Systemy Operacyjne 2

Zarządzanie czasem i liczniki czasu w Linuksie

Arkadiusz Chrobot

Katedra Systemów Informatycznych

25 kwietnia 2024

Plany

Wstęp

Informacja o upływie czasu jest istotna zarówno dla jądra, jak i oprogramowania przestrzeni użytkownika. Niektóre funkcje jądra są wywoływane periodycznie, po upływie określonego odcinka czasu, który stanowi *czas względny* (od jednego takiego momentu do następnego). Aplikacje użytkownika, takie jak np. systemy zarządzania bazami danych, potrzebują informacji o *czasie bezwzględnym* (ang. *absolute time*) nazywanym także *czasem zegara ściennego* (ang. *wall-clock time*) lub rzeczywistym czasem (ang. *real-world time*). Jądro systemu jest odpowiedzialne za zaspokojenie obydwu wymagań. W ramach tego wykładu zostaną przedstawione podsystemy jądra odpowiedzialne za pomiar czasu oraz liczniki czasu (ang. *timers*).

Zarządzanie czasem

Głównym elementem pozwalającym jądru mierzyć czas jest *zegar systemowy* (ang. *system timer*), który generuje przerwania z określoną częstością. Dla tych przerwań jest zarejestrowana specjalna procedura obsługi. Platformy systemowe obsługiwane przez Linuksa mają wiele urządzeń, które mogą spełniać rolę takiego zegara. Ich działanie zazwyczaj bazuje na impulsach elektrycznych generowanych przez kwarcowy oscylator kryształowy. Przykładowo, komputery bazujące na procesorach x86 mają kilka typów takich urządzeń. Najstarsze nazwane jest PIT (ang. *Programmable Interval Timer*). Do nowszych zaliczają się: HPET (ang. *High Precision Event Timer*), zegar (L)APIC. W podobne urządzenia wyposażone są także inne platformy sprzętowe. Każde z nich zapewnia inną dokładność i wymaga innej obsługi. W serii 2.6 jądra programiści dodali warstwę abstrakcji, która ukrywa większość tych różnic i ujednolica sposób obsługi opisywanych urządzeń.

Zegary wirtualne

Ta warstwa abstrakcji bazując na urządzeniach sprzętowych tworzy dwa typy zegarów wirtualnych.

źródła czasu (ang. *clock sources*) są licznikami, których wartość monotonicznie rośnie i może być tylko odczytywana;

urządzenia zdarzeń czasowych (ang. *clock event devices*) generują po upływie określonego interwału czasu przerwanie sygnalizujące zdarzenie, mogą to robić cyklicznie lub jednokrotnie.

Każde ▸ źródło czasu i urządzenie zdarzeń czasowych ma przypisaną jakość wyrażoną liczbą naturalną. Zegar z najwyższą jakością jest wybierany jako domyślne źródło czasu lub domyślne urządzenie zdarzeń czasowych dla systemu. Dodatkowo, domyśle urządzenie zdarzeń czasowych zostaje zegarem systemowym.

Stała HZ

Częstotliwość zegara systemowego jest określana przez stałą HZ[1]. Okres zegara jest odwrotnością tej stałej. Jej wartość, w przypadku większości platform obsługiwanych przez Linuksa, wynosi 100. Do wyjątków zaliczają się komputery bazujące na procesorach x86. W ich przypadku ta stała także miała wartość 100 (okres 10 ms), ale w serii 2.4 jądra zmieniono ją na 1000 (okres 1 ms), aby zaspokoić potrzeby multimedialnego oprogramowania użytkowego. Zmiana polepszyła rozdzielczość przerwania zegarowego i sterowanie czynnościami jądra zależnymi od czasu. Niestety, spowodowała ona także wzrost obciążenia CPU obsługą większej liczby przerwań zegarowych. Dodatkowo, opisywana zmiana spowodowała problemy z obsługą protokołu NTP (ang. *Network Time Protocol*). Ostatecznie wartość stałej HZ dla komputerów z procesorami x86 została ustalona na 250 (okres 4 ms). Wymogi aplikacji multimedialnych zostały spełnione przy użyciu liczników czasu wysokiej rozdzielczości.

[1] Hz (hertz) jest jednostką częstotliwości.

Dynamiczne odliczanie (ang. *Dynamic Ticks*)

W przypadku platform, które wymagają oszczędności energii, jak systemy wbudowane i laptopy, jądro może zostać skonfigurowane w taki sposób, aby ignorowało stałą `HZ` i generowało zdarzenia czasowe tylko wtedy, gdy są one potrzebne. Obsługa przerwań zegarowych za każdym razem, kiedy się pojawiają oznacza zbędny wydatek energii. Jeśli jej oszczędność jest priorytetem, a jądro oraz aplikacje użytkownika nie muszą wykonywać żadnych czynności przez, np. dwie sekundy, to zegar wygeneruje kolejne przerwanie zegarowe za 2 s, a nie co 4 ms w tym przedziale czasu.

Zmienna `jiffies`

Liczba przerwań zegarowych wygenerowanych od uruchomienia systemu komputerowego jest przechowywana w zmiennej `jiffies`. Czas pracy tego systemu (ang. *up-time*) może być wyznaczony jako iloraz `jiffies/HZ`. W komputerach 32-bitowych `jiffies` jest 32-bitowa, a 64-bitowa w komputerach 64-bitowych. Zmiana wartości stałej `HZ` spowodowała problemy z tą zmienną dla 32-bitowych platform sprzętowych bazujących na procesorach x86. Kiedy częstotliwość zegara systemowego wynosiła 100 przepełnienie `jiffies` następowało co 497 dni. Ten okres zmniejszył się do $49,7$ dnia po jej zwiększeniu do 1000. By rozwiązać ten problem, programiści jądra dodali 64-bitową zmienną o nazwie `jiffies_64`. W 32-bitowych komputerach zmienna `jiffies` jest nałożona na mniej znaczące 4 bajty zmiennej `jiffies_64`, a w komputerach 64-bitowych obie nazwy oznaczają tę samą zmienną. W przypadku 32-bitowych komputerów wartość zmiennej `jiffies_64` powinna być odczytywana za pomocą funkcji `get_jiffies_64()`, aby zapewnić niepodzielność tej operacji.

Zmienna `jiffies`

Programiści jądra zdefiniowali cztery makra, które ułatwiają określanie czasu przy pomocy wartości zmiennej `jiffies`. Uwzględniają one jej przepełnienia. Każde z nich przyjmuje dwa argumenty, którymi są wartości zmiennej `jiffies`. Makro `time_after` daje niezerową wartość (prawdę), kiedy moment wyznaczony jego pierwszym argumentem zachodzi po momencie wyznaczonym jego drugim argumentem. Makro `time_before` daje prawdę, gdy moment wyznaczony jego pierwszym argumentem zachodzi przed momentem wyznaczonym jego drugim argumentem. Dwa pozostałe makra to `time_after_eq` i `time_before_eq`, które działają podobnie jak ich odpowiedniki bez przyrostka `_eq`, ale dodatkowo dają prawdę, gdy oba argumenty są równe.

Stała USER_HZ

Aplikacje użytkowe zakładają, że wartość stałej HZ jest równa 100. Jest to prawdą dla większości platform sprzętowych, z pewnymi wyjątkami, takimi jak komputery bazujące na procesorach x86 lub DEC Alpha. Z powodu takich systemów komputerowych programiści jądra zdefiniowali odrębną stałą o nazwie USER_HZ, której wartość określa częstotliwość zegara systemowego oczekiwaną przez aplikacje użytkownika. Konwersją faktycznej liczby przerwań zegarowych do liczby odpowiadającej wartości stałej USER_HZ zajmują się funkcje `jiffies_to_clock_t()` i `jiffies_64_to_clock_t()`.

Zegar rzeczywistego czasu

Jądro Linuksa dokonuje również pomiaru czasu bezwzględnego, którego potrzebują niektóre aplikacje użytkowe. Zwykle informacja o bieżącym czasie jest dostarczana przez urządzenie sprzętowe[2], które jest odczytywane przez jądro w trakcie startu systemu, a potem tylko aktualizowane przez proceduję obsługi przerwania zegara systemowego. Momentem, od którego liczony jest czas w Linuksie jest tzw. początek epoki Uniksa, czyli północ 1 stycznia 1970 roku[3]. Oprogramowanie użytkowe może uzyskać informację o bieżącej dacie i czasie za pomocą wywołania systemowego `gettimeofday()`.

[2] Popularnym wyjątkiem są komputery Raspberry Pi.

[3] W 32-bitowych komputerach ten sposób przechowywania informacji o czasie może spowodować problem nazwany Y2K38.

Procedura obsługi przerwania zegara systemowego

Kod odpowiedzialny za obsługę przerwania zegara systemowego jest podzielony na dwie części: zależną i niezależną od sprzętu. Pierwsza z nich wykonuje następujące czynności:

- obsługuje zegar systemowy,
- okresowo aktualizuje urządzenie rzeczywistego czasu,
- wywołuje funkcję `tick_periodic()`, która stanowi implementację części niezależnej od sprzętu obsługi przerwania zegara systemowego.

Druga część zajmuje się następującymi operacjami:

- aktualizuje zmienną `jiffies_64`,
- aktualizuje dla każdego procesora statystyki użycia zasobów przez bieżący proces,
- aktualizuje zmienne przechowujące informację o bieżącym czasie,
- oblicza średnie obciążenie systemu.

Liczniki czasu niskiej rozdzielczości

Jądro udostępnia *liczniki czasu* (ang. *timers*), zwane również *licznikami jądra* (ang. *kernel timers*) lub *licznikami dynamicznymi* (ang. *dynamic timers*), które pozwalają opóźnić pewne operacje do wykonania w kontekście przerwania, na określony czas. Innymi słowy liczniki czasu są kolejną implementacją dolnych połówek. Istnieją dwa rodzaje takich liczników. Pierwszy to *liczniki o niskiej rozdzielczości* (ang. *low-resolution timers*). Ich rozdzielczość jest rzędu okresu zegara systemowego. Nie są one wystarczające w zastosowaniach multimedialnych lub związanych z czasem rzeczywistym, ale nadają się do rozwiązywania większości innych problemów z mierzeniem czasu. Te liczniki nie są także cykliczne, po tym jak skończą odliczać nie są automatycznie odnawiane. Pojedynczy licznik czasu o niskiej rozdzielczości jest reprezentowany przez zmienną typu `struct timer_list`. Należy nadać wartości jej polom `expires` i `function`.

Liczniki czasu niskiej rozdzielczości

Pierwsze z tych pól określa czas, po którym licznik jest uruchamiany. Wartość ta jest wyrażana w okresach zegara systemowego. W drugim należy umieścić adres funkcji realizującej operacje wykonywane po uruchomieniu licznika. Jej prototyp jest następujący:

```
void timer_function(struct timer_list *)
```

Nazwa tej funkcji jest zazwyczaj inna niