

# Sprawozdanie z ćwiczenia nr 3

Mateusz Okulus, Mariusz Pakulski

## Cel ćwiczenia

Celem laboratorium było zapoznanie się z tworzeniem pakietów dla OpenWRT za pomocą SDK oraz debugowanie aplikacji za pomocą gdb.

## Połączenie RaspberryPI

RaspberryPI łączymy tak jak w pierwszym ćwiczeniu.

## Pierwszy pakiet

Pobieramy i rozpakowujemy pakiet demo1. Pobieramy SDK z:

```
https://downloads.openwrt.org/releases/22.03.3  
/targets/bcm27xx/bcm2711  
/openwrt-sdk-22.03.3-bcm27xx-bcm2711  
_gcc-11.2.0_musl.Linux-x86_64.tar.xz
```

Wywołujemy `make config` i w celu przyspieszenia pracy wyłączamy następujące opcje w Global Build Settings:

- Select all target specific packages by default
- Select all kernel module packages by default
- Select all userspace packages by default
- Cryptographically sign package lists

Do pliku `feeds.conf.default` dodajemy linię:

```
src-link skps /home/user/demo1_owrt_pkg
```

Aktualizujemy repozytorium:

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

i instalujemy pakiety:

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

Pakiet dodatkowo wybieramy w `make menuconfig` w sekcji `Examples`.

Pakiet budujemy za pomocą:

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

Uruchamiamy serwer http:

```
python3 -m http.server
```

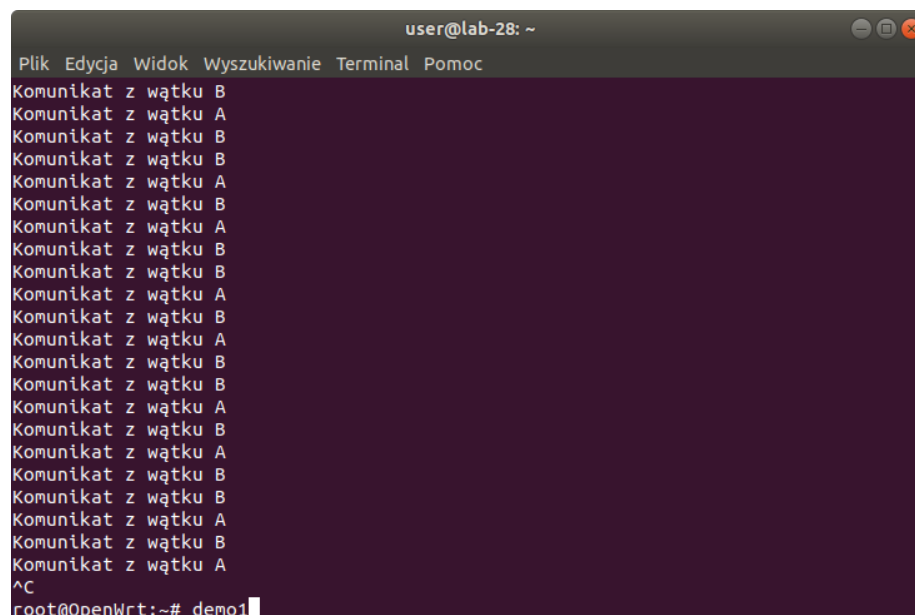
i pobieramy zbudowany pakiet z

```
bin/packages/aarch64_cortex-a72/skps/demo1_1.0-1_aarch64_cortex-a72.ipk
```

Po pobraniu instalujemy pakiet

```
opkg install demo1_1.0-1_aarch64_cortex-a72.ipk.
```

Program uruchamiamy poprzez wywołanie `demo1`.



```
user@lab-28: ~  
Plik Edycja Widok Wyszukiwanie Terminal Pomoc  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku A  
Komunikat z wątku B  
Komunikat z wątku A  
^C  
root@OpenWrt:~# demo1
```

