Laboratorium 7: „Liczniki czasu” (jedne zajęcia)

dr inż. Arkadiusz Chrobot

27 kwietnia 2020

Spis treści

Wprowadzenie

Liczniki czasu nazywane także czasomierzami (ang. *timer*) są jednym z mechanizmów dolnych połówek. Operacje realizowane w ramach liczników czasu są wykonywane w kontekście przerwania. Dostępne są dwa rodzaje tych mechanizmów: liczniki czasu niskiej rozdzielczości (ang. *low resolution timers*), które odliczają czas w taktach zegara systemowego i liczniki czasu wysokiej rozdzielczości (ang. *high resolution timers*), które odmierzają czas w nanosekundach (1 nanosekunda = 10^{-9} sekundy). Rozdział 1 tej instrukcji jest poświęcony licznikom czasu niskiej rozdzielczości, a rozdział 2 licznikom czasu wysokiej rozdzielczości. Instrukcja kończy się listą zadań do samodzielnego wykonania w ramach zajęć laboratoryjnych.

1. Liczniki czasu niskiej rozdzielczości

Mechanizm liczników czasu niskiej rozdzielczości, znany także jako „koło liczników czasu" (ang. *timers wheel*) historycznie pojawił się jako pierwszy w jądrze Linuksa, ale od wersji 2.6.16 jego działanie jest oparte na mechanizmie liczników czasu wysokiej rozdzielczości. Opóźnienie dla liczników niskiej rozdzielczości liczone jest w taktach zegara systemowego. Długość trwania takiego taktu jest zależna od platformy sprzętowej, na której działa Linux. Przykładowo, dla komputerów z procesorami o architekturze x86 pojedynczy takt zegara trwa 4 ms, natomiast dla komputera Raspberry Pi 3 jego długość wynosi 10 ms. Dla większości typowych zastosowań taka rozdzielczość jest wystarczająca. Licznik niskiej rozdzielczości po zaszeregowaniu wykonywany jest tylko raz. Aby mógł być ponownie wykonany musi zostać ponownie zaszeregowany z nową wartością opóźnienia, które wyznaczane jest względem momentu uruchomienia pomiaru czasu w systemie operacyjnym. Najczęściej jest ono wyliczane jako suma taktów zegara systemowego, po którym licznik ma się wykonać i bieżącej wartości zmiennej `jiffies`. Jest to zmienna, w której jądro przechowuje liczbę taktów zegara systemowego od chwili jego uruchomienia.

1.1. Opis API

Aby w module móc użyć liczników niskiej rozdzielczości konieczne jest włączenie do jego kodu pliku nagłówkowego `linux/timer.h`. Takie liczniki są reprezentowane wewnątrz jądra systemu za pomocą zmiennych strukturalnych typu `struct timer_list`. Dla programisty użytkującego liczniki niskiej rozdzielczości, istotne są tylko trzy jej pola:

1. `expires` - zawiera liczbę taktów zegara, po których funkcja związana z licznikiem ma zostać wywołania,
2. `function` - jest wskaźnikiem na wspomnianą funkcję,
3. `data` - zawiera liczbę typu `unsigned long int`, która jest przekazywana jako argument wywołania do funkcji związanej z licznikiem.

Operacje wykonywane w ramach licznika niskiej rozdzielczości muszą być oprogramowane w funkcji o następującym prototypie:

`void timer_handler(unsigned long int data);`

Zarówno nazwa funkcji, jaki i nazwa parametru mogą być inne. Kod tej funkcji musi uwzględniać te same wymagania, co w przypadku funkcji realizowanej w ramach taskletu. Przez parametr może być przekazana tej funkcji dowolna liczba naturalna mieszcząca się w zakresie typu `unsigned long int`. Może ona posłużyć do odróżnienia liczników, w przypadku kiedy funkcja wywoływana jest z poziomu nie jednego, ale kilku z nich.

Do obsługi liczników niskiej rozdzielczości służą między innymi następujące funkcje i makra:

`init_timer(timer)` - makro, które inicjuje strukturę typu `struct timer_list`, której adres jej mu przekazywany jako argument. Inicjacja ta nie obejmuje opisanych wcześniej pól, które programista musi zainicjować samodzielnie.

`void add_timer(struct timer_list *timer)` - funkcja, która szereguje (aktywuje) licznik czasu do wykonania. Jako argument wywołania przyjmuje adres struktury `struct timer_list`, które reprezentuje w systemie ten licznik.

`int mod_timer(struct timer_list *timer, unsigned long expires)` - funkcja pozwalająca zmodyfikować opóźnienie, po którym licznik zostanie uruchomiony. Jeśli licznik nie był aktywowany, to funkcja go uaktywni i zwróci liczbę `0`. Jeśli licznik był aktywowany to funkcja zmodyfikuje jego opóźnienie i zwróci wartość `1`. **Opóźnienie liczników, które zostały uaktywnione można zmieniać jedynie przy pomocy tej funkcji.**

`int del_timer(struct timer_list * timer)` - funkcja deaktywująca licznik czasu. Przyjmuje ona jako argument wywołania adres struktury typu `struct timer_list`, która reprezentuje licznik do deaktywacji. Funkcja ta zwraca `0` jeśli licznik był przed jej wywołaniem nieaktywowany lub `1`, jeśli był aktywowany.

`int del_timer_sync(struct timer_list *timer)` - funkcja, która w przypadku systemów wieloprocesorowych deaktywuje licznik i dodatkowo sprawdza, czy funkcja związana z tym licznikiem nie jest wykonywana na którymś z procesorów. Jeśli tak jest, to czeka z zakończeniem swojego działania tak długo, aż zakończy się wykonanie funkcji licznika. Zwraca te same wartości co `del_timer()`. Zaleca się jej używanie zamiast `del_timer()` zarówno w przypadku systemów wieloprocesorowych, jak i jednoprocesorowych. W przypadku tych ostatnich kompilator automatycznie zamienia wywołanie tej funkcji, na makro o takiej samej nazwie, które następnie rozwijane jest do wywołania funkcji `del_timer()`.

1.2. Przykład

Listing 1 zawiera kod źródłowy modułu, który demonstruje użycie licznika niskiej rozdzielczości.

Listing 1: Przykładowy moduł używający licznika niskiej rozdzielczości

```
1  #include<linux/module.h>
2  #include<linux/timer.h>
3
4  static struct timer_list timer;
5
6  static void timer_handler(unsigned long int data)
7  {
8          pr_info("Timer nr %lu is active!\n",data);
9  }
10
11 static int __init timer_module_init(void)
12 {
13         init_timer(&timer);
14         timer.expires = jiffies + 15*HZ;
15         timer.data = 0;
```

```
16          timer.function = timer_handler;
17          add_timer(&timer);
18          return 0;
19  }
20
21  static void __exit timer_module_exit(void)
22  {
23          if(del_timer_sync(&timer))
24                  pr_notice("The timer was not active!\n");
25  }
26
27  module_init(timer_module_init);
28  module_exit(timer_module_exit);
29  MODULE_LICENSE("GPL");
30  MODULE_AUTHOR("Arkadiusz Chrobot <a.chrobot@tu.kielce.pl>");
31  MODULE_DESCRIPTION("A module demonstrating the usage of timers.");
32  MODULE_VERSION("1.0");
```

