

Partie 1: Utilisation d'un simulateur GPS

Q1: Makefile : est un fichier spécifiquement conçu pour être utilisé par un programme make.

la commande make permet de compiler le fichier makefile, donc va ainsi compiler tout le projet.

Q2: Le compilateur utilisé ici est gcc.

Q3: Une librairie partagée : est un fichier ayant une extension .so contenant des fonctions.

Q4: soit le fichier test.c

```
#include <stdio.h>
int main(){
    printf("hello world !\n");
    return 0;
}
```

```
gcc -Wall -o test test.c
./test
```

Q5 : gcc -g test.c -o test.o
gcc -g libtest.o -shared test.o

Partie 2: Compilation, debug et gestionnaire de signaux

Exercice 1 : GDB et fichier core

Q1 : Au bout de quelques secondes, on se rend compte que le simulateur par en **segmentation fault**; c'est dû au faite qu'on a tenté d'accéder à un emplacement mémoire qui ne lui était pas alloué.

Q2 : C'est le signal **SIGSEGV** que le processus reçoit pour entrer en segmentation fault. Ce signal est récupéré par détermination de la valeur de retour du binaire. La commande **echo \$?** affiche **139** avec un offset de 128, on tombe sur le signal **SIGSEGV**.

Q3 : Après l'exécution de **gdb**, on se rend compte que l'erreur de segmentation fault est causée par la fonction **knot_to_kmh_str()**, en faisant la commande **grep -r knot_to_kmh_str()**, on se rend compte que l'erreur vient du fichier **libnmea.so** et que la fonction **knot_to_kmh_str()** est définie plus précisément dans le fichier **src/lib/nmea/nmea.c**. En examinant le fichier **nmea.c**, on se rend compte que la partie du code fautive est **puts(NULL)**. Elle s'exécute quand la variable **GPS_OK** n'est pas définie.

Q4 : Lorsqu'on lance **gdb** en mode interactif, le simulateur ne s'exécute pas à cause de l'absence de la librairie **libptmx.so**

Q5: D'après la sortie du **man ldd**, et l'exécution de la commande **ldd ./gps**, on se rend compte qu'il y'a absence de la librairie partagée **libnmea.so**; donc la commande **ldd** liste les librairies absentes lors de l'exécution d'un fichier binaire exécutable.

Q6: Dû au fait que le fichier binaire du gps ne s'exécute pas en mode interactif, mais s'exécute avec **run.sh**, nous avons examiné ce fichier et on se rend compte que la variable d'environnement **LD_LIBRARY_PATH** a été exportée, d'après sa documentation, il s'agit d'un ensemble de répertoires préparés où les librairies doivent être cherchées en premier. La résolution du problème se fait lorsqu'on exporte cette variable d'environnement en ligne de commande.

Q7:

Q8: Cet outil est utilisé lors du débogage d'un binaire sur une carte embarquée au lieu de déboguer directement sur la carte.

Exercice 2 : LD_PRELOAD et sigaction

Q1 : Implémentons dans le fichier **hook.c** la fonction à l'origine du problème
Il s'agira de copier la fonction **knot_to_kmh_str()** dans le fichier **hook.c** du dossier **ld_preload**.

```
#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;
}
```

Q2 : Éditez le Makefile pour compiler **hook.c** sous la forme d'une librairie partagée nommée **libhook.so**
voir **makefile**

SONAME = libhook.so

GCC = gcc

all:

\$(GCC) -g -c -fPIC hook.c -o hook.o

\$(GCC) -g -shared -Wl,-soname,\$(SONAME) -o \$(SONAME) hook.o -lm

clean:

rm -f *.so *.o

Q3: Éditions le fichier run.sh pour utiliser LD_PRELOAD afin de ne plus avoir une segmentation fault

Il 'a plus de segmentation fault lorsqu'on modifie et exécute le fichier **run.sh**

#!/bin/sh

SCRIPT=`readlink -f \$0`

ROOT_DIR=`dirname \$SCRIPT`/../.././gps

export LD_LIBRARY_PATH=\$ROOT_DIR/lib

LD_PRELOAD = \$ (pwd)/libhook.so

\$ROOT_DIR/bin/gps

Q4: Utilisons le man pour déterminer le prototype de la fonction printf

man printf et **man man printf** donne les informations sur la commande printf fournit par le manuel alors que **man 3 printf** donne le shell de la commande **printf**

Q5: Hookons le simulateur pour que ce dernier ne puisse plus être interrompu par le signal SIGINT

voir fichier hook.c

// Question 8 : write here the buggy function without errors

#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 = KNOT_TO_KMH * not;
    snprintf(kmh_str, size, format, kmh);
    return kmh;
}
```

// Question 12 : write printf with a signal handler

//-----

```
void signal_handler(int signal_number)
```

```
{
```

```
    fprintf(stdout, "NIARK!\n");
```

```
}
```

//-----

```
int printf(const char *format, ...)
```

```
{
```

```
    struct sigaction action;
```

```
    action.sa_handler = signal_handler;
```

```
    sigemptyset(& (action.sa_mask));
```

```
    action.sa_flags = 0;
```

```
    sigaction(SIGINT, & action, NULL);
```

```
}
```

Après exécution on se rend compte que le **Ctrl-C** n'interrompt plus le programme

Q6: Citons deux méthodes pour faire interrompre le processus

- Utilisation de la commande **kill**
- fermeture du terminal

PARTIE 3 : Multiplexage, threads, mutex et IPC

Exercice 1 : Multiplexage

Q1 : Le champs indiqué par **PTTY** correspond au nom du port virtuel à travers lequel on va communiquer avec le gps. Il renvoie de façon périodique les trames **NMEA**.

Q2 : On ne trouve pas de gestionnaire de signaux pour catcher les CTRL-C, le CTRL-C arrête la simulation par conséquent la socket n'est jamais fermé, donc son port ne serait plus disponible, si on veut le relancer, il ne marchera pas.

Q3 : L'heure est définie dans :

- Trame: **GPGLL**

- Champs : **5**

Q4 : Les fonctions utilisées dans reader.c

- **Ouvrir :** c'est la fonction open

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

- **Ecouter :** select

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

- **Lire :** read

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

- **fermer :** close

```
close(fd);
```

Q5 : voir fichier reader.c

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

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

```
#include <util.h>
```

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

    // parse comand line
    if (argc != 5)
    {
        fprintf(stderr, "Invalid usage: reader -p port_name -p second_port\n");
        exit(EXIT_FAILURE);
    }

    char * options = "p:s:";
    int option;
    printf("%d", getopt(argc, argv, options));
    while((option = getopt(argc, argv, options)) != -1)
    {
        switch(option)
        {
            case 'p':
                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);


// read port
char buff[50];
fd_set fdset;

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

    FD_ZERO(&fdset);
    FD_ZERO(fd1,&fdset);
    FD_ZERO(fd2,&fdset);
int maxfd=fd1;
if (fd2 > maxfd) maxfd=fd2;

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

    if (FD_ISSET(fd1, &fdset))
    {
        int bytes = read (fd1, buff, sizeof(buff));

        if (bytes > 0)
        {
            printf("%s\n", buff);
            //fflush(stdout);
        }
    }

    if (FD_ISSET(fd1, &fdset))
    {

```

```

        int bytes = read (fd1, buff, sizeof(buff));

        if (bytes > 0)
        {
            printf("%s\n", buff);
            //fflush(stdout);
        }
    }

    fflush(stdout);

}

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

exit(EXIT_SUCCESS);
}

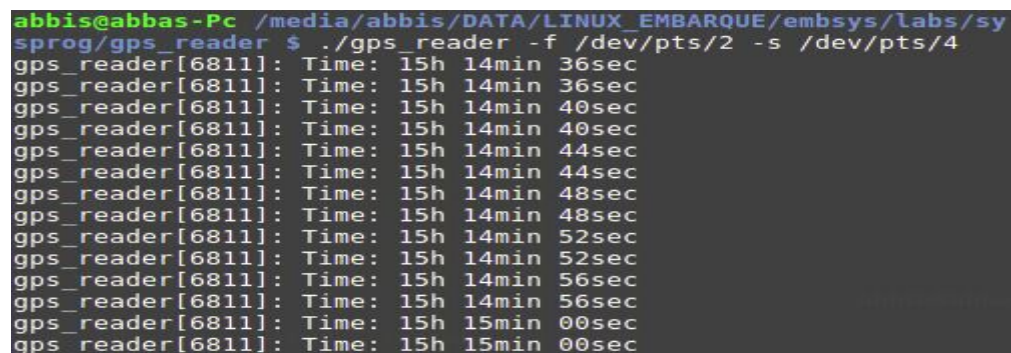
```

Q6

:

voir

fichier



```

abbis@abbas-Pc /media/abbis/DATA/LINUX_EMBARQUE/embsys/labs/sy
sprog/gps_reader $ ./gps_reader -f /dev/pts/2 -s /dev/pts/4
gps_reader[6811]: Time: 15h 14min 36sec
gps_reader[6811]: Time: 15h 14min 36sec
gps_reader[6811]: Time: 15h 14min 40sec
gps_reader[6811]: Time: 15h 14min 40sec
gps_reader[6811]: Time: 15h 14min 44sec
gps_reader[6811]: Time: 15h 14min 44sec
gps_reader[6811]: Time: 15h 14min 48sec
gps_reader[6811]: Time: 15h 14min 48sec
gps_reader[6811]: Time: 15h 14min 52sec
gps_reader[6811]: Time: 15h 14min 52sec
gps_reader[6811]: Time: 15h 14min 56sec
gps_reader[6811]: Time: 15h 14min 56sec
gps_reader[6811]: Time: 15h 15min 00sec
gps_reader[6811]: Time: 15h 15min 00sec

```

Exercice 2 : Mémoire partagée et sémaphore

Q1: **myshm** correspond au nom de la mémoire partagée et **lock** correspond au nom du sémaphore (variable qui permet d'accéder à la mémoire partagée)

Q2: l'emplacement des segments de mémoire partagées est : /dev/shm

La commande **ipcs -m** permet de lister ces segments de système

Q4: Décrivons les fonctions utilisées pour gérer le segment de mémoire partagée.

- **sem_open** permet l'ouverture du sémaphore
- **open** ouverture du port permettant la connexion avec le gps
- **shm_open** ouverture de la mémoire partagée
- **mmap** créer une adresse virtuelle

Q5: C'est la fonction **shm_open** qui utilise le paramètre **myshm** passé en ligne de commande.

Q6: le flag **O_RDWR|O_CREAT** indique la creation de segment (s'il n'existe pas) et l'ouverture en **WR**

Q7:

```
handlers->shm = opts.shm;
handlers->shmfd = shm_open(opts.shm,O_RDWR,S_IRUSR);
if (handlers->shmfd == -1)
{
    perror("shm_open");
    goto err;
}
if (ftruncate(handlers->shmfd, sizeof(handlers->shdata)) != 0)
{
    perror("ftruncate");
    goto err;
}
handlers->shdata = mmap(NULL, sizeof(handlers->shdata),
PROT_READ|PROT_WRITE, MAP_SHARED, handlers->shmfd, 0);
if (handlers->shdata == MAP_FAILED)
{
    perror("mmap");
    goto err;
}
return 0;
```

Q8:

```
GCC = gcc
BIN = shm_reader
all:
    $(GCC) opts.c handler.c shm_reader.c -o $(BIN) \
    -I. -I../../gps/include -lrt -lpthread
clean:
    rm -f $(BIN)
```

Q9: La valeur **time** est modifiée à chaque fois qu'une trame est lu par le writer, puis modifie la valeur des différents champs de la mémoire partagée

Q10: la particularité d'une variable globale est qu'on peut l'utiliser dans toutes les fonctions , et pour la déclarer on doit en mettre dehors de toutes les fonctions

Q11:

```
void *shmreader()
{
    while(1)
    {
        if(handlers.shdata != NULL)
        {
```



```

        printf("\n");
        printf("time: %d\n", handlers.shdata->time);
        printf("longitude: %d\n", handlers.shdata->longitude);
        printf("latitude: %d\n", handlers.shdata->latitude);
        fflush(stdout);
    }
    usleep(500000);
}
}

```

Q13: A côté de **shm_reader** on ne voit plus le temps actuel , le temps affiché est 0 par conséquent le sémaphore était initialisé avec une valeur nulle

Q14:

Q15:

Exercice 3 : thread et mutex