

# Sprawozdanie laboratorium 3

Marcin Jarczewski 330234

Bartosz Jasiński 318777

## Cel i przebieg laboratorium

Celem laboratorium jest wykonanie trzech zadań dotyczących pracy z OpenWRT. Ponadto pogłębienie wiedzy :

- dotyczącej SDK
- budowania własnych pakietów (za pomocą SDK)
- debugowania pakietu (zdalne, przez gdb)

## Zadanie 1:

1. Pobraliśmy SDK a następnie w pliku `feeds.conf.default` dodaliśmy ścieżkę do folderu z pakietami
2. Skonfigurowaliśmy SDK aktualizując listę modułów a następnie skompilowaliśmy moduły `demo1` oraz `demo1mak` odpowiednio zaznaczając które pakiety mają zostać skompilowane w sekcji `Examples` dostępnej po wywołaniu `make menuconfig`
3. Na koniec wgraliśmy moduły na RPi i uruchomiliśmy je.

Poniżej załączamy przykład dla pakietu `demo1` , analogicznie dla pakietu `demo1mak`

```
# w pliku feeds.conf.default
src-link skps /home/user/Pobrane/WZ_W03_Przyklady/demo1_owrt_pkg

export LANG=C
scripts/feeds update -a
scripts/feeds install -p skps -a
```

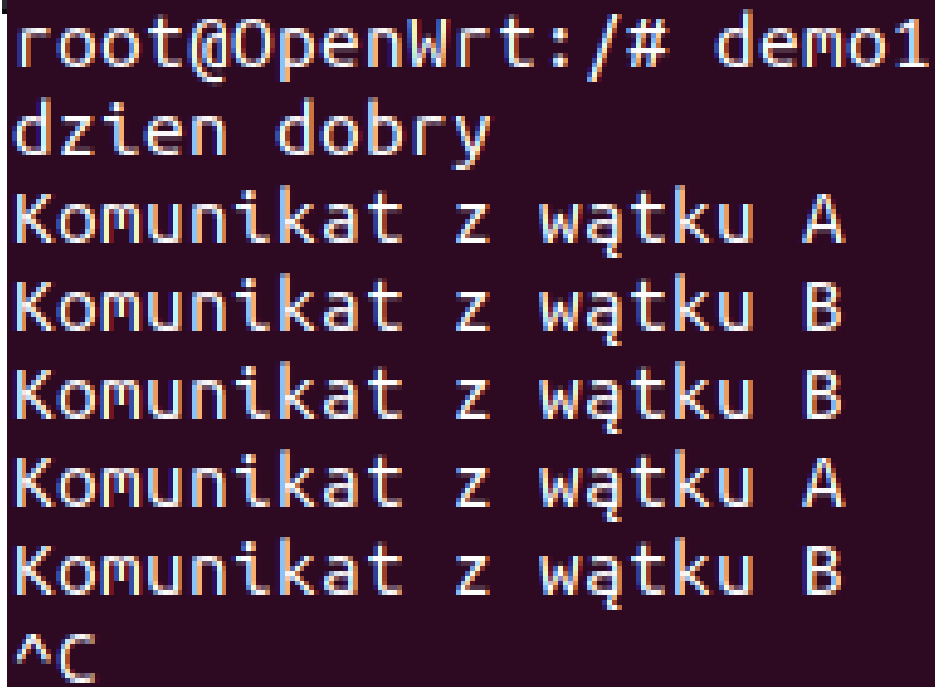
```
make menuconfig

make package/demo1/compile

# przesłanie pliku na RPi z wykorzystaniem wget

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

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
^C
```

