

Patryk Welenc, Jan Szczepkowski

Raport ZAKO

Wstęp

Tematem projektu były „Nietypowe języki programowania”, w naszym przypadku były to języki GO oraz Rust. W ramach projektu skupialiśmy się na łączeniu wymienionych języków z językiem C. Jako przykład zastosowania posłużył nam protokół Sampled Values.

O językach

Prace nad językiem GO rozpoczęły się w 2007 z inicjatywy inżynierów Google’a Roberta Griesemera, Roba Pike’a oraz Kena Thompsona. Głównymi założeniami języka miało być bezpieczeństwo C++ wraz z czytelnością Pythona. Język zaprezentowano w 2009 roku na blogu Google’a. Język zyskał uznanie za swoje założenia takie jak minimalizm, proste reguły, łatwość obsługi współbieżności oraz czytelność kodu. W GO zostały napisane takie projekty jak Docker czy Kubernetes.

Prace nad językiem Rust rozpoczęły się w 2006 roku z inicjatywy Graydona Hoare’a, a od 2009 roku rozwój języka został wsparty przez Mozillę. Głównym celem Rust była eliminacja błędów związanych z pamięcią, typowych dla C i C++, bez konieczności używania garbage collector’a. Język został oficjalnie zaprezentowany w 2010 roku, a jego pierwsze stabilne wydanie ukazało się w 2015 roku. Rust zdobył uznanie dzięki unikalnemu systemowi własności pamięci, bezpieczeństwu w czasie kompilacji oraz wysokiej wydajności. W Rust powstały projekty takie jak silnik CSS Stylo w przeglądarce Firefox, mikrohipernadzorca Firecracker Amazona czy narzędzie do wyszukiwania tekstu ripgrep.

Porównanie języków

Cecha	Go	Rust	C
Współbieżność	Dobra – ma goroutines i kanały, bardzo prosty model	Średnia – bezpieczna, ale bardziej złożona	Słaba – brak wsparcia w standardzie, wszystko ręcznie
Wydajność	Średnia – narzut GC, ale wystarczająca dla większości aplikacji	Wysoka – brak GC, nowoczesne optymalizacje	Bardzo wysoka – pełna kontrola, bez runtime’u
Bezpieczeństwo	Średnie – statyczne typowanie, ale możliwe wyścigi danych	Bardzo wysokie – wymusza bezpieczeństwo pamięci i współbieżności	Niskie – brak ochrony przed błędami pamięci
Zarządzanie pamięcią	Automatyczne – GC usuwa potrzebę ręcznego zarządzania	Manualne bez GC – własność i pożyczki, brak wycieków	Ręczne – łatwo o błędy jak wycieki lub użycie po zwolnieniu

Cecha	Go	Rust	C
Szybkość działania	Dobra – wystarczająca dla aplikacji serwerowych	Bardzo dobra – bliska C, z bezpieczeństwem	Najlepsza – minimalny narzut, bezpośrednie sterowanie sprzętem
Szybkość kompilacji	Bardzo szybka – szybki kompilator i system build	Wolna – skomplikowana analiza zależności i optymalizacji	Szybka – prosty model kompilacji
Ekosystem	Dobry – solidne wsparcie dla backendu i DevOps	Rośnie – dynamiczna społeczność, dużo nowoczesnych bibliotek	Ogromny (legacy) – dominacja w systemach operacyjnych
Typowe zastosowania	Backend, DevOps – np. Docker, Kubernetes	Systemy krytyczne – np. silnik CSS Stylo dla Firefox	Embedded, OS – np. Linux, mikrokontrolery

Protokół Sampled Values

Protokół Sampled Values wykorzystywany jest w elektroenergetyce. Protokół wyróżnia komunikację publisher/subscriber. Służy do wymiany informacji pomiędzy jednostkami scalającymi (Merging Units) a inteligentnymi urządzeniami elektronicznymi (IED – Intelligent Electronic Device) w stacji elektroenergetycznej za pośrednictwem sieci Ethernet. Koncepcja komunikacji SV polega na tym, że nadawca (publisher) okresowo wysyła wiadomości w ściśle określonych odstępach czasu. Interwał czasowy zależy od dwóch czynników: częstotliwości sygnału pomiarowego oraz liczby próbek na okres. Ponieważ protokół SV opiera się na modelu publisher/subscriber, komunikacja jest możliwa wyłącznie w obrębie lokalnej sieci (LAN).

