute manière, le multi-processeurs peut changer la donne...
- Sinon... masquer le signal SIGCLD jusqu'au moment où le père peut / veut le recevoir...

Code N°10

```
sa.sa_handler = sigchld;
sa.sa_flags = 0;
nr = sigemptyset(&sa.sa_mask);

nr = sigaction(SIGCHLD, &sa, NULL);
nr = sigemptyset(&set);

nr = sigaddset(&set, SIGCHLD);

nr = sigprocmask(SIG_BLOCK, &set, &old);
```

Code N°10

```
int main(void){
    rc = sigaction(SIGCHLD, &action, NULL);
    switch(fork()){
        case 0: return EXIT_SUCCESS;
        case -1: return EXIT_FAILURE;
        default: sigsuspend(&old);
}
```

Code N°11

```
unsigned long delta, total;

gettimeofday(&t1, NULL);
for (i=0; i < 5; i++) {
    my_func();
    gettimeofday(&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);
```

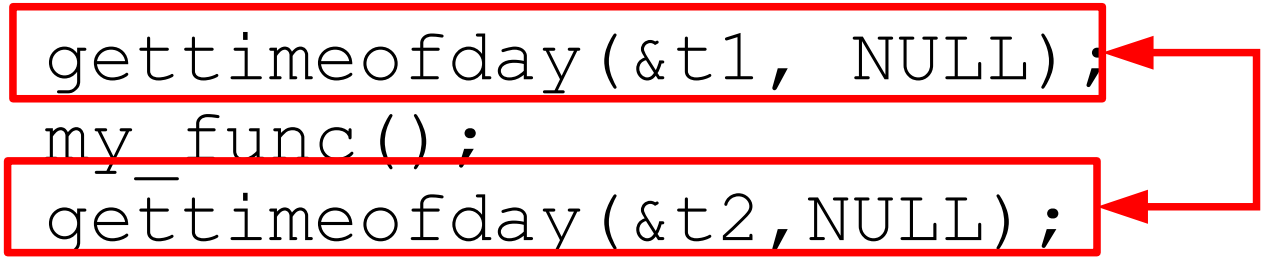
Code N°11

```
unsigned long delta, total;

gettimeofday(&t1, NULL);
for (i=0; i < 5; i++) {
    my_func();
    gettimeofday(&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 N°11

```
unsigned long delta, total;
for (i=0; i < 5; i++) {
    gettimeofday(&t1, NULL);
    my_func();
    gettimeofday(&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 N°11

```
unsigned long delta, total;

gettimeofday(&t1, NULL);
for (i=0; i < 5; i++) {
    my_func();
    gettimeofday(&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 N°12

```
t1 = malloc(sizeof(struct timeval *));
t2 = malloc(sizeof(struct timeval *));

gettimeofday(t1,NULL);

for (i=0; i < 5; i++) {
    my_func();
}

gettimeofday(t2,NULL);

delta = (t1->tv_sec - t2->tv_sec) * 1000000;
delta += (t2->tv_usec - t2->tv_usec);
```

Code N°12

```
t1 = malloc(sizeof(struct timeval*));
t2 = malloc(sizeof(struct timeval*));

gettimeofday(t1,NULL);

for (i=0; i < 5; i++) {
    my_func();
}

gettimeofday(t2,NULL);

delta = (t1->tv_sec - t2->tv_sec) * 1000000;
delta += (t2->tv_usec - t2->tv_usec);
```

Code N°13

```
void * my_func(void* arg)
{
    printf("pthread created!\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;
}
```

Code N°13

```
void * my_func(void* arg)
{
    printf("pthread created!\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);
    res = pthread_join(th, NULL);
    return 0;
}
```