## Zadanie 2:

1. Pobraliśmy i rozpakowaliśmy przygotowaną na cele zadania paczkę. Dodaliśmy je do folderu `/home/user/Pobrane/WZ_W03_Przyklady/demo1_owrt_pkg`
2. Do pakietu *worms* należało dodać zależność `libncurses`, którą również trzeba było doinstalować ręcznie w SDK za pomocą komendy `./scripts/feeds install libncurses` oraz zaktualizować listę pakietów poleceniem `scripts/feeds update -a`
3. Utworzyliśmy pliki `Makefile` odpowiednio dla pakietu *worms* i *buggy* umieszczając je w katalogu nadrzędnym wraz z kodem źródłowym danego pakietu oraz zainstalowaliśmy te pakiety poleceniem `scripts/feeds install -p skps -a`
4. W ustawieniach, wywołanych poleceniem `make menuconfig` należało zaznaczyć kompilację obu pakietów w opcji `Examples`

5. zbudowaliśmy pakiety poleceniami `make package/feeds/skps/worms/compile` oraz `make package/feeds/skps/buggy/compile`
6. Na koniec przesłaliśmy pakiety wykorzystując polecenie `wget` oraz zainstalowaliśmy je poleceniem `opkg install <nazwa pakietu>`

Poniżej zamieściliśmy użyte przez nas polecenia oraz zrzuty ekranu z uruchomionymi programami.

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

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

# przesłanie plików
opkg install worms_1.0-1_aarch64_cortex-a72.ipk
opkg install buggy_1.0-1_aarch64_cortex-a72.ipk

# uruchomienie
worms
bug1
bug2
bug3
```

```
0
0 x
0
0
0
0
00000

You hit a wall!
Your score is 152
root@OpenWrt:/#
```

```
root@OpenWrt:/# bug1
Segmentation fault
root@OpenWrt:/# bug2
Segmentation fault
root@OpenWrt:/# bug3
s1=@ABCDEFGHJKLMNOPQRSTUVW
s2=JKLMNOPQRSTUVW
root@OpenWrt:/#
```

## Zadanie 3:

Zainstalowaliśmy pakiety `gdb` oraz `gdbserver` wywołując polecenia:

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

Następnie uruchomiliśmy na RPi gdbserver używając komendy `gdbserver :9000 /usr/bin/bug1` oraz połączyliśmy się z komputera do gdbserver `./scripts/remote-gdb 10.42.0.188:9000 ./build_dir/target-aarch64_cortex-a72_musl/buggy-1.0/bug1`

Aby dodać katalog ze źródłami, wywołaliśmy komendę: `directory /home/user/Pobrane/openwrt-sdk-22.03.3-bcm27xx-bcm2711_gcc-11.2.0_musl.Linux-x86_64/build_dir/target-aarch64_cortex-a72_musl/`

Analogicznie postąpiliśmy w przypadku programów `bug2` i `bug3`. Aby przełączyć się między programami na hoście używaliśmy polecenia `monitor exit`.

## Znalezione błędy

- bug1 - niezainicjalizowana tablica. Wskaźnik `table` ma wartość NULL wobec czego próba jego dereferencji rzuca błąd.

```

Program received signal SIGSEGV, Segmentation fault.
0x000000000040046c in main () at buggy-1.0/bug1.c:9
9       for(i=0;i<1000;i++) {
(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      }
(gdb) l 3
1       #include <stdio.h>
2       #include <stdlib.h>
3       int *table=NULL;
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;
(gdb) p table
$1 = (int *) 0x0
(gdb)

```

bug1

- bug2 - wyjście poza rozmiar tablicy. Rozmiar tablicy wynosi 1000 a pętla iteruje się aż do miliona. Wobec czego następuje odwołanie do niezainicjalizowanej pamięci co powoduje błąd.