Zalety protokołu

- Niskie opóźnienie
- Eliminacja sygnałów analogowych
- Jest częścią standardu IEC 61850
- Pozwala na analizę próbek w czasie rzeczywistym

Etap pierwszy

Pierwszym etapem projektu była kompilacja protokołu z biblioteki [libiec61850](#) napisanej w języku C. W ramach łatwości odtwarzania korzystaliśmy z wirtualnych interfejsów. Aby je stworzyć posłużyliśmy się komendami:

```
sudo ip link add veth0 type veth peer name veth1
sudo ip link set veth0 up
sudo ip link set veth1 up
```

A następnie skompilowaliśmy subskrybenta oraz publishera. Aby to zrobić należy wejść do folderu [examples](#) a następnie do folderu [sv_subscriber](#) oraz [sv_publisher](#). W obu tych folderach do kompilacji należy użyć komendy:

make

Utworzone pliki odpaliliśmy z uprawnieniami super usera:

```
sudo ./sv_publisher veth0  
sudo ./sv_subscriber veth1
```

Należy pamiętać o zdefiniowaniu interfejsów w komendzie, domyślnie ustawiony jest interfejs `eth0`

Odpalenie programów powinno dać następujący efekt:

Publisher:

```
jan@jan: ~/zako/libiec61850/examples/sv_publisher  
jan@jan:~/zako/libiec61850/examples/sv_publisher$ sudo ./sv_publisher veth0  
Using interface veth0  
█
```

Subscriber:

```
jan@jan: ~/zako/libiec61850/examples/sv_subscriber  
jan@jan:~/zako/libiec61850/examples/sv_subscriber$ sudo ./sv_subscriber veth1  
svUpdateListener called  
svID=(svpub1)  
smpCnt: 2119  
confRev: 1  
DATA[0]: 3564.484375  
DATA[1]: 211.927200  
svUpdateListener called  
svID=(svpub2)  
smpCnt: 2119  
confRev: 1  
DATA[0]: 7128.968750  
DATA[1]: 423.854401  
svUpdateListener called  
svID=(svpub1)  
smpCnt: 2119  
confRev: 1  
DATA[0]: 3564.484375  
DATA[1]: 211.927200  
svUpdateListener called  
svID=(svpub2)  
smpCnt: 2119  
confRev: 1  
DATA[0]: 7128.968750  
DATA[1]: 423.854401  
svUpdateListener called  
svID=(svpub1)  
smpCnt: 2120  
confRev: 1  
DATA[0]: 3565.584473  
DATA[1]: 212.027206
```

```
DATA[1]: 212.027200
svUpdateListener called
svID=(svpub2)
smpCnt: 2120
confRev: 1
DATA[0]: 7131.168945
DATA[1]: 424.054413
svUpdateListener called
svID=(svpub1)
smpCnt: 2121
confRev: 1
DATA[0]: 3566.684570
DATA[1]: 212.127213
svUpdateListener called
svID=(svpub2)
smpCnt: 2121
confRev: 1
DATA[0]: 7133.369141
DATA[1]: 424.254425
svUpdateListener called
svID=(svpub1)
smpCnt: 2122
confRev: 1
DATA[0]: 3567.784668
DATA[1]: 212.227219
svUpdateListener called
svID=(svpub2)
smpCnt: 2122
confRev: 1
DATA[0]: 7135.569336
DATA[1]: 424.454437
svUpdateListener called
svID=(svpub1)
smpCnt: 2123
confRev: 1
DATA[0]: 3568.884766
DATA[1]: 212.327225
svUpdateListener called
svID=(svpub2)
smpCnt: 2123
confRev: 1
DATA[0]: 7137.769531
DATA[1]: 424.654449
svUpdateListener called
```

Do weryfikacji działania programu posłużyliśmy się programem [Wireshark](#). Po zainstalowaniu programu odpaliliśmy go z uprawnieniami super użytkownika:

```
sudo wireshark
```

Wybraliśmy odpowiedni interfejs:

```
veth1
veth0
```

```
}
}
```

A następnie zweryfikowaliśmy protokół wysyłanych ramek:

Protocol
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values

Etap drugi

Etap drugi polegał na połączeniu programu napisanego w języku C z językami Go oraz Rust w taki sposób, aby cała funkcjonalność była wywoływana z poziomu tych języków.

Język GO

W języku [GO](#) łączenie odbywało się przy pomocy biblioteki [CGO](#).