W wierszu nr 2 kodu źródłowego modułu włączany jest plik nagłówkowy `linux/timer.h`, którego zawartość pozwala skorzystać z mechanizmu liczników niskiej rozdzielczości. W wierszu nr 4 została zdefiniowana struktura o nazwie `timer` i typie `struct timer_list`, która będzie reprezentowała pojedynczy licznik czasu o niskiej rozdzielczości. Wiersze 6-9 zawierają definicję funkcji, która będzie wykonywana w ramach tego licznika. Jej działanie jest proste. Umieszcza ona w buforze jądra komunikat zawierający numer licznika przekazany jej przez parametr `data`. Przyjęto w tym module, że liczniki czasu będą numerowane od 0. W konstruktorze modułu (wiersze 11-19) struktura `timer` najpierw jest inicjowana przez funkcję `init_timer()`, a następnie przypisywane są wartości do jej pól `expires`, `data` i `function`. W tym ostatnim zapisywany jest adres funkcji `timer_handler()`, która będzie wykonana w ramach licznika. W polu `data` zapisywana jest liczba, która zostanie przekazana funkcji licznika, jako argument jej wywołania (jest to `0`). W polu `expires` zapisywana jest wartość opóźnienia dla tego licznika, która jest wyliczana jako suma bieżącej wartości zmiennej `jiffies` i iloczynu `15*HZ`, gdzie `HZ` jest stałą określającą częstotliwość pracy zegara systemowego (liczbę taktów, które ten zegar wykona w ciągu sekundy). Wyrażenie to zapewnia zatem, że licznik zostanie wykonany po upłynięciu około[1] piętnastu sekund od chwili bieżącej. Po inicjacji wspomnianych pól licznik jest aktywowany przy użyciu funkcji `add_timer()`. W destruktorze modułu (wiersze 21-25) licznik jest deaktywowany przez usunięciem modułu z jądra za pomocą funkcji `del_timer_sync()`. Jeśli licznik do tego momentu się nie uaktywnił, to w buforze jądra zostanie umieszczony przez moduł odpowiedni komunikat.

2. Liczniki czasu wysokiej rozdzielczości

Liczniki czasu wysokiej rozdzielczości, podobnie jak liczniki czasu niskiej rozdzielczości nie są mechanizmem precyzyjnym, ale oferują odliczanie czasu z rozdzielczością nanosekundową, co jest wymagane w niektórych zastosowaniach, takich, jak np. obsługa multimediów. Aby pełne możliwości mechanizmu liczników czasu wysokiej rozdzielczości były dostępne musi istnieć w systemie odpowiedni zegar, który będzie stanowił źródło czasu dla tego podsystemu, a ponadto jądro musi być skompilowane z włączoną odpowiednią opcją (obecnie ta opcja domyślnie jest zaznaczona). Jeśli te warunki nie zostaną spełnione, to API liczników czasu wysokiej rozdzielczości będzie dostępne, ale same liczniki będą działać z taką samą rozdzielczością, jak liczniki o niskiej rozdzielczości. Najczęściej w nowoczesnych systemach komputerowych dostępne są dwa rodzaje zegarów, które spełniają wymagania mechanizmu liczników o wysokiej rozdzielczości. Te typy zegarów są identyfikowane przy pomocy dwóch stałych: `CLOCK_MONOTONIC` i `CLOCK_REALTIME`. Pierwszy rodzaj, to zegary, których wskazania rosną w tym samym tempie od momentu włączenia komputera do momentu jego wyłączenia. Drugi rodzaj, to zegary odliczające „czas ścienny” (ang. *wall-clock time*), czyli czas bieżący, obowiązujący w danej strefie czasowej. Ich wskazania mogą być zatem korygowane np. przez administratora systemu lub przez podsystem odpowiedzialny za obsługę protokołu NTP, co wiąże się z tym, że mogą one przyrastać w nierównym tempie lub nawet

[1] Mechanizm liczników czasu niskiej rozdzielczości nie jest precyzyjny. Należy także uwzględnić czas, który upłynie od chwili przypisania wartości polu `exipres` do chwili aktywacji licznika.

się cofać. Więcej informacji, choć trochę już nieaktualnych, na temat tego mechanizmu można znaleźć w artykule dostępnym pod adresem: `https://lwn.net/Articles/167897/`.