## Pakiety worms i buggy

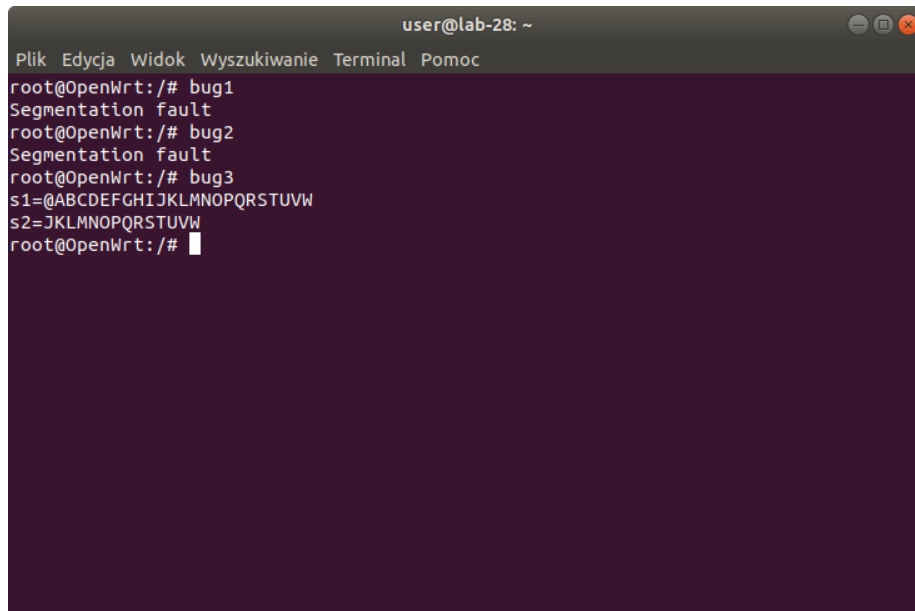
Pakiety pobieramy z pliku `WZ_W03_przyklad_extbr.tar.xz` na Moodle. Interesują nas głównie pliki źródłowe. Plik `Makefile` z definicją pakietu wzorujemy na pliku z pakietu `demo1`. Dla pakietu `buggy` zmieniamy `PKG_NAME`, `TITLE` i sekcję `Build/Compile`. Dla pakietu `worms` postępujemy podobnie, ale w sekcji `Package` dodajemy jeszcze `DEPENDS:+=libncurses`, ponieważ pakiet `worms` korzysta z tej biblioteki.

Ze względu na rozmiary plików `Makefile` zdecydowaliśmy nie umieszczać ich bezpośrednio w sprawozdaniu, są one dostępne w repozytorium, w folderze `demo1_owrt_pkg`.