Instalacja

Instalacja odbyła się za pomocą polecenia:

```
sudo apt install golang-go
```

Tutorial

CGO

CGO nie wymaga osobnej instalacji, jest ono pobierane wraz z językiem GO. Aby korzystać z biblioteki należy dodać odpowiedni import. Bezpośrednio nad tym importem powinien znaleźć się blok komentarzy z odpowiednimi deklaracjami w języku C. Należy pamiętać, że pomiędzy blokiem komentarzy a importem nie może być pustych linii. Aby wywołać funkcję z C należy przed nazwą funkcji dodać przedrostek [C](#). Przykładowy program wygląda następująco:

```
package main

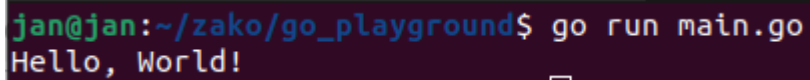
/*
#include <stdio.h>
void helloWorld() {
    printf("Hello, World!\n");
}
*/
```

```
import "C"

func main() {
    C.helloWorld()
}
```

Aby taki program uruchomić należy wpisać komendę

```
go run <nazwa pliku>
```



```
jan@jan:~/zako/go_playground$ go run main.go
Hello, World!
```

Łączenie protokołu SV

Pierwszym krokiem było utworzenie statycznej biblioteki protokołu SV. Aby to zrobić należy wejść do root directory [libiec61850](#) a następnie wykonać polecenie:

```
make
```

Następnie biblioteka pojawi się w ścieżce `./build/libiec61850.a`.

Kolejnym krokiem było przepisanie [sv_subscriber](#) oraz [sv_publisher](#) na język GO. Ze względu na problem z plikami nagłówkowymi działaliśmy w obrębie tego samego repozytorium.

W celu przepisania plików najpierw stworzyliśmy pliki nagłówkowe z deklaracjami wszystkich wykorzystywanych funkcji oraz typów danych w funkcji main w programie w języku C. Natomiast funkcje deklarowane w plikach w języku C przekopiowaliśmy do bloku komentarzy w CGO.

Następnie ustawiliśmy odpowiednie opcje dla kompilatora C oraz linkera. Ma to miejsce w bloku komentarzy nad importem i wygląda następująco:

```
#cgo CFLAGS: -I../src
#cgo LDFLAGS: -L../src/build -liec61850
```

Finalnie przepisaliśmy programy na język GO. Po dołączeniu kilku bibliotek programy wyglądają następująco:

