Correction TP1 : Programmation Système (partie 1)

Contents

1 Rappel

Le but de ce premier TP était de vous familiariser avec l'environnement de travail d'un développeur sous Linux ainsi qu'avec quelques concepts de base de la programmation système :

- librairie partagée, gcc et Makefile
- segmentation fault, ulimit, fichier core et gdb
- ldd, LD_LIBRARY_PATH et LD_PRELOAD
- getopt, select, sigaction, fork, pipe et syslog

2 Exercice 1 : GDB et fichier core

Question 1 : *Que se passe-t-il au bout de quelques secondes? Qu'en déduisez vous?*

Au bout de quelques secondes, le simulateur part en segmentation fault. Il y a donc un problème d'accès mémoire quelque part dans le code du simulateur.

Question 2 : *Quel signal a reçu le processus pour se terminer ainsi? Comment vérifiez vous le numéro du signal reçu?*

Le signal que reçoit un processus dans le cas d'une erreur de segmentation est SIGSEGV. On le vérifie en récupérant la valeur de retour du binaire et en étudiant la liste des signaux :

```
> sh run.sh
PTTY: /dev/pts/4
run.sh: line 7:  4659 Segmentation fault      $ROOT_DIR/bin/gps
> echo $?
11
> kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Question 3 : *Grâce à GDB et au fichier **core** généré, analysez la source du problème du binaire **gps**. Quelle partie du code est fausse? Pourquoi?*

```
> ulimit
0
> ulimit -c unlimited
> sh run.sh
PTTY: /dev/pts/4
run.sh: line 7:  4659 Segmentation fault      $ROOT_DIR/bin/gps
> ls
bin  core  include  lib  Makefile  run.sh  src
> gdb ./bin/gps ./core
GNU gdb (Debian 7.10-1) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
```

```
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bin/gps...(no debugging symbols found)...done.
[New LWP 6028]
Core was generated by `/home/tergeist/devel/packages/embsys/labs/gps/bin/gps'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  strlen () at ../sysdeps/x86_64/strlen.S:106
106     ../sysdeps/x86_64/strlen.S: No such file or directory.
(gdb) bt
#0  strlen () at ../sysdeps/x86_64/strlen.S:106
#1  0x00007f51323d2a0c in _IO_puts (str=0x0) at ioputs.c:36
#2  0x00007f5132710ad1 in knot_to_kmh_str () from /home/tergeist/devel/packages/embsys/labs ↩
    /gps/lib/libnmea.so
#3  0x00007f5132710ed3 in nmea_vtg () from /home/tergeist/devel/packages/embsys/labs/gps/ ↩
    lib/libnmea.so
#4  0x0000000000400b35 in write_vtg ()
#5  0x0000000000400d67 in main ()
(gdb)
```

D'après la sortie de GDB, on remarque que le bug provoquant l'erreur de segmentation se trouve dans la fonction **knot_to_kmh_str()** de la librairie **libnmea.so**.

On retrouve le fichier où cette fonction est définie :

```
> cd gps
> grep -r knot_to_kmh_str
Binary file core matches
Binary file lib/libnmea.so matches
src/lib/nmea/nmea.c:int knot_to_kmh_str(float not, size_t size, char * format, char *  ↩
    kmh_str)
src/lib/nmea/nmea.c:    knot_to_kmh_str(vtg->speed_knot, NMEA_SPEED_SIZE, "%05.1f",  ↩
    speed_kmh_str);
Binary file src/lib/nmea/nmea.o matches
```

La fonction est définie dans le fichier **src/lib/nmea/nmea.c** :

```
int knot_to_kmh_str(float not, size_t size, char * format, char * kmh_str)
{
    float kmh = KNOT_TO_KMH * not;
    snprintf(kmh_str, size, format, kmh);

#ifndef GPS_OK
    iteration++;
    if (iteration == 2)
    {
        puts(NULL);
    }
#endif

    return kmh;
}
```

La portion de code fausse est **puts(NULL)**. Il s'agit visiblement d'une erreur volontaire car déclenchée seulement dans le cas où la variable **GPS_OK** n'est pas définie.

Question 4 : *Que se passe-t-il quand vous lancez GDB en mode interactif sur le binaire **gps**? Pourquoi?*

```
> gdb ./bin/gps
GNU gdb (Debian 7.10-1) 7.10
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bin/gps...(no debugging symbols found)...done.
(gdb) r
Starting program: /home/tergeist/devel/packages/embsys/labs/gps/bin/gps
/home/tergeist/devel/packages/embsys/labs/gps/bin/gps: error while loading shared libraries ↩
    : libptmx.so: cannot open shared object file: No such file or directory
[Inferior 1 (process 6811) exited with code 0177]
(gdb)
```

La sortie de GDB nous indique qu'il n'arrive pas à exécuter le simulateur car il lui manque la librairire **libptmx.so**.

Question 5 : *À quoi sert la commande ldd? Quelle information supplémentaire cela vous apporte-t-il?*

D'après la sortie du man :

```
> man ldd
NAME
       ldd - print shared library dependencies

SYNOPSIS
       ldd [option]... file...
...
```

En lançant la commande ldd sur le simulateur :

```
> ldd ./bin/gps
    linux-vdso.so.1 (0x00007ffe40fc8000)
    libptmx.so => not found
    libnmea.so => not found
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc37f1d8000)
    /lib64/ld-linux-x86-64.so.2 (0x0000556c2f378000)
```

En plus du manque de la librairie **libptmx.so**, on voit qu'il manque aussi la librairie partagée **libnmea.so** pour l'exécution du simulateur GPS.

Question 6 : *Comment résoudre ce problème en tant qu'utilisateur? N'hésitez pas à regarder le fichier *labs/gps/run.sh**

Malgré cela, le script **run.sh** est bien capable de lancer le simulateur (voir question 1). Analysons ce script :

```
#! /bin/sh

SCRIPT=`readlink -f $0`
ROOT_DIR=`dirname $SCRIPT`

export LD_LIBRARY_PATH=$ROOT_DIR/lib
$ROOT_DIR/bin/gps
```

On voit que la variable d'environement **LD_LIBRARY_PATH** est utilisée. D'après le **Program Library HOWTO** de Linux :

```
In Linux, the environment variable LD_LIBRARY_PATH is a colon-separated set of
directories where libraries should be searched for first, before the standard
set of directories; this is useful when debugging a new library or using a
nonstandard library for special purposes.
```

Essayons :

```
> cd gps
> ./bin/gps
./bin/gps: error while loading shared libraries: libptmx.so: cannot open shared object file ↩
    : No such file or directory
> ldd ./bin/gps
    linux-vdso.so.1 (0x00007ffdd22c0000)
    libptmx.so => not found
    libnmea.so => not found
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f09b7e8c000)
    /lib64/ld-linux-x86-64.so.2 (0x00005570c993f000)
> export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/lib
> ldd ./bin/gps
    linux-vdso.so.1 (0x00007ffd4bd43000)
    libptmx.so => .../embsys/labs/gps/lib/libptmx.so (0x00007f7bdac4e000)
    libnmea.so => .../embsys/labs/gps/lib/libnmea.so (0x00007f7bdaa4c000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7bda675000)
    libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f7bda374000)
    /lib64/ld-linux-x86-64.so.2 (0x0000564bb9289000)
> ./bin/gps
PTTY: /dev/pts/4
```

Question 7 : *Dans quel contexte ce type d'outils peut être intéressant?*

Gdbserver est intéressant lors de la phase de débugage d'un binaire tournant sur un système embarqué. Pas besoin de travailler avec GDB directement sur la carte.

3 Exercice 2 : LD_PRELOAD et sigaction

Question 8 : *Implémentez dans le fichier hook.c la fonction à l'origine du problème repéré au sein du simulateur GPS mais cette fois-çi sans erreur.*

La fonction provoquant l'erreur de segmentation fault est knot_to_kmh_str. On recopie cette fonction dans le fichier ld_preload/hook.c la fonction sans erreur :

```
#include <stdio.h>

#define NOT_TO_KMH 1.852

int knot_to_kmh_str(float not, size_t size, char * format, char * kmh_str)
{
    float kmh = KNOT_TO_KMH * not;
    snprintf(kmh_str, size, format, kmh);

    return kmh;
}
```

Question 9 : *Éditez le Makefile pour compiler hook.c sous la forme d'une librairie partagée nommée libhook.so (s'inspirer de **labs/gps/src/lib/ptmx/Makefile**). Testez la compilation.*

On édite le Makefile pour avoir ceci :

```
SONAME = libhook.so
GCC = gcc

all:
    $(GCC) -c -fPIC hook.c -o hoock.o
        $(GCC) -shared -Wl,-soname,$(SONAME) -o $(SONAME) hook.o -lm

clean:
        rm -f *.so *.o
```

On compile :

```
> cd src/ld_preload
> ls
hook.c  Makefile  run.sh
> make
gcc -c -fPIC hook.c -o hoock.o
gcc -shared -Wl,-soname,libhook.so -o libhoock.so hoock.o -lm
> ls
hook.c  hook.o libhook.so Makefile  run.sh
```

Question 10 : *Éditez le fichier run.sh pour utiliser LD_PRELOAD au moment de lancer le simulateur et ainsi hooker le binaire avec la librairie libhook.so. Exécutez run.sh : le simulateur ne doit plus partir en segfault.*

On édite le fichier run.sh :

```
#! /bin/sh

SCRIPT=`realpath $0`
ROOT_DIR=`dirname $SCRIPT`/../../../gps

export LD_LIBRARY_PATH=$ROOT_DIR/lib
LD_PRELOAD=$(pwd)/libhook.so $ROOT_DIR/bin/gps
```

Lors de l'exécution du binaire par l'intermédiaire du script run.sh précédent, il n'y a plus de segmentation fault.

Question 11 : *Utilisez le **man** pour déterminer le prototype de la fonction **printf** (expliquez comment vous utilisez **man** dans ce cas et pourquoi). Comment est appelé ce type de fonction?*

D'après le man du man :

```
> man man
...
       The table below shows the section numbers of the manual followed by the types of  ↩
           pages they contain.

       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
       4   Special files (usually found in /dev)
       5   File formats and conventions eg /etc/passwd
       6   Games
       7   Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
       8   System administration commands (usually only for root)
       9   Kernel routines [Non standa]
```

On cherche ici à avoir des informations non pas sur la commande **printf** du shell mais sur la commande printf fournie par la libc. On va donc regarder dans la section 3 du man pour récupérer les informations souhaitées :

```
> man 3 printf
...
int printf(const char *format, ...);
```

C'est ce qu'on appelle une fonction variadique, c'est à dire acceptant un nombre de paramètres variable.

Question 12 : *Hookez le simulateur pour que ce dernier ne puisse plus être interrompu par le signal SIGINT (Ctrl-C) en réimplémentant la fonction **printf** dans libhook.so. Pour cela, utilisez la fonction **sigaction** pour mettre en place un gestionnaire de signaux (s'inspirer de **gps/src/bin/gps/gps.c**).*

On édite le fichier hook.c :

```
#include <stdio.h>
#include <stdarg.h>
#include <signal.h>

#define NOT_TO_KMH 1.852

//------------------------------------------------------------------------------
int knot_to_kmh_str(float not, size_t size, char * format, char * kmh_str)
{
    float kmh = NOT_TO_KMH * not;
    snprintf(kmh_str, size, format, kmh);

    return kmh;
}

//------------------------------------------------------------------------------
void handler(int signal_number)
{
    fprintf(stdout, "NIARK!\n");
}

//------------------------------------------------------------------------------
int printf(const char *format, ...)
{
    struct sigaction action;
    action.sa_handler = handler;
    sigemptyset(& (action.sa_mask));
    action.sa_flags = 0;
    sigaction(SIGINT, & action, NULL);
}
```

Ensuite on compile, on exécute et on essaie d'interrompre le programme avec Ctrl-C (signal SIGINT) :

```
> make
> sh run.sh
^CNIARK!
^CNIARK!
^CNIARK!
^CNIARK!
^CNIARK!
^CNIARK!
```

Question 13 : *Comment faire pour interrompre le processus étant donné que ce dernier ne répond plus au Ctrl-C? Citez deux méthodes.*

On ferme le terminal ou bien on utilise la commande kill :

```
> sh run.sh
^CNIARK!
^CNIARK!
^Z
[1]+  Stopped                 sh run.sh
> bg
[1]+ sh run.sh &
> ps
```

```
  PID TTY          TIME CMD
 4445 pts/3    00:00:01 bash
11433 pts/3    00:00:00 sh
11436 pts/3    00:00:00 gps
11690 pts/3    00:00:00 ps
> kill 11436
run.sh: line 7: 11436 Terminated LD_PRELOAD="$(pwd)/libhack.so" $ROOT_DIR/bin/gps
[1]+  Exit 143                sh run.sh
> ps
  PID TTY          TIME CMD
 4445 pts/3    00:00:01 bash
11747 pts/3    00:00:00 ps
```

4 Exercice 3 : select, fork et pipe

Comme énoncé pendant le TP, vous pouvez recompiler le simulateur ainsi pour ne plus avoir d'erreur de segmentation (et donc sans avoir besoin de passer par LD_PRELOAD) :

```
> cd gps
> make ok
```

Question 14 : *Selon vous, à quoi correspond le champs indiqué par **PTTY**?*

C'est le port virtuel à travers lequel on va pouvoir communiquer avec le simulateur GPS. Ce dernier envoie periodiquement des trames NMEA sur ce port.

En exécutant le code de gps reader :

```
> cd src/gps_reader
> make
gcc util.c reader.c -o gps_reader -I. -I../../../gps/include
> ./gps_reader
Invalid usage: reader -p port_name
> ./gps_reader -p /dev/pts/5
$GPVTG,054.8,T,034.5,M,005.6,010.4,K
$GPGLL,4837.14,N,00741.54,E,232529,A
$GPVTG,054.8,T,034.5,M,005.6,010.4,K
$GPGLL,4837.20,N,00741.60,E,232533,A
...
```

Question 15 : *En regardant le code de reader.c, y a-t-il quelque chose qui vous chagrine?*

Il n'y a pas de gestionnaire de signaux. Dans le cas d'un signal d'interruption comme Ctrl-C, la boucle while est interrompue et on ne va jamais fermer proprement le file descriptor associé au port du simulateur.

Question 16 : *Grâce à des recherches Internet (ou en fouinant dans le code du simulateur), déterminez dans quelle trame et dans quel champs la date est définie.*

L'heure est définie dans la trame GPGLL dans le champs 5.

Question 17 : *Quelles fonctions sont utilisées dans reader.c pour ouvrir/écouter/lire/fermer le port virtuel du simulateur?*

Ouverture du port :

```
int fd = open(port, O_RDWR | O_NOCTTY);
```

Écoute des évènements (mutliplexage) :

```
select(fd+1, &fdset, NULL, NULL, NULL);
```

Lecture des données :

```
int bytes = read (fd, buff, sizeof(buff));
```

Fermeture du port :

```
close(fd);
```

Question 18/19/20/21

Le code :

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <termios.h>
#include <syslog.h>

#include <util.h>

//------------------------------------------------------------------------------
void print_usage()
{
    fprintf(stderr, "Invalid usage: gps_reader -f first_port -s second_port\n");
}

//------------------------------------------------------------------------------
void event(int fd_in, int fd_out, char * buff)
{
    int bytes = read (fd_in, buff, BUFSIZ);

    // printf("serial event for fd=%d (%d): %s\n", fd_in, bytes, buff);

    if (bytes > 0)
    {
        if (is_gll(buff) == 0)
        {
            char time_str[7];
            int time = get_time(buff, time_str);

            write(fd_out, time_str, sizeof(time_str));
        }
    }
}

//------------------------------------------------------------------------------
int father(int fd1, int fd2, int pipe, char * port1, char * port2)
{
    //
    int maxfd = fd1;
    if (fd2 > maxfd)
        maxfd = fd2;

    // read port
    char buff[BUFSIZ];
    fd_set fdset;

    while(1)
    {
        bzero(buff, sizeof(buff));
```

```
        FD_ZERO(&fdset);
        FD_SET(fd1, &fdset);
        FD_SET(fd2, &fdset);

        select(maxfd+1, &fdset, NULL, NULL, NULL);

        if (FD_ISSET(fd1, &fdset))
            event(fd1, pipe, buff);

        if (FD_ISSET(fd2, &fdset))
            event(fd2, pipe, buff);

        fflush(stdout);
    }
}

//---------------------------------------------------------------------------
int child(int fd)
{
    // read port
    char buff[BUFSIZ];
    fd_set fdset;

    while(1)
    {
        bzero(buff, sizeof(buff));

        FD_ZERO(&fdset);
        FD_SET(fd, &fdset);

        select(fd+1, &fdset, NULL, NULL, NULL);

        if (FD_ISSET(fd, &fdset))
        {
            int bytes = read (fd, buff, sizeof(buff));

            // printf("pipe event: %s\n", buff);

            if (bytes == 7)
            {
                char time_str_h[3];
                memcpy(time_str_h, &buff[0], 2);
                time_str_h[2] = '\0';

                char time_str_m[3];
                memcpy(time_str_m, &buff[2], 2);
                time_str_m[2] = '\0';

                char time_str_s[3];
                memcpy(time_str_s, &buff[4], 2);
                time_str_s[2] = '\0';

                char message[BUFSIZ];
                sprintf(message, "Heure: %sh %smin %ssec", time_str_h,
                        time_str_m, time_str_s);

                syslog(LOG_INFO, message);
            }
        }
    }
}
```

```
//------------------------------------------------------------------------------
int main(int argc, char *argv [])
{
    char * port1 = NULL;
    char * port2 = NULL;

    // parse comand line
    if (argc != 5)
    {
        print_usage();
        exit(EXIT_FAILURE);
    }

    char * options = "f:s:";
    int option;
    while((option = getopt(argc, argv, options)) != -1)
    {
        switch(option)
        {
            case 'f':
                port1 = optarg;
                break;

            case 's':
                port2 = optarg;
                break;

            case '?':
                fprintf(stderr, "Invalid option %c\n", optopt);
                exit(EXIT_FAILURE);
        }
    }

    // open serial port
    int fd1 = open(port1, O_RDWR | O_NOCTTY);
    if (fd1 == -1)
    {
        perror("open first port");
        exit(EXIT_FAILURE);
    }
    tcflush(fd1, TCIOFLUSH);

    int fd2 = open(port2, O_RDWR | O_NOCTTY);
    if (fd2 == -1)
    {
        perror("open second port");
        exit(EXIT_FAILURE);
    }
    tcflush(fd2, TCIOFLUSH);

    // open pipe
    int pipe_fds[2];
    if (pipe(pipe_fds) == -1)
    {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // open syslog
    openlog("gps_reader", LOG_PERROR | LOG_PID, LOG_LOCAL0 | LOG_MAIL);

    // fork
```

```
    pid_t child_pid = fork();

    if (child_pid == -1)
    {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    // child
    if (child_pid == 0)
    {
        child(pipe_fds[0]);
    }
    else
    {
        father(fd1, fd2, pipe_fds[1], port1, port2);
    }

    // close serial port
    close(fd1);
    close(fd2);

    // close syslog
    closelog();

    exit(EXIT_SUCCESS);
}
```

Lors de l'exécution :

```
> ./gps_reader -f /dev/pts/5 -s /dev/pts/6
gps_reader[13345]: Heure: 23h 35min 45sec
gps_reader[13345]: Heure: 23h 35min 49sec
gps_reader[13345]: Heure: 23h 35min 53sec
...
```