2.1. Opis API

Interfejs liczników czasu wysokiej rozdzielczości staje się dostępny po włączeniu do kodu źródłowego modułu pliku nagłówkowego `linux/hrtimer.h`, aby jednak skorzystać z tych liczników należy wcześniej zapoznać się typem `ktime_t` zdefiniowanym w pliku `linux/ktime.h`. Tego pliku nie trzeba włączać do kodu modułu korzystającego z liczników wysokiej rozdzielczości, bo jest on dołączany także do pliku `linux/hrtimer.h`. Typ `k_time` definiuje unię, która zawiera pojedyncze pole, przechowujące czas w nanosekundach. Ze względu na zachowanie kompatybilności z wcześniejszymi wersjami jądra unia ta nie może zostać zamieniona na zmienną prostego typu i musi być obsługiwana przy pomocy między innymi następujących makr i funkcji:

`ktime_t ktime_set(const s64 secs, const unsigned long nsecs)` - funkcja `inline` , która zwraca unię typu `k_time` zawierającą wskazanie czasu, które wyliczane jest na podstawie przekazanych jej przez parametry wartości. Przez pierwszy parametr przekazywana jest liczba sekund, a przez drugi liczba nanosekund.

`ktime_sub(lhs, rhs)` - makro, które odejmuje od wartości unii typu `k_time` przekazanej przez parametr `lhs` wartość unii typu `k_time` przekazanej przez parametr `rhs`. Wynik zwracany przez to makro jest także typu `k_time`.

`ktime_add(lhs, rhs)` - makro, które dodaje do siebie wartości unii typu `k_time` przekazanych przez parametry i zwraca wynik także w postaci unii typu `k_time`.

`ktime_add_ns(kt, nsval` - makro, które dodaje do wartości unii typu `k_time` przekazanej przez parametr `kt` liczbę nanosekund przekazanych przez parametr `nsval`. Wynik zwracany jest w postaci unii typu `k_time`.

`ktime_to_ns(kt)` - makro, które konwertuje czas zapisany w unii typu `ktime_t` będącej jego argumentem do liczby nanosekund.

Liczniki czasu wysokiej precyzji są reprezentowane w jądrze systemu za pomocą struktur typu `struct hrtimer`. Dla programisty posługującego się tego typu licznikami najważniejszym polem w tej strukturze jest pole o nazwie `function`, które jest wskaźnikiem na funkcję realizowaną w ramach licznika. Ta funkcja musi mieć następujący prototyp:

```
enum hrtimer_restart hrtimer_handler(struct hrtimer *hrtimer);
```

Nazwa funkcji podana w tym prototypie oraz nazwa parametru mogą być zmienione. Ponieważ tak, jak w przypadku liczników niskiej rozdzielczości, funkcja ta wykonywana jest w kontekście przerwania, to nie może zawierać żadnych operacji, które związane są z przełączaniem wątku w stan oczekiwania, ponieważ kontekst ten nie jest związany z żadnym wątkiem jądra. Funkcja ta powinna zwrócić jedną z dwóch wartości będących elementami typu wyliczeniowego `enum hrtimer_restart`. Są to: `HRTIMER_NORESTART` oraz `HRTIMER_RESTART`. Jeśli funkcja zwróci pierwszą z nich, to będzie to oznaczało, że licznik nie zostanie automatycznie aktywowany i ponownie wykonany. Jeśli jedna zwróci ona drugą z opisywanych wartości, to licznik zostanie automatycznie aktywowani i powtórnie wykonany. Z obsługą liczników wysokiej rozdzielczości związane są między innymi następujące funkcje:

`void hrtimer_init(struct hrtimer *timer, clockid_t which_clock, enum hrtimer_mode mode)` - - funkcja ta inicjuje strukturę typu `struct hrtimer`, której adres jest jej przekazywany jako pierwszy argument wywołania. Jako drugi argument przekazywana jest jedna z dwóch wcześniej opisanych stałych, służących do określenia rodzaju zegara używanego do odliczania czasu dla licznika. Trzecim argumentem jej wywołania powinna być wartość będąca elementem typu wyliczeniowego `hrtimer_mode`, która określa sposób odliczania opóźnienia dla licznika. Najczęściej jest stosowany jeden z dwóch następujących elementów: `HRTIMER_MODE_ABS` - opóźnienie będzie liczone względem momentu uruchomienia komputera, lub `HRTIMER_MODE_REL` - opóźnienie będzie liczone względem chwili bieżącej. Funkcja ta nie inicjuje pola `function`, które trzeba zainicjować przy użyciu zwykłego operatora przypisania.

`void hrtimer_start(struct hrtimer *timer, ktime_t tim, const enum hrtimer_mode mode)` - funkcja `inline`, która aktywuje licznik czasu wysokiej rozdzielczości reprezentowany przez strukturę typu `struct hrtimer`, której adres jest jej przekazywany jako pierwszy argument wywołania. Jak drugi argument wywołania tej funkcji przekazywana jest unia typu `k_time` zawierająca liczbę nanosekund, po których funkcja związana z licznikiem powinna zostać uruchomiona. Trzeci argument jest taki sam, jak w przypadku funkcji `hrtimer_init()`.

`int hrtimer_cancel(struct hrtimer *timer)` - funkcja, które dezaktywuje licznik reprezentowany przez strukturę typu `struct hrtimer`, której adres jest jej przekazany jako argument wywołania. Jeśli funkcja związana z licznikiem jest już wykonywana, to `hrtimer_cancel()` czeka na jej zakończenie. Opisywana funkcja zwraca `0`, jeśli anulowany licznik nie był aktywowany, lub `1` jeśli był.

`int hrtimer_try_to_cancel(struct hrtimer *timer)` - funkcja ta działa podobnie do `hrtimer_cancel()`. Różnica między nimi polega na tym, że jeśli funkcja związana z anulowanym licznikiem jest w trakcie wykonania, to `hrtimer_try_to_cancel()` nie czeka na zakończenie jej wykonania, tylko od razu zwraca wartość `-1`.

`u64 hrtimer_forward(struct hrtimer *timer, ktime_t now, ktime_t interval)` - funkcja ta wywoływana jest zazwyczaj przez funkcję realizowaną w ramach licznika. Jej zadaniem jest ustalenie nowego czasu ponownej aktywacji tego licznika. Przyjmuje ona trzy argumenty wywołania. Pierwszym jest adres struktury typu `struct hrtimer` reprezentującej licznik, drugim jest czas w nanosekundach, zapisany w unii typu `k_time`. Względem tego czasu będzie liczone nowe opóźnienie. Trzecim argumentem jest właśnie opóźnienie, liczone w nanosekundach i zapisane w unii typu `ktime_t`. Funkcja zwraca liczbę określającą ile okresów opóźnień czekał licznik na uruchomienie. Jeśli system nie jest obciążony i działa w pełni poprawnie, to zwykle ta liczba wynosi `1`.

`u64 hrtimer_forward_now(struct hrtimer *timer, ktime_t interval)` - funkcja `inline`, która jest prostszą odpowiedniczką funkcji `hrtimer_forward()`, gdyż nie wymaga przekazania czasu względem którego liczone jest opóźnienie.

`void hrtimer_restart(struct hrtimer *timer)` - funkcja `inline`, która aktywuje licznik, który został wcześniej anulowany. Przyjmuje jeden argument wywołania, którym jest adres struktury typu `struct hrtimer` reprezentującej ten licznik.

`ktime_t hrtimer_get_expires(const struct hrtimer *timer)` - funkcja `inline`, która zwraca w postaci unii typu `ktime_t` opóźnienie, po którym funkcja związana z licznikiem zostanie uruchomiona. Adres struktury typu `struct hrtimer` reprezentującej ten licznik jest przekazywany do funkcji jako argument wywołania.