Subscriber

```

main.go X
libiec1850 > examples > sv_publisher > go > main.go
1 package main
2
3 /*
4  * #cgo CFLAGS: -I../././src
5  * #cgo LDFLAGS: -L../././build -licec1850
6  * #include <stdio.h>
7  * #include <signal.h>
8  * #include <stdbool.h>
9  * #include "sv_publisher_example.h"
10
11 bool running = true;
12
13 void sigint_handler(int signalId)
14 {
15     running = 0;
16 }
17
18 import "C"
19 import (
20     "fmt"
21     "time"
22 )
23
24 func main() {
25     C.signal(C.SIGINT, (*[0]byte)(C.sigint_handler))
26
27     svPublisher := C.SVPublisher_create(nil, C.CString("veth1"))
28
29     if svPublisher != nil {
30         asdu1 := C.SVPublisher_addASDU(svPublisher, C.CString("svpub1"), nil, 1)
31
32         var float1 C.int = C.SVPublisher_ASDU_addFLOAT(asdu1)
33         var float2 C.int = C.SVPublisher_ASDU_addFLOAT(asdu1)
34         var ts1 C.int = C.SVPublisher_ASDU_addTimestamp(asdu1)
35
36         asdu2 := C.SVPublisher_addASDU(svPublisher, C.CString("svpub2"), nil, 1)
37         var float3 C.int = C.SVPublisher_ASDU_addFLOAT(asdu2)
38         var float4 C.int = C.SVPublisher_ASDU_addFLOAT(asdu2)
39         var ts2 C.int = C.SVPublisher_ASDU_addTimestamp(asdu2)
40
41         C.SVPublisher_setupComplete(svPublisher)
42
43         var fVal1 C.float = C.float(1234.5678)
44         var fVal2 C.float = C.float(0.12345)
45
46         for C.running == true {
47             var ts C.Timestamp
48             C.Timestamp_clearFlags(&ts)
49             C.Timestamp_setTimeInMilliseconds(&ts, C.Hal_getTimeInMs())
50
51             C.SVPublisher_ASDU_setFLOAT(asdu1, float1, fVal1)
52             C.SVPublisher_ASDU_setFLOAT(asdu1, float2, fVal2)
53             C.SVPublisher_ASDU_setTimestamp(asdu1, ts1, ts)
54
55             C.SVPublisher_ASDU_setFLOAT(asdu2, float3, fVal1 * 2)
56             C.SVPublisher_ASDU_setFLOAT(asdu2, float4, fVal2 * 2)
57             C.SVPublisher_ASDU_setTimestamp(asdu2, ts2, ts)
58
59             C.SVPublisher_ASDU_increasesmpCnt(asdu1)
60             C.SVPublisher_ASDU_increasesmpCnt(asdu2)
61
62             fVal1 += 1.1
63             fVal2 += 0.1
64
65             C.SVPublisher_publish(svPublisher)
66
67             time.Sleep(50 * time.Millisecond)
68
C sv_publisher_example.c X
libiec1850 > examples > sv_publisher > C sv_publisher_example.c
1 /*
2  * sv_publisher_example.c
3  *
4  * Example program for Sampled Values (SV) publisher
5  */
6
7 #include <signal.h>
8 #include <stdio.h>
9 #include "hal_thread.h"
10 #include "sv_publisher.h"
11
12 static bool running = true;
13
14 void sigint_handler(int signalId)
15 {
16     running = 0;
17 }
18
19 int
20 main(int argc, char** argv)
21 {
22     char* interface;
23
24     if (argc > 1)
25         interface = argv[1];
26     else
27         interface = "eth0";
28
29     printf("Using interface %s\n", interface);
30
31     signal(SIGINT, sigint_handler);
32
33     SVPublisher svPublisher = SVPublisher_create(NULL, interface);
34
35     if (svPublisher) {
36         /* Create first ASDU and add data points */
37
38         SVPublisher_ASDU asdu1 = SVPublisher_addASDU(svPublisher, "svpub1", NULL, 1);
39
40         int float1 = SVPublisher_ASDU_addFLOAT(asdu1);
41         int float2 = SVPublisher_ASDU_addFLOAT(asdu1);
42         int ts1 = SVPublisher_ASDU_addTimestamp(asdu1);
43
44         /* Create second ASDU and add data points */
45
46         SVPublisher_ASDU asdu2 = SVPublisher_addASDU(svPublisher, "svpub2", NULL, 1);
47
48         int float3 = SVPublisher_ASDU_addFLOAT(asdu2);
49         int float4 = SVPublisher_ASDU_addFLOAT(asdu2);
50         int ts2 = SVPublisher_ASDU_addTimestamp(asdu2);
51
52         SVPublisher_setupComplete(svPublisher);
53
54         float fVal1 = 1234.5678f;
55         float fVal2 = 0.12345f;
56
57         while (running) {
58             Timestamp ts;
59             Timestamp_clearFlags(&ts);
60             Timestamp_setTimeInMilliseconds(&ts, Hal_getTimeInMs());
61
62             /* update the values in the SV ASDUs */
63
64             SVPublisher_ASDU_setFLOAT(asdu1, float1, fVal1);
65             SVPublisher_ASDU_setFLOAT(asdu1, float2, fVal2);
66             SVPublisher_ASDU_setTimestamp(asdu1, ts1, ts);
67
68

```

Publisher

```

main.go X
libiec1850 > examples > sv_publisher > go > main.go
1 package main
2
3 /*
4  * #cgo CFLAGS: -I../././src
5  * #cgo LDFLAGS: -L../././build -licec1850
6  * #include <stdio.h>
7  * #include <signal.h>
8  * #include <stdbool.h>
9  * #include "sv_publisher_example.h"
10
11 bool running = true;
12
13 void sigint_handler(int signalId)
14 {
15     running = 0;
16 }
17
18 import "C"
19 import (
20     "fmt"
21     "time"
22 )
23
24 func main() {
25     C.signal(C.SIGINT, (*[0]byte)(C.sigint_handler))
26
27     svPublisher := C.SVPublisher_create(nil, C.CString("veth1"))
28
29     if svPublisher != nil {
30         asdu1 := C.SVPublisher_addASDU(svPublisher, C.CString("svpub1"), nil, 1)
31
32         var float1 C.int = C.SVPublisher_ASDU_addFLOAT(asdu1)
33         var float2 C.int = C.SVPublisher_ASDU_addFLOAT(asdu1)
34         var ts1 C.int = C.SVPublisher_ASDU_addTimestamp(asdu1)
35
36         asdu2 := C.SVPublisher_addASDU(svPublisher, C.CString("svpub2"), nil, 1)
37         var float3 C.int = C.SVPublisher_ASDU_addFLOAT(asdu2)
38         var float4 C.int = C.SVPublisher_ASDU_addFLOAT(asdu2)
39         var ts2 C.int = C.SVPublisher_ASDU_addTimestamp(asdu2)
40
41         C.SVPublisher_setupComplete(svPublisher)
42
43         var fVal1 C.float = C.float(1234.5678)
44         var fVal2 C.float = C.float(0.12345)
45
46         for C.running == true {
47             var ts C.Timestamp
48             C.Timestamp_clearFlags(&ts)
49             C.Timestamp_setTimeInMilliseconds(&ts, C.Hal_getTimeInMs())
50
51             C.SVPublisher_ASDU_setFLOAT(asdu1, float1, fVal1)
52             C.SVPublisher_ASDU_setFLOAT(asdu1, float2, fVal2)
53             C.SVPublisher_ASDU_setTimestamp(asdu1, ts1, ts)
54
55             C.SVPublisher_ASDU_setFLOAT(asdu2, float3, fVal1 * 2)
56             C.SVPublisher_ASDU_setFLOAT(asdu2, float4, fVal2 * 2)
57             C.SVPublisher_ASDU_setTimestamp(asdu2, ts2, ts)
58
59             C.SVPublisher_ASDU_increasesmpCnt(asdu1)
60             C.SVPublisher_ASDU_increasesmpCnt(asdu2)
61
62             fVal1 += 1.1
63             fVal2 += 0.1
64
65             C.SVPublisher_publish(svPublisher)
66
67             time.Sleep(50 * time.Millisecond)
68
C sv_publisher_example.c X
libiec1850 > examples > sv_publisher > C sv_publisher_example.c
1 /*
2  * sv_publisher_example.c
3  *
4  * Example program for Sampled Values (SV) publisher
5  */
6
7 #include <signal.h>
8 #include <stdio.h>
9 #include "hal_thread.h"
10 #include "sv_publisher.h"
11
12 static bool running = true;
13
14 void sigint_handler(int signalId)
15 {
16     running = 0;
17 }
18
19 int
20 main(int argc, char** argv)
21 {
22     char* interface;
23
24     if (argc > 1)
25         interface = argv[1];
26     else
27         interface = "eth0";
28
29     printf("Using interface %s\n", interface);
30
31     signal(SIGINT, sigint_handler);
32
33     SVPublisher svPublisher = SVPublisher_create(NULL, interface);
34
35     if (svPublisher) {
36         /* Create first ASDU and add data points */
37
38         SVPublisher_ASDU asdu1 = SVPublisher_addASDU(svPublisher, "svpub1", NULL, 1);
39
40         int float1 = SVPublisher_ASDU_addFLOAT(asdu1);
41         int float2 = SVPublisher_ASDU_addFLOAT(asdu1);
42         int ts1 = SVPublisher_ASDU_addTimestamp(asdu1);
43
44         /* Create second ASDU and add data points */
45
46         SVPublisher_ASDU asdu2 = SVPublisher_addASDU(svPublisher, "svpub2", NULL, 1);
47
48         int float3 = SVPublisher_ASDU_addFLOAT(asdu2);
49         int float4 = SVPublisher_ASDU_addFLOAT(asdu2);
50         int ts2 = SVPublisher_ASDU_addTimestamp(asdu2);
51
52         SVPublisher_setupComplete(svPublisher);
53
54         float fVal1 = 1234.5678f;
55         float fVal2 = 0.12345f;
56
57         while (running) {
58             Timestamp ts;
59             Timestamp_clearFlags(&ts);
60             Timestamp_setTimeInMilliseconds(&ts, Hal_getTimeInMs());
61
62             /* update the values in the SV ASDUs */
63
64             SVPublisher_ASDU_setFLOAT(asdu1, float1, fVal1);
65             SVPublisher_ASDU_setFLOAT(asdu1, float2, fVal2);
66             SVPublisher_ASDU_setTimestamp(asdu1, ts1, ts);
67
68

```

Jedyną różnicą jest na sztywno ustalony interfejs **veth0**.

Następnie programy należy uruchomić w obu ścieżkach za pomocą polecenia:

```
sudo go run main.go
```

Co powinno dać następujący efekt:

Subscriber

```
jan@jan:~/zako/libiec61850/examples/sv_subscriber/go$ sudo go run main.go
svUpdateListener called
  svID=(svpub1)
  smpCnt: 75
  confRev: 1
  DATA[0]: 1315.965942
  DATA[1]: 7.523445
svUpdateListener called
  svID=(svpub2)
  smpCnt: 75
  confRev: 1
  DATA[0]: 2631.931885
  DATA[1]: 15.046890
svUpdateListener called
  svID=(svpub1)
  smpCnt: 75
  confRev: 1
  DATA[0]: 1315.965942
  DATA[1]: 7.523445
svUpdateListener called
  svID=(svpub2)
  smpCnt: 75
  confRev: 1
  DATA[0]: 2631.931885
  DATA[1]: 15.046890
svUpdateListener called
```

Publisher

```
jan@jan:~/zako/libiec61850/examples/sv_publisher/go$ sudo go run main.go
█
```

Tak jak w pierwszym etapie poprawność programu zweryfikowaliśmy za pomocą programu [Wireshark](#).

Najpierw wpisaliśmy komendę:

```
sudo wireshark
```

Wybraliśmy odpowiedni interfejs:

veth1	┌
veth0	└

A następnie zweryfikowaliśmy protokół wysyłanych ramek:

Protocol
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values

Język Rust

W języku [Rust](#) łączenie odbywało się poprzez utworzenie biblioteki statycznej.

Instalacja

Instalacja odbyła się według [tutoriala](#)

Łączenie protokołu SV

Pierwszym krokiem było utworzenie statycznej biblioteki protokołu SV. Aby to zrobić należy wejść do root directory [libiec61850](#) a następnie wykonać polecenie:

```
make
```

Następnie biblioteka pojawi się w ścieżce `./build/libiec61850.a`.

⚠ UWAGA! Wszystkie opisane poniżej operacje na plikach muszą zostać wykonane zarówno na plikach w folderze `sv_subscriber` jak i folderze `sv_publisher`. Podane poniżej komendy są napisane tylko dla plików folderu `sv_publisher`. Utworzenie komend dla folderu `subscriber` odbywa się analogicznie. Jedyną konieczną zmianą jest podmienienie odpowiadających plików np. `sv_publisher_example.c` na `sv_subscriber_example.c`.

Kolejnym krokiem było przepisanie [sv_subscriber](#) oraz [sv_publisher](#) na język Rust. Finalny efekt prezentuje się następująco:

Subscriber

```

@ mains.rs x
home > patryk-welenc > pg > semVI > ZAKO > projekt > repo > ZAKO > rust > subscriber > @ mains.rs
52     let cstr = CString::from_ptr(sv_id_ptr);
53     println!(" svID={}", cstr.to_string_lossy());
54 }
55
56 let smp_cnt = SVSubscriber_ASDU_getSmpCnt(asdu);
57 let conf_rev = SVSubscriber_ASDU_getConfRev(asdu);
58
59 println!(" smpCnt: {}", smp_cnt);
60 println!(" confRev: {}", conf_rev);
61
62 let data_size = SVSubscriber_ASDU_getDataSize(asdu);
63 if data_size >= 8 {
64     let data0 = SVSubscriber_ASDU_getFLOAT32(asdu, 0);
65     let data1 = SVSubscriber_ASDU_getFLOAT32(asdu, 4);
66     println!(" DATA[0]: {}", data0);
67     println!(" DATA[1]: {}", data1);
68 }
69 }
70
71
72 fn main() {
73     unsafe {
74         signal(2, sigint_handler); // SIGINT = 2
75
76         let receiver = SVReceiver_create();
77
78         println!("Using interface veth1");
79         let iface = CString::new("veth1").unwrap();
80         SVReceiver_setInterfaceId(receiver, iface.as_ptr());
81
82         let subscriber = SVSubscriber_create(ptr::null(), 0x4000);
83         SVSubscriber_setListener(subscriber, sv_update_listener, ptr::null_mut());
84         SVReceiver_addSubscriber(receiver, subscriber);
85         SVReceiver_start(receiver);
86
87         if SVReceiver_isRunning(receiver) {
88             while RUNNING {
89                 Thread_sleep(1);
90             }
91
92             SVReceiver_stop(receiver);
93         } else {
94             println!("Failed to start SV subscriber.");
95         }
96
97         SVReceiver_destroy(receiver);
98     }
99 }

```

```

C sv_subscriber_example.c 3
home > patryk-welenc > pg > semVI > ZAKO > projekt > libiec61850-1.6 > examples > sv_subscriber > C sv_subscriber_example.c
24 svUpdateListener (SVSubscriber subscriber, void* parameter, SVSubscriber_ASDU asdu)
36 /*
42 *
43 * To prevent damages due configuration, please check the length of the
44 * data block of the SV message before accessing the data.
45 */
46 if (SVSubscriber_ASDU_getDataSize(asdu) >= 8) {
47     printf(" DATA[0]: %f\n", SVSubscriber_ASDU_getFLOAT32(asdu, 0));
48     printf(" DATA[1]: %f\n", SVSubscriber_ASDU_getFLOAT32(asdu, 4));
49 }
50
51 int run()
52 {
53     SVReceiver receiver = SVReceiver_create();
54
55     printf("Using interface veth1\n");
56     SVReceiver_setInterfaceId(receiver, "veth1");
57
58     /* Create a subscriber listening to SV messages with APPID 4000h */
59     SVSubscriber subscriber = SVSubscriber_create(NULL, 0x4000);
60
61     /* Install a callback handler for the subscriber */
62     SVSubscriber_setListener(subscriber, svUpdateListener, NULL);
63
64     /* Connect the subscriber to the receiver */
65     SVReceiver_addSubscriber(receiver, subscriber);
66
67     /* Start listening to SV messages - starts a new receiver background thread */
68     SVReceiver_start(receiver);
69
70     if (SVReceiver_isRunning(receiver)) {
71         signal(SIGINT, sigint_handler);
72
73         while (running)
74             Thread_sleep(1);
75
76         /* Stop listening to SV messages */
77         SVReceiver_stop(receiver);
78     }
79     else {
80         printf("Failed to start SV subscriber. Reason can be that the Ethernet interf
81     }
82
83     /* Cleanup and free resources */
84     SVReceiver_destroy(receiver);
85     return 0;
86 }
87

```

Publisher

```

@ mains.rs x
home > patryk-welenc > pg > semVI > ZAKO > projekt > repo > ZAKO > rust > publisher > @ mains.rs > {} extern "C"
41 fn main() {
42     let running = Arc::new(AtomicBool::new(true));
43     let r = running.clone();
44
45     let interface = CString::new("veth0").unwrap();
46     println!("Using interface veth0");
47
48     unsafe {
49         let publisher: *mut {unknown} = SVPublisher_create(src_mac: std::ptr::null(),
50
51         if publisher.is null() {
52             println!("Failed to create SV publisher");
53             return;
54         }
55
56         let asdu1: *mut {unknown} = SVPublisher_addASDU(publisher, sv_id: CString::new(
57         let float1: i32 = SVPublisher_ASDU_addFLOAT(asdu: asdu1);
58         let float2: i32 = SVPublisher_ASDU_addFLOAT(asdu: asdu1);
59         let ts1: i32 = SVPublisher_ASDU_addTimestamp(asdu: asdu1);
60
61         let asdu2: *mut {unknown} = SVPublisher_addASDU(publisher, sv_id: CString::new(
62         let float3: i32 = SVPublisher_ASDU_addFLOAT(asdu: asdu2);
63         let float4: i32 = SVPublisher_ASDU_addFLOAT(asdu: asdu2);
64         let ts2: i32 = SVPublisher_ASDU_addTimestamp(asdu: asdu2);
65
66         SVPublisher_setupComplete(publisher);
67
68         let mut f_val1: f32 = 1234.5678f32;
69         let mut f_val2: f32 = 0.12345f32;
70
71         while running.load(Ordering::SeqCst) {
72             let mut ts: Timestamp = Timestamp {
73                 seconds: 0,
74                 nanoseconds: 50,
75                 accuracy: 0,
76                 flags: 0,
77             };
78
79             Timestamp_clearFlags(&mut ts);
80             Timestamp_setTimeInMilliseconds(t: &mut ts, ms: Hal_getTimeInMs());
81
82             SVPublisher_ASDU_setFLOAT(asdu: asdu1, index: float1, val: f_val1);
83             SVPublisher_ASDU_setFLOAT(asdu: asdu1, index: float2, val: f_val2);
84             SVPublisher_ASDU_setTimestamp(asdu: asdu1, index: ts1, timestamp: ts);
85
86             SVPublisher_ASDU_setFLOAT(asdu: asdu2, index: float3, val: f_val1 * 2.0);
87             SVPublisher_ASDU_setFLOAT(asdu: asdu2, index: float4, val: f_val2 * 2.0);
88             SVPublisher_ASDU_setTimestamp(asdu: asdu2, index: ts2, timestamp: ts);
89

