SKPS-LAB3
Sprawozdanie

Matvii Ivashchenko
Katsyaryna Anikevich

1 Pierwszy pakiet

Zgodnie z wykładem, pobraliśmy i rozpakowaliśmy SDK dla Raspberry Pi.
Dodaliśmy poleceniem

```
nano feeds.conf.default
```

ścieżkę do pakietu demo1:

```
src-link skps /home/student-pl/WZ_W03_przyklady/demo1_owrt_pkg
```

Zaktualizowaliśmy listę pakietów i zainstalowaliśmy pakiet skps poleceniami

```
scripts/feeds update -a
scripts/feeds install -p skps -a
```

Wywołaliśmy make menuconfig i zaznaczyliśmy dodane pakiety w menu Examples: **demo1, demo1mak**
Zbudowaliśmy paczkę poleceniem:

```
make package/demo1/compile
```

Przekazaliśmy skompilowany plik demo1 do RPi4

```
scp /home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64/bin/
    packages/aarch64_cortex-a72/skps/demo1_1.0-r1_aarch64_cortex-a72.ipk root@10
    .42.0.248:/root/
```

Zainstalowaliśmy pakiet demo1 poleceniem :

```
opkg install demo1_1.0-r1_aarch64_cortex-a72.ipk
```

Uruchomiliśmy zainstalowany pakiet demo1:

```
root@OpenWrt:~# demo1
dzien dobry
Komunikat z wątku A
Komunikat z wątku B
Komunikat z wątku B
Komunikat z wątku A
Komunikat z wątku B
```

2 Pakiety "worms"i "buggy"

Skopiowaliśmy foldery worms i buggy do folderu

```
/home/user/Pobrane/demo/demo1_owrt_pkg
```

oraz przygotowaliśmy pliki Makefile dla pobranych pakietów.

Zaktualizowaliśmy listę pakietów poleceniem

```
./scripts/feeds update -a.
```

Następnie zainstalowaliśmy pakiet ncurses w SDK oraz feed skps poleceniami:

```
./scripts/feeds install libncurses
./scripts/feeds install -p skps -a.
```

Po wywołaniu make menuconfig zaznaczyliśmy dodane pakiety w menu Examples: **buggy, worms.**
Zbudowaliśmy paczki poleceniami:

```
make package/feeds/skps/buggy/compile
make package/feeds/skps/worms/compile
```

Po urochomieniu **make menuconfig** w pozycji Global Build Settings wyłączyliśmy opcje: Select all target specific packages by default, Select all kernel module packages by default, Select all userspace packages by default, Cryptographically sign package lists i wybraliśmy pakiety buggy, worms.
Przekazaliśmy skompilowane paczki do RPi4:

```
student-pl@lab-44:~/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64$ scp
/home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64/bin/
    packages
/aarch64_cortex-a72/skps/buggy_1.0-r1_aarch64_cortex-a72.ipk  root@10.42.0.248:/root/
buggy_1.0-r1_aarch64_cortex-a72.ipk           100% 3340     2.7MB/s   00:00

student-pl@lab-44:~/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64$ scp
/home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64/bin/
    packages
/aarch64_cortex-a72/skps/worms_1.0-r1_aarch64_cortex-a72.ipk   root@10.42.0.248:/root/
worms_1.0-r1_aarch64_cortex-a72.ipk           100% 3985     2.7MB/s   00:00
```

Zainstalowaliśmy paczki odpowiednio poleceniami:

```
root@OpenWrt:~# opkg install worms_1.0-r1_aarch64_cortex-a72.ipk
Installing worms (1.0-r1) to root...
Configuring worms.
root@OpenWrt:~# opkg install buggy_1.0-r1_aarch64_cortex-a72.ipk
Installing buggy (1.0-r1) to root...
Configuring buggy.
```

Przetestowaliśmy aplikację worms (po wywołaniu komendy worms program się uruchamia), natomiast aplikacje bug1 oraz bug2 zwracają Segmentation fault, a bug3 dwa ciągi znaków.

```
root@OpenWrt:~# bug1
Segmentation fault
root@OpenWrt:~# bug2
Segmentation fault
root@OpenWrt:~# bug3
s1=@ABCDEFGHIJKLMNOPQRSTUVW
s2=JKLMNOPQRSTUVW
```

3 Debugowanie zdalne

Zaktualizowaliśmy listę pakietów i zainstalowaliśmy gdb oraz gdbserver:

```
opkg update
opkg install gdb gdbserver
```

bug1

Uruchomiliśmy gdbserver z poziomu OpenWrt:

```
root@OpenWrt:~# gdbserver :1234 /usr/bin/bug1
gdbserver: Unable to determine the number of hardware watchpoints available.
```

Połączenie z gdbserver z komputera hosta:

```
./scripts/remote-gdb 10.42.0.248:1234 ./build_dir/target-aarch64_cortex-a72_musl/buggy
    -1.0/bug1
```

W GDB ustawiliśmy ścieżki do plików źródłowych i plików debugowania:

```
(gdb) directory ~/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64/
    build_dir/target-aarch64_cortex-a72_musl/
Source directories searched:
/home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64/
    build_dir/target-aarch64_cortex-a72_musl:
/home/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-x86_64
/build_dir/target-aarch64_cortex-a72_musl:$cdir:$cwd

(gdb) set debug-file-directory ~/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux
    -x86_64
/build_dir/target-aarch64_cortex-a72_musl/buggy-1.0/
```

Sprawdzili kod z pomocą polecenia **list**, już to problem był znaleziony

```
(gdb) list bug1.c:10
5 {
6   int i;
7   //"Zapomniana" inicjalizacja wska nika "table"
8   //table=(int*) malloc(1000*sizeof(int));
9   for(i=0;i<1000;i++) {
10      table[i]=i;
11    }
12  }
(gdb)
```

Umieściliśmy punkt przerwania w pliku bug1.c w linii 10 i uruchomiliśmy program:

```
(gdb)break bug1.c:10
(gdb)run

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
```

Wskaźnik table został użyty bez wcześniejszej inicjalizacji – linia odpowiedzialna za przydzielenie pamięci dynamicznej (malloc) była zakomentowana.

bug2

Uruchomiliśmy gdbserver z poziomu OpenWrt i łączymyśmy z gdbserver z komputera hosta. W GDB ustawiliśmy ścieżki do plików źródłowych i plików debugowania. Też ustawiliśmy punkt przerwania (breakpoint). Jednak to nie dalo rozumienia w jakim momencie zdarza się błęd. Dla tego breakpoint został usunięty i zdecydowaliśmy sprawdzic wartosć i w moment błędu z pomocą polecenia **display i** i dostaliśmy wynik:

```
Program received signal SIGSEGV, Segmentation fault.
main () at buggy-1.0/bug2.c:8
8     table[i]=i;
1: i = 1008
2: i = 1008
(gdb)
```

W 1008 iteracji program spadł z błędem segmentacji — SIGSEGV) - wyszliśmy poza przydzieloną tablicę (maksymalny dozwolony indeks: 999).

Program próbuje czytać wartości spoza zakresu tablicy. Tablica została zadeklarowana z rozmiarem 1000, jednak pętla próbuje odczytywać wartości z indeksu 1000000, co prowadzi do wyjścia poza jej granice. Poniżej znajduje sie proba uzucia breakpoint orarz step:

```
Breakpoint 1, main () at buggy-1.0/bug2.c:8
8     table[i]=i;
(gdb) step
7   for(i=0;i<1000000;i++) {
```

bug3

Uruchomiliśmy gdbserver z poziomu OpenWrt i łączymyśmy z gdbserver z komputera hosta. W GDB ustawiliśmy ścieżki do plików źródłowych i plików debugowania. Też ustawiliśmy punkt przerwania (breakpoint). Włączyłismy systemowe watchpoin:

```
set breakpoint auto-hw off
set can-use-hw-watchpoints 0
```

```
(gdb) list bug3.c:10
5 /*
6 Ten program pokazuje, że błędy dostępu do pamięci nie zawsze są wykrywane przez
7 mechanizmy ochrony i mogą doprowadzić do błędnego działania aplikacji.
8 */
9 int main()
10 {
11   int i;
12   for(i=0;i<24;i++) {
```

```
13      s1[i]=i+64;
14    }
(gdb)
```