```

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) l 1
1        #include <stdio.h>
2        int table[1000];
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) p table
$2 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136,
137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193,
194, 195, 196, 197, 198, 199...}
(gdb) p table[1000]
$3 = 1000
(gdb) p table[1001]
$4 = 1001
(gdb) p table[1002]
$5 = 1002
(gdb) p table[1003]
$6 = 1003
(gdb) p table[1007]
$7 = 1007
(gdb) p table[1008]
Cannot access memory at address 0x412000

```

bug2

Wartość zmiennej `i`, dla której pojawia się błąd jest zagadkowy dlaczego akurat `1008`? W związku z tym przeprowadziliśmy dodatkowe testy, mające na celu sprawdzenie tej zależności. Przy rozmiarze tablicy `2000` oraz `3000` występuje przesunięcie o wielkości `1024 (=2^10)`. Następnie po zwiększeniu rozmiaru tablicy otrzymujemy błąd przy wartości zmiennej `i` `2032 (= 1008 + 1024)` a dla rozmiaru `3000` wartość zmiennej `i` wynosi `3056 (=1008 + 2 * 1024)`.

Nie udało nam się dokładnie ustalić co jest tego przyczyną. Przypuszczamy że początkowe przesunięcie o `8` (`32B`, rozmiar `int`'a został przez nas sprawdzony w trakcie laboratorium i wynosi `4B`) może być spowodowane koniecznością inicjalizacji

programu i przydzielenia mu pamięci przez system operacyjny i związanych z tym operacji np. **stronicowaniem**.

Warto jednak zauważyć, że przy zmianie rozmiaru tablicy, przesunięcie zwiększa się o tę samą wartość **1024**, która jest najbliższą potęgą **2** większą od **1000** (a o tyle zwiększamy rozmiar tablicy).

W zależności od konkretnego systemu operacyjnego, kompilatora a nawet architektury sprzętowej, moment w którym występuje błąd **SIGSEGV** może się różnić. Błąd ten może nie występować natychmiast, gdy następuje próba nadpisania danych poza tablicą. Dzieje się tak dlatego, że przydzielane rozmiary pamięci występują w postaci potęgi liczby **2**. Dopiero gdy nadpisujemy jakąś ważną strukturę, taką jak **wskaźnik powrotu** na stosie, to możemy ten błąd zauważyć.

```
Program received signal SIGSEGV, Segmentation fault.
main () at buggy-1.0/bug2.c:8
8      table[i]=i;
(gdb) p i
$1 = 2032
(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) l 1
1      #include <stdio.h>
2      int table[2000];
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     }
```

bug2 z tablica 2000

```
Program received signal SIGSEGV, Segmentation fault.
main () at buggy-1.0/bug2.c:8
8      table[i]=i;
(gdb) p i
$1 = 3056
(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) █
```

bug2 z tablica 3000

- bug3 - **s1** zostaje nadpisany ponieważ są zmienne **s1** i **s2** jedno po drugim więc przydzielone im adresy występują po sobie i zawartość zmiennej **s2** zostaje nadpisana bo nadpisany zostaje znak **\0** w **s1**

```

Breakpoint 1, main () at buggy-1.0/bug3.c:12
12      for(i=0;i<24;i++) {
(gdb) l 1
1      #include <stdio.h>
2      #include <stdlib.h>
3      char s1[10]="012345678\0";
4      char s2[10]="abcdefgh\0";
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     {
(gdb) l
11     int i;
12     for(i=0;i<24;i++) {
13         s1[i]=i+64;
14     }
15     printf("s1=%s\ns2=%s\n",s1,s2);
16 }
(gdb)

```

bug3

```

Process /usr/bin/bug3 created; pid = 1824
s1=@ABCDEFGHIJKLMNOPQRSTUVWXYZ
s2=JKLMNOPQRSTUVWXYZ

Child exited with status 0

```

## Podsumowanie

- W trakcie laboratorium udało nam się wykonać wszystkie zaplanowane ćwiczenia. Zostały one sprawdzone i zatwierdzone przez prowadzącego laboratorium.
- Przeciwiczyliśmy zagadnienia z tworzeniem, uruchamianiem i debugowaniem własnych pakietów w `OpenWRT` przy pomocy `gdb` oraz `gdbserver`
- Dodatkowo zapoznaliśmy się z procesem tworzenia oprogramowania przy użyciu `SDK` na konkretną platformę sprzętową.
- Wszystkie cele merytoryczne laboratorium zostały przez nas osiągnięte