SCZR lab. 3

Realizacja komunikacji ze sprzętem w systemie Linux

W ćwiczeniu 3 będziemy badać komunikację programu napisanego w języku C i działającego pod kontrolą systemu Linux z urządzeniem zrealizowanym w emulatorze QEMU.

Materiał do przejrzenia
Slajdy z wykładów: WZ_W02 – tworzenie pakietów dla OpenWRT i używanie SDK, WZ_W04 – obsługa przerwań, zagadnienia związane z komunikacją ze sprzętem oraz omówienie modelu układu czasowego (timera) używanego w tym ćwiczeniu, WZ_W03 i WZ_W05 – ustawienia dotyczące priorytetu i szeregowania procesów.
Oczywiście, korzystne a nawet konieczne może być samodzielne uzupełnienie z innych źródeł wiadomości niezbędnych do realizacji ćwiczenia.

Wprowadzenie

W systemach czasu rzeczywistego bardzo istotną kwestią jest czas reakcji systemu na zdarzenia zgłaszane przez dołączone do niego urządzenia weścia/wyjścia.
Te zdarzenia mogą oznaczać na przykład otrzymanie nowego pakietu sieciowego, zawierającego dane, które należy przetworzyć w czasie rzeczywistym. Może to być powiadomienie o dostarczeniu nowej porcji danych z przetwornika analogowo-cyfrowego, które to dane po przetworzeniu zostaną wykorzystane do generacji sygnałów sterujących jakimś procesem. Może to być też informacja z układu odmierzającego czas o upłynięciu zaprogramowanego okresu czasu i konieczności wykonannia pewnych zaprogramowanych na dany moment działań. Ze względu na prostotę tego ostatniego scenariusza, właśnie on będzie badany w ćwiczeniu 3.
Urządzenie wykorzystane w ćwiczeniu zostało zrealizowane jako model w emulatorze QEMU. Pozwala to ominąć ograniczenia dotyczące sprzętu dostępnego w laboratorium, oraz ułatwia realizację ćwiczenia. Należy jednak podkreślić, że wyniki, jakie można uzyskać w tym ćwiczeniu wykazują dobrą zgodność z zachowaniem rzeczywistego sprzętu. Stosowne testy zostały przeprowadzone z wykorzystaniem układu „System on Chip” na płytce DE0-Nano-SoC, wspomnianej na wykładzie.
Ćwiczenie pozwala zaobserwować różnice między czasem reakcji systemu przy różnych sposobach realizacji obsługi tych zdarzeń – przez przerwania, albo przez aktywne odpytywanie (ang. „polling”). Można też zobaczyć, jak na ten czas wpływa obciążenie systemu. Rozwiązanie optymalne przy małym obciążeniu systemu może okazać się fatalne przy dużym obciążeniu.
Przy interpretacji wyników może też pomóc informacja, że przy bardzo małym obciążeniu, dla zmniejszenia poboru mocy, procesor może przejść w jeden z trybów oszczędzania energii. W takiej sytuacji obsługa przerwania może być znacznie opóźniona.
W trakcie ćwiczenia konieczne jest samodzielne skompilowanie emulatora QEMU uzupełnionego o model testowanego urządzenia. Szczególnie zainteresowani studenci mogą przy tej okazji przeanalizować sposób realizacji modelu urządzenia, a nawet spróbować go zmodyfikować i sprawdzić działanie zmodyfikowanej wersji.

1 Opis modelu urządzenia

Urządzeniem, które będziemy badać jest prosty układ czasowy (timer), generujący okresowe przerwania.
Urządzenia zawiera 6 32-bitowych rejestrów, dostępnych pod kolejnymi adresami.
Możemy je opisać jako strukturę w języku C:

```
typedef struct {
  uint32_t id;
  uint32_t stat;
  uint32_t divl;
  uint32_t divh;
  uint32_t cntl;
  uint32_t cnth;
} WzTim1Regs;
```