W celu zaobserwowania zapisu poza zakresem, ustawiliśmy watchpoint jednak program się zawiesił, znaleźliśmy rozwiązanie - najpierw tworzymy breakpoint, a z jednym z kroków watchpoint:

```
   Breakpoint 9 at 0x400500: file buggy-1.0/bug3.c, line 10.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-
    x86_64/build_dir/target-aarch64_cortex-a72_musl/buggy-1.0/bug3

Breakpoint 9, main () at buggy-1.0/bug3.c:12
12    for(i=0;i<24;i++) {
(gdb) display i
1: i = 0
(gdb) next
13      s1[i]=i+64;
1: i = 0
(gdb) next
12    for(i=0;i<24;i++) {
1: i = 1
(gdb) next
.
.
.
(gdb) next
13      s1[i]=i+64;
1: i = 7
(gdb) watch s1[10]
Watchpoint 10: s1[10]
(gdb) next
.
.
.
(gdb) next
12    for(i=0;i<24;i++) {
1: i = 10
(gdb) next
13      s1[i]=i+64;
1: i = 10
(gdb) next

Watchpoint 10: s1[10]

Old value = 97 'a'
New value = 74 'J'
main () at buggy-1.0/bug3.c:12
12    for(i=0;i<24;i++) {
1: i = 11
(gdb)
```

Gdy ‘i = 10‘, program nadpisał wartość w ‘s1[10]‘, mimo że tablica miała tylko 10 elementów wychodząc poza pamięć zarezerwowaną dla table.

4 Wykorzystanie poleceń:

Ustawienie breakpointu,praca krokowa(step), podgląd wartości zmiennej jednorazowy:

```
(gdb) break bug2.c:8
Breakpoint 1 at 0x4004c4: file buggy-1.0/bug2.c, line 8.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-
    x86_64/build_dir/target-aarch64_cortex-a72_musl/buggy-1.0/bug2
Breakpoint 1, main () at buggy-1.0/bug2.c:8
8       table[i]=i;
(gdb) step
7     for(i=0;i<1000000;i++) {
```

```
(gdb) print table
$1 = {0 <repeats 1000 times>}
(gdb)
```

Podgląd wartości zmiennej przy każdym kroku, praca krokowa(next)

```
(gdb) display i
1: i = 0
(gdb) run
The program being debugged has been started already.
tart it from the beginning? (y or n) y
Starting program: /home/student-pl/openwrt-sdk-24.10.0-bcm27xx-bcm2711_gcc-13.3.0_musl.Linux-
    x86_64/build_dir/target-aarch64_cortex-a72_musl/buggy-1.0/bug1
Breakpoint 1, main () at buggy-1.0/bug1.c:10
10      table[i]=i;
1: i = 0
(gdb) next

Program received signal SIGSEGV, Segmentation fault.
main () at buggy-1.0/bug1.c:10
10      table[i]=i;
1: i = 0
(gdb) next

Program terminated with signal SIGSEGV, Segmentation fault.
The program no longer exists.
(gdb)
(gdb) monitor exit
```

Podgląd stosu:

```
(gdb) info stack
#0  main () at buggy-1.0/bug2.c:8
(gdb) x/20x $sp
0x7fffffd60: 0x00000000 0x00000000 0x00000000 0x00000000
0x7fffffd70: 0xfffffd90 0x0000007f 0xf7ffec30 0x0000007f
0x7fffffd80: 0xf7fff078 0x0000007f 0xf7ffd7a8 0x0000007f
0x7fffffd90: 0x00000001 0x00000000 0xffffff68 0x0000007f
0x7fffffda0: 0x00000000 0x00000000 0xffffff76 0x0000007f
(gdb)
```

Backtrace:

```
(gdb) backtrace
#0  main () at buggy-1.0/bug2.c:8
```

Watchpoint już był zademonstrowany powyżej w bug3.