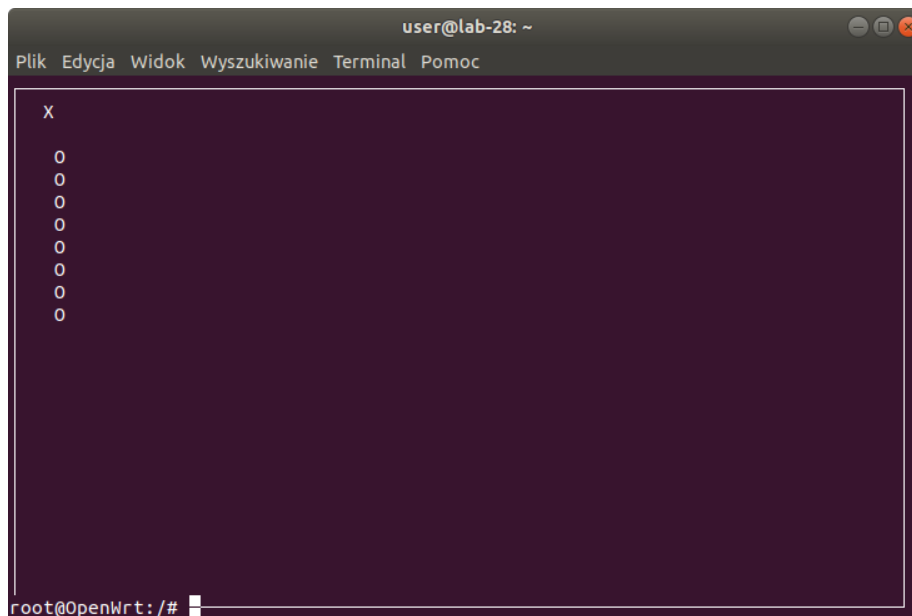
Ponownie wywołujemy

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

i zaznaczamy pakiety w sekcji **Examples**. Pakiety budujemy poprzez wywołanie `make package/buggy/compile` i `make package/worms/compile`. Znowu uruchamiamy serwer HTTP i w OpenWRT pobieramy je i instalujemy za pomocą `opkg install <nazwa.ipk>`.



```
user@lab-28: ~  
Plik Edycja Widok Wyszukiwanie Terminal Pomoc  
root@OpenWrt:/# bug1  
Segmentation fault  
root@OpenWrt:/# bug2  
Segmentation fault  
root@OpenWrt:/# bug3  
s1=@ABCDEFGHJKLMNOPQRSTUVWXYZ  
s2=JKLMNOPQRSTUVWXYZ  
root@OpenWrt:/#
```



## Debugowanie zdalne

Na RaspberryPI instalujemy gdb i gdbserver poprzez:

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

Uruchamiamy serwer na RPi za pomocą `gdbserver :9000 /usr/bin/bug1`. Na hoście uruchamiamy

```
./scripts/remote-gdb 10.42.0.227:9000 \
./staging_dir/target-*/root-*/usr/bin/bug1
```

Adres IP 10.42.0.227 uzyskujemy wywołując `ifconfig` w OpenWRT.

## Debugowanie bug1

Pakiet przebudowujemy z symbolami do debugowania

```
make package/buggy/{clean,compile} CONFIG_DEBUG=y
```

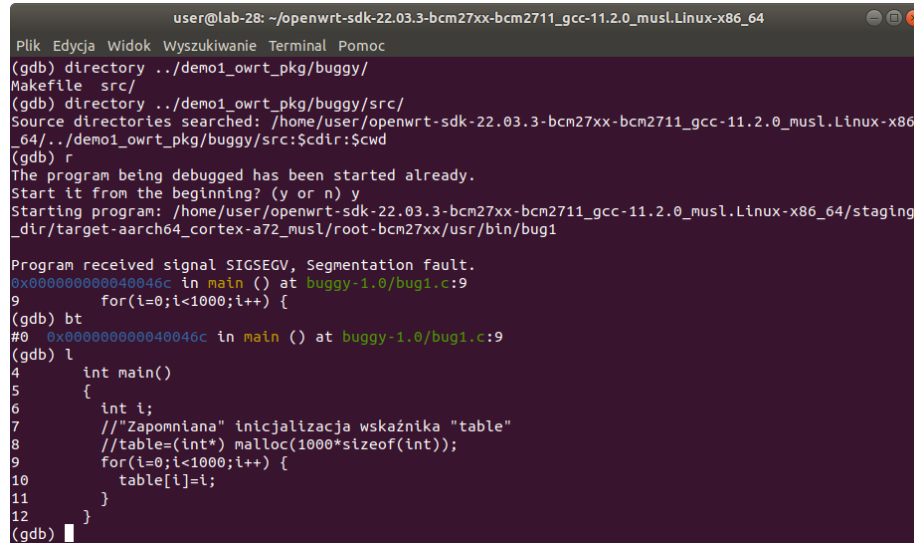
i instalujemy go ponownie na OpenWRT poprzez

```
opkg install --force-reinstall
```