```

```

C sv_publisher_example.c 3
home > patryk-welenc > pg > semVI > ZAKO > projekt > libiec61850-1.6 > examples > sv_publisher > C sv_publisher_example.c
18
19 int run()
20 {
21     char* interface;
22
23     interface = "veth0";
24
25     printf("Using interface %s\n", interface);
26
27     signal(SIGINT, sigint_handler);
28
29     SVPublisher svPublisher = SVPublisher_create(NULL, interface);
30
31     if (svPublisher) {
32
33         /* Create first ASDU and add data points */
34
35         SVPublisher_ASDU asdu1 = SVPublisher_addASDU(svPublisher, "svpub1", NULL, 1);
36
37         int float1 = SVPublisher_ASDU_addFLOAT(asdu1);
38         int float2 = SVPublisher_ASDU_addFLOAT(asdu1);
39         int ts1 = SVPublisher_ASDU_addTimestamp(asdu1);
40
41         /* Create second ASDU and add data points */
42
43         SVPublisher_ASDU asdu2 = SVPublisher_addASDU(svPublisher, "svpub2", NULL, 1);
44
45         int float3 = SVPublisher_ASDU_addFLOAT(asdu2);
46         int float4 = SVPublisher_ASDU_addFLOAT(asdu2);
47         int ts2 = SVPublisher_ASDU_addTimestamp(asdu2);
48
49         SVPublisher_setupComplete(svPublisher);
50
51         float f_val1 = 1234.5678f;
52         float f_val2 = 0.12345f;
53
54         while (running) {
55             Timestamp ts;
56             Timestamp_clearFlags(&ts);
57             Timestamp_setTimeInMilliseconds(&ts, Hal_getTimeInMs());
58
59             /* update the values in the SV ASDUs */
60
61             SVPublisher_ASDU_setFLOAT(asdu1, float1, f_val1);
62             SVPublisher_ASDU_setFLOAT(asdu1, float2, f_val2);
63             SVPublisher_ASDU_setTimestamp(asdu1, ts1, ts);
64
65             SVPublisher_ASDU_setFLOAT(asdu2, float3, f_val1 * 2);
66             SVPublisher_ASDU_setFLOAT(asdu2, float4, f_val2 * 2);

```

Następnie konieczne było utworzenie pliku obiektowego z pliku C, ze względu na to, że Rust nie rozumie języka C. Potrafi się z nim łączyć, jednak nie kompiluje go na kod maszynowy. Utworzenie pliku obiektowego wykonujemy analogicznie dla obydwu folderów [sv_subscriber](#) oraz [sv_publisher](#). Najpierw weszliśmy do wskazanego folderu, a następnie przy pomocy gcc skompilowaliśmy istniejący plik .c do pliku obiektowego przy pomocy poniższej komendy:

```
gcc -I/home/sciezka_do_pliku/libiec61850-1.6/src \
-I/home/sciezka_do_pliku/libiec61850-1.6/hal/inc \
-I/home/sciezka_do_pliku/libiec61850-1.6/src/sampled_val\
ues \
-I/home/sciezka_do_pliku/libiec61850-1.6/src/common/inc \
\
-I/home/sciezka_do_pliku/libiec61850-1.6/src/mms/inc \
-I/home/sciezka_do_pliku/libiec61850-1.6/src/iec61850/in\
c \
-I/home/sciezka_do_pliku/libiec61850-1.6/src/logging \
-I/home/sciezka_do_pliku/libiec61850-1.6/src/r_session \
-c sv_publisher_example.c -o sv_publisher_example.o
```

Po wykonaniu tego zabiegu, z utworzonego pliku obiektowego stworzyliśmy bibliotekę statyczną, zwaną później jako [wrapper](#). Dokonałiśmy tego w ten sposób:

```
ar rcs libsvwrapper.a sv_publisher_example.o
```

Kiedy już utworzyliśmy wrapper byliśmy gotowi, aby zbudować nasz plik Rust. Skorzystaliśmy zatem z poniższej komendy, która zlinkowała odpowiednie biblioteki i utworzyła gotowy plik binarny:

```
rustc main.rs \
-L . \
-L ../../build \
-l static=svwrapper \
-l static=iec61850 \
-o sv_publisher_runner
```

Dzięki [rustc](#) jesteśmy w stanie połączyć odpowiednie biblioteki w języku C razem ze skompilowanym plikiem main napisanym w języku Rust. W ten sposób został utworzony plik sv_publisher_runner. Uruchomienie go wykonuje się poprzez komendę:

```
sudo ./sv_publisher_runner
```

Tak jak w pierwszym etapie poprawność programu zweryfikowaliśmy za pomocą programu [Wireshark](#) wpisując komendę:

```
sudo wireshark
```

Wybraliśmy odpowiedni interfejs:

```
veth1
veth0
```

```
}
}
```

A następnie zweryfikowaliśmy protokół wysyłanych ramek:

Protocol
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values
IEC61850 Sampled Values

Repozytorium

Jeżeli ktoś chciałby się bardziej zagłębić w temat to serdecznie zapraszamy na repozytorium [GitHub](#) ze wszystkimi plikami opracowanymi przez nas.