`s64 hrtimer_get_expires_ns(const struct hrtimer *timer)` - funkcja `inline`, która działa podobnie do funkcji `hrtimer_get_expires()`, ale zwraca opóźnienie wyrażone w nanosekundach.

`ktime_t hrtimer_expires_remaining(const struct hrtimer *timer)` - funkcja `inline`, która zwraca czas pozostały do uruchomienia licznika, z którym związana jest struktura typu `struct hrtimer`, której adres jej przekazany funkcji jako argument wywołania. Zwrócona wartość jest zapisana w unii typu `k_time`.

`int hrtimer_is_hres_active(struct hrtimer *timer)` - funkcja ta pobiera jako argument wywołania adres struktury typu `struct hrtimer` związanej z licznikiem i zwraca wartość `1` jeśli działa on jako licznik wysokiej rozdzielczości lub `0` jeśli działa on jako licznik niskiej rozdzielczości.

2.2. Przykład

Listing 2 zawiera kod źródłowy modułu, który demonstruje użycie licznika czasu wysokiej rozdzielczości.

Listing 2: Przykładowy moduł używający licznika wysokiej rozdzielczości

```
1 #include<linux/module.h>
2 #include<linux/hrtimer.h>
3
4 static struct hrtimer timer;
5 static ktime_t delay;
6
```

```
7  static enum hrtimer_restart hrtimer_function(struct hrtimer *hrtimer)
8  {
9          u64 overruns;
10         pr_info("The timer is active!\n");
11         overruns = hrtimer_forward_now(hrtimer,delay);
12         pr_info("The overruns number since last activation: %llu.", overruns);
13         return HRTIMER_RESTART;
14 }
15
16 static int __init hrtimer_module_init(void)
17 {
18         delay = ktime_set(1,0);
19         hrtimer_init(&timer,CLOCK_MONOTONIC,HRTIMER_MODE_REL);
20         timer.function = hrtimer_function;
21         hrtimer_start(&timer,delay,HRTIMER_MODE_REL);
22         return 0;
23 }
24
25 static void __exit hrtimer_module_exit(void)
26 {
27         if(!hrtimer_cancel(&timer))
28                 pr_alert("The timer was not active!\n");
29 }
30
31 module_init(hrtimer_module_init);
32 module_exit(hrtimer_module_exit);
33 MODULE_LICENSE("GPL");
34 MODULE_AUTHOR("Arkadiusz Chrobot <a.chrobot@tu.kielce.pl>");
35 MODULE_DESCRIPTION("A module demonstrating the use of high resolution timers.");
36 MODULE_VERSION("1.0");
```

Do kodu źródłowego modułu w wierszu nr 2 włączany jest plik nagłówkowy `linux/hrtimer.h`. W wierszu nr 4 deklarowana jest zmienna typu `struct hrtimer`, która będzie reprezentowała licznik, a wierszu nr 5 zmienna `delay`, która jest unią typu `ktimer_t` i będzie przechowywała liczbę nanosekund, po których licznik zostanie aktywowany. Wiersze 7-14 zawierają definicję funkcji, która będzie wykonywana w ramach licznika. Ta funkcja oprócz umieszczania w buforze jądra komunikatów o uruchomieniu licznika i liczbie opóźnień, które minęły od jego ostatniej aktywacji (wartość prawidłowa to `1`), reaktywuje licznik, z którym jest związana. Dokonuje tego zmieniając czas opóźnienia tego licznika przy pomocy funkcji `hrtimer_forward_now()`. Do jej wywołania `hrtimer_function()` przekazuje wskaźnik na strukturę licznika, który otrzymała przez swój parametr, oraz opóźnienie zapisane w zmiennej `delay`. Bezpośrednie odwołania do tej zmiennej są dopuszczalne w funkcjach, gdyż jest to zmienna zadeklarowana ze słowem kluczowym `static`, a więc jest ona lokalna wewnątrz modułu, nie jest dostępna na zewnątrz. Jest to często spotykana praktyka w bibliotekach języka C przeznaczonych dla przestrzeni użytkownika, więc można też skorzystać z niej w bibliotece przeznaczonej dla jądra, jaką jest moduł. Wartość zwrócona przez `hrtimer_forward_now()` jest zapisywana do zmiennej lokalnej i jest to wspomniana wcześniej liczba opóźnień. Aby licznik był ponownie aktywowany funkcja `hrtimer_function()` musi jeszcze zwrócić wartość `HRTIMER_RESTART`. W konstruktorze modułu najpierw inicjowana jest zmienna `delay` (wiersz nr 18) za pomocą funkcji `ktime_set()`. Zapisana w niej liczba nanosekund jest równoważna jednej sekundzie. W wierszu nr 19 inicjowana jest zmienna `timer` przy pomocy funkcji `timer_init()`. Jako źródło czasu dla licznika reprezentowanego przez tę zmienną wybierany jest zegar monotoniczny, a opóźnienie będzie liczone względem chwili bieżącej. W wierszu nr 20 licznik do pola `function` zmiennej `timer` przypisywany jest adres funkcji `hrtimer_function()`, która będzie wykonywana w ramach licznika. W wierszu nr 21 licznik jest aktywowany z opóźnieniem określonym przez wartość zmiennej `delay` (1 sekunda), liczonym względem momentu bieżącego. Ta aktywacja jest dokonywana za pomocą wywołania funkcji `hrtimer_start()`. W destruktorze licznik jest anulowany przed usunięciem modułu z jądra systemu za pomocą funkcji `htimer_cancel()`. Działanie tego modułu wygodnie jest obserwować korzystając z polecenia `dmesg -w -d`.

Zdania

1. `[3 punkty]` Zmodyfikuj funkcję związaną z licznikiem z modułu 1 tak, aby aktywowała ona ponownie ten licznik.

2. `[5 punktów]` Napisz moduł jądra, w którym licznik czasu niskiej rozdzielczości będzie aktywowany cyklicznie, ale jego opóźnienie przy każdej aktywacji będzie zwiększane dwukrotnie. Działanie licznika powinno być przerwane wtedy, gdy wartość tego opóźnienia przekroczy ustalony przez Ciebie próg, np. 10 sekund.

3. `[7 punktów]` Napisz moduł, w którym użyte będą dwa liczniki wysokiej rozdzielczości. Pierwszy powinien mieć długie opóźnienie, a drugi powinien być wykonywany cyklicznie i używając odpowiednich funkcji opisanych w podrozdziale 2.1 umieszczać informacje o pierwszym liczniku w pliku, w systemie plików `procfs`.

4. `[3 punkty]` Zmodyfikuj funkcję związaną z licznikiem z modułu 2, tak, aby ten licznik był uruchomiony tylko raz.

5. `[5 punktów]` Napisz moduł, w którym dwa liczniki czasu wysokiej rozdzielczości będą uruchamiały funkcje wspólnie użytkujące jedną zmienną. Każda z tych funkcji powinna umieszczać wartość tej zmiennej w buforze jądra, a następnie ją modyfikować. Ta związana z pierwszym licznikiem może np. zwiększać ją o dwa, a druga o trzy.

6. `[7 punktów]` Napisz moduł, w którym funkcja wykonywana w ramach cyklicznie uruchamianego licznika czasu wysokiej rozdzielczości będzie tworzyła i dodawała na koniec listy kolejne elementy. W tych elementach powinny być zapisywane liczby, np. pseudolosowe. Drugi licznik czasu, też wysokiej rozdzielczości powinien cyklicznie odczytywać zawartość elementów tej listy i umieszczać ją w buforze jądra. Zabezpiecz moduł tak, aby pierwszy licznik nie tworzył więcej niż 100 elementów. Przed usunięciem modułu z jądra systemu lista powinna być zlikwidowana.