Na hoście, w gdb wskazujemy na pliki źródłowe poprzez

```
directory ../demo1_owrt_pkg/buggy/src/
```

Uruchamiamy program za pomocą `r` (`run`). Następuje błąd segmentacji w pętli `for`. `bt` (`backtrace`) pokazuje, że jesteśmy w `main`. `l` wyświetla kod źródłowy i zauważamy, że tablica nie jest zainicjowana.



```
user@lab-28: ~/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64
Plik Edycja Widok Wyszukiwanie Terminal Pomoc
(gdb) directory ../demo1_owrt_pkg/buggy/
Makefile src/
(gdb) directory ../demo1_owrt_pkg/buggy/src/
Source directories searched: /home/user/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64/./demo1_owrt_pkg/buggy/src:$cd:$cd
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64/staging_dir/target-aarch64_cortex-a72_musl/root-bcm27xx/usr/bin/bug1
Program received signal SIGSEGV, Segmentation fault.
0x0000000040046c in main () at buggy-1.0/bug1.c:9
9   for(i=0;i<1000;i++) {
(gdb) bt
#0  0x0000000040046c in main () at buggy-1.0/bug1.c:9
(gdb) l
4   int main()
5   {
6       int i;
7       // "Zapomniana" inicjalizacja wskaźnika "table"
8       // table=(int*) malloc(1000*sizeof(int));
9       for(i=0;i<1000;i++) {
10          table[i]=i;
11      }
12  }
```

## Debugowanie bug2

Zamykamy sesję na OpenWRT poprzez monitor `exit` w `gdb`. Na OpenWRT uruchamiamy `gdbserver :9000 /usr/bin/bug2`. Na hoście podobnie uruchamiamy `./scripts/remote-gdb` podając tym razem ścieżkę do `bug2`. W `gdb` ponownie uruchamiamy `directory ../demo1_owrt_pkg/buggy/src/`. Uruchamiamy program poprzez `r`. Następuje błąd segmentacji przy przypisywaniu do tablicy. Poprzez `l` oglądamy kod. Poprzez `p` i wypisujemy wartość zmiennej `i`, która jest większa od rozmiaru tablicy.

```
user@lab-28: ~/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64
Plik Edycja Widok Wyszukiwanie Terminal Pomoc

Program received signal SIGSEGV, Segmentation fault.
main () at buggy-1.0/bug2.c:8
8      table[i]=i;
(gdb) bt
#0  main () at buggy-1.0/bug2.c:8
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/user/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64/staging
_dir/target-aarch64_cortex-a72_musl/root-bcm27xx/usr/bin/bug2

Program received signal SIGSEGV, Segmentation fault.
main () at buggy-1.0/bug2.c:8
8      table[i]=i;
(gdb) l
3      int main()
4      {
5          int i;
6          //Próba wyjścia poza obszar tablicy
7          for(i=0;i<1000000;i++) {
8              table[i]=i;
9          }
10     }
(gdb) p i
$1 = 1008
(gdb)
```

Błąd segmentacji występuje dopiero dla  $i = 1008$ , a nie  $i = 1000$ , jak można by się tego spodziewać. Po powiększeniu tablicy do 2000 błąd występuje przy  $i = 2032$ . Rozmiar strony jest standardowo równy 4K, czyli 4096 bajtów. `int` ma rozmiar 4 bajtów, więc w stronie mieszczą się 1024 `int`'y.  $1024 - 1008 = 16$ , a  $2048 - 2032 = 16$ . 16 `int`'ów to 64 bajty. Ponieważ tablica `table` jest statyczna to będzie w sekcji `bss`. Możemy zobaczyć znajduje się w sekcji `.bss` i `.data` poprzez:

```
objdump -D \
staging_dir/target-*/root-*/usr/bin/bug2 | \
sed -n '/section .data/,/section .comment/p'
```

Disassembly of section `.data`:

```
0000000000411000 <__dso_handle>:
...
```

Disassembly of section `.bss`:

```
0000000000411008 <completed.1>:
...
```

```
0000000000411010 <object.0>:
...
```

```
0000000000411040 <table>:
...
```

Disassembly of section `.comment`:

Począwszy od adresu 411000 znajdują się obiekty `<__dso_handle>` (8 bajtów), `<completed.1>` (8 bajtów), `<object.0>` ( $3 \cdot 16$  bajtów = 48 bajtów). Daje to razem brakujące 64 bajty - `<table>` zaczyna się od 411040, czyli jest właśnie przesunięte o  $4 \cdot 16 = 64$  bajty. Dopiero wyjście poza stronę generuje błąd segmentacji.

### Debugowanie bug3

Konfiguracja jest analogiczna. Wyłączamy watchpointy sprzętowe poprzez `set breakpoint auto-hw off` i `set can-use-hw-watchpoints 0`. Ponieważ watchpointy programowe są dosyć wolne, zanim go włączymy ustawiamy zwykły breakpoint na `main` i uruchamiamy program. Dzięki temu debugger nie sprawdza odczytów pamięci przy ładowaniu programu i watchpoint nie spowalnia programu. Pomocniczo ustawiamy `display s1[i]`. Ustawiamy watchpoint `watch s1[10]` na bajt po stringu `s1`. Zapoznajemy się z działaniem `s (step)`, `info frame` i `info locals`. Kontynuujemy program poprzez `c`. Zauważamy, że wartość `a` (ze stringa `s2`) została nadpisana.

```
user@lab-28: ~/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64
Plik Edycja Widok Wyszukiwanie Terminal Pomoc
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./staging_dir/target-aarch64_cortex-a72_musl/root-bcm27xx/usr/bin/bug3...
(gdb) directory ../demo1_owrt_pkg/buggy/src/
Source directories searched: /home/user/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64/./demo1_owrt_pkg/buggy/src:$cd:$cd
(gdb) set breakpoint auto-hw off
(gdb) set can-use-hw-watchpoints 0
(gdb) br main
Breakpoint 1 at 0x4004b0: file buggy-1.0/bug3.c, line 12.
(gdb) r
Starting program: /home/user/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64/staging_dir/target-aarch64_cortex-a72_musl/root-bcm27xx/usr/bin/bug3

Breakpoint 1, main () at buggy-1.0/bug3.c:12
12      for(i=0;i<24;i++) {
(gdb) p i
$1 = 0
(gdb) display s1[i]
1: s1[i] = 48 '0'
(gdb) watch s1[10]
Watchpoint 2: s1[10]
(gdb) s
13      s1[i]=i+64;
1: s1[i] = 48 '0'
(gdb) s
12      for(i=0;i<24;i++) {
1: s1[i] = 49 '1'
(gdb) bt
#0  main () at buggy-1.0/bug3.c:12
(gdb) info frame
Stack level 0, frame at 0x7fffffd80:
pc = 0x4004cc in main (buggy-1.0/bug3.c:12); saved pc = 0x7fff7f93190
source language c.
Argvlist at 0x7fffffd70, args:
Locals at 0x7fffffd70, Previous frame's sp is 0x7fffffd80
Saved registers:
x29 at 0x7fffffd70, x30 at 0x7fffffd78
(gdb) info locals
i = 1
(gdb) c
Continuing.

Watchpoint 2: 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: s1[i] = 98 'b'
(gdb)
```

## Podsumowanie

Udało nam się wykonać wszystkie wymagane zadania. Nauczyliśmy się tworzyć pakiety dla systemu OpenWRT. Zapoznaliśmy się z wieloma poleceniami gdb - breakpointami, watchpointami, backtracem, display, info. Nauczyliśmy się debugować proste przypadki w których występuje błąd segmentacji, jak i takie, w których niepoprawny dostęp do pamięci nie jest od razu widoczny.