Znaczenie poszczególnych rejestrów jest następujące:
Rejestr “id”
Jest to rejestr tylko do odczytu. Zawiera wartość 0x7130900d. Pozwala się upewnić, że rzeczywiście pod podanym adresem znajduje się nasz timer.
Rejestr “stat”
Jest to rejestr zwracający status timera i pozwalający sterować zgłaszaniem przerwań.
Bit 31-szy (tylko do odczytu) informuje, że upłynął kolejny okres timera i że timer nie został powiadomiony o jego obsłużeniu.
Bit 0-wy (do zapisu i odczytu) pozwala włączać i wyłączać zgłaszanie przerwań. Timer zgłasza przerwanie, gdy ustawiony jest bit 31-szy i ustawiony jest bit 0-wy.
Rejestry “divl” i “divh”
Są to dwa rejestry, tworzące w sumie 64-bitowy rejestr, ustalający co ile nanosekund timer ma zgłaszać przerwanie. Rejestry te można zarówno odczytywać, jak i zapisywać. W przypadku zapisu należy najpierw zapisywać rejestr **divh** (starsze 32-bity), a potem **divl** (młodsze 32-bity). W momencie zapisania rejestru **divl**, timer pobiera zawartość wcześniej wisaną do rejestru **divh**, łącząc je w pełną 64-bitową wartość. Wpisanie wartości zerowej do obu rejestrów wyłącza timer. Wpisanie wartości niezerowej, włącza timer.
Rejestry “cntl” i “cnth”
Te dwa rejestry pozwalją odczytać czas, jaki upłynął od ostatniego upłynięcia okresu timera. Odczyt rejestru **“cntl”** dostarcza młodszych 32 bitów, powodując równocześnie zatrzaśnięcie w rejestrze **“cnth”** starszych 32 bitów.
Dlatego odczyt należy zawsze najpierw odczytywać rejestr **“cntl”**, a potem **“cnth”**.
Zapis do rejestru **“cntl”** potwierdza, że program został powiadomiony o upłynięciu okresu timera.

1.1 Opis sterownika

Skorzystanie z timera wymaga użycia sterownika. Po załadowaniku sterownika komendą “modprobe drv-sysbus-tim1”, timer będzie dostępny przez plik specjalny “/dev/my_tim0”. Sterownik pozwala używać urządzenia na dwa sposoby – albo dając użytkownikowi bezpośredni dostęp do rejestrów timera i pozwalając zrealizować pełną obsługę w programie, albo zapewniając praktycznie pełną obsługę w sterowniku z wykorzystaniem przerwań.

1.1.1 Obsługa timera z pełnym wykorzystaniem sterownika

W tym trybie krytyczna operacja odczytania rejestrów **“cntl”** i **“cnth”** jest realizowana przez sterownik w procedurze obsługi przerwania.
Z punktu widzenia programu użytkownika, komunikacja z timerem realizowana jest przez zapisu i odczyt pliku specjalnego. Zarówno zapis, jak i odczyt muszą być wykonywane blokami po 8 bajtów (można przesłać po prostu zmienną typu uint64_t). Przy zapisie, przesłana wartość jest wpisywana (z zachowaniem właściwej kolejności) do rejestrów **“divl”** i **“divh”**, odpowiednio włączając, lub wyłączając timer.
Odczyt z pliku blokuje proces czytający, dopóki nie upłynie okres timera i nie zostanie zgłoszone przerwanie. Odczytana wartość zawiera czas w nanosekundach od ostatniego upłynięcia okresu

timera do chwili odczytu (jest on pobierany z rejestrów **“cntl”** i **“cnth”**). Dzięki temu możemy łatwo zbadać, jaki czas upływa między zażądaniem przez urządzenie obsługi, a jego obsłużeniem w procedurze obsługi przerwania.
Przy zamykaniu pliku urządzenia sterownik dba o wyłączenie timera i zgłaszania przerwań.

1.1.2 Obsługa timera przez program z minimalnym wsparciem sterownika

W tym przypadku, sterownik wykorzystywany jest jedynie do zmapowania rejestrów timera w przestrzeń adresową programu, przy pomocy instrukcji mmap. Uzyskany wskaźnik można potraktować jako wskaźnik do struktury WzTim1Regs, odwołując się następnie bezpośrednio do rejestrów. W takim przypadku należy wyłączyć zgłaszanie przerwań przez timer.
Obsługa przez program może wyglądać następująco:

1. Otwarcie pliku specjalnego /dev/my_tim0
2. Zmapowanie rejestrów timera w pamięć procesu
3. Wyłączenie przerwań przez rejestr “stat”
4. Zaprogramowanie żądanego okresu timera
5. Pętla obsługi timera
    1. Czekanie, aż zostanie ustawiony bit 31-szy rejestru “stat”
    2. Odczyt rejestrów “cntl” i “cnth” (czas od ustawienia wyżej wspomnianego bitu)
    3. Zapis do rejestru “cntl” (skasowanie 31-go bitu w rejestrze „stat”)
6. Wyłączenie timera
7. Usunięcie mapowania rejestrów
8. Zamknięcie pliku

2 Realizacja ćwiczenia

2.1 Kompilacja emulatora QEMU z modelem timera

Proszę pobrać źródła emulatora QEMU z dodanym modelem timera. Znajdują się one w repozytorium https://github.com/wzab/qemu , w gałęzi “sczr1”.

```
git clone https://github.com/wzab/qemu
cd qemu
git checkout sczr1
```

Po pobraniu źródeł, należy je skompilować tak, aby zmodyfikowany emulator został zainstalowany w wybranym katalogu.

```
./configure --target-list=arm-softmmu,aarch64-softmmu --enable-virtfs \
            --prefix=/moj/katalog/na/qemu
scripts/hxtool -h < hmp-commands-info.hx  > hmp-commands-info.h
scripts/hxtool -h < hmp-commands.hx  > hmp-commands.h
make -j 8 install
```

W trakcie kompilacji QEMU, można rozpocząć pracę nad kolejnymi etapami ćwiczenia.
Po udanej kompilacji i instalacji, proszę uruchomić w skompilowanym QEMU emulowaną maszynę virt z systemem OpenWRT.

2.2 Stworzenie i uruchomienie programów współpracujących z timerem

W archiwum SCZR_lab_cw3.tar.bz2 znajdują się źródła pakietów OpenWRT. Pakiet “drv-sysbus-tim1” zawiera źródła sterownika do timera. Pakiet “user-sysbus-tim1” zawiera źrodła aplikacji “gen_load1” i szkielety aplikacji “user_tim1” i “user_tim2”.

Zadaniem aplikacji “gen_load1” jest pełne obciążenie procesora, aby zbadać wpływ obciążenia systemu na opóźnienie obsługi timera. W plikach “user_tim1.c” i “user_tim2.c” proszę zaimplementować programy obsługujące timer. Pierwszy z nich powinien w pełni wykorzystywać sterownik, a drugi realizować obsługę w całości w programie, używając sterownika tylko do mapowania rejestrów.

Oba programy powinny przyjmować dwa parametry linii poleceń: okres timera, podany w nanosekundach i liczbę przebiegów pętli. Oba programy powinny też wyprowadzać na standardowe wyjście (będziecie Państwo mogli przekierować je do pliku) informacje o opóźnieniu obsługi timera w kolejnych przebiegach pętli:

```
1 32543
2 41656
[...]
```

2.3 Badania wpływu sposobu obsługi oraz obciążenia systemu na opóźnienie obsługi timera

2.3.1 Badanie opóźnienia obsługi timera na maszynie jednoprocesorowej

Proszę uruchomić w emulatorze QEMU 64-bitową maszynę virt z jednym procesorem.

Proszę zaobserwować rozkład opóźnień obsługi timera dla programu korzystającego w pełni ze sterownika, oraz dla programu realizującego całość obsługi w programie. Proszę przeprowadzić te obserwacje dla przypadku, gdy jest to jedyny uruchomiony program, oraz dla 1, 2 lub 3 instancji programu “gen_load1” uruchomionych w tle.

Proszę przeanalizować uzyskane wyniki.

2.3.2 Badanie opóźnienia obsługi timera na maszynie dwuprocesorowej

Proszę uruchomić w emulatorze QEMU 64-bitową maszynę virt z dwoma procesorami i powtórzyć eksperymenty z poprzedniego punktu.

Proszę porównać uzyskane wyniki z tymi z poprzedniego punktu.

2.3.3 Zbadanie wpływu parametrów szeregowania procesów na opóźnienie obsługi timera

Proszę zaproponować zmiany parametrów szeregowania procesów mogące wpłynąć na czas opóźnienia obsługi timera, przeprowadzić odpowiednie eksperymenty i przedyskutować wyniki.

Cennik
zadanie 2.1 – 0 do 3 pkt
zadanie 2.2 – 0 do 4 pkt
zadanie 2.3.1 – 0 do 3 pkt
zadanie 2.3.2– 0 do 2 pkt
zadanie 2.3.3 – 0 do 3 pkt
czytelność kodu, komentarze – -5 do 0 pkt
prezentacja wyników – -10 do 0 pkt

Maximum: 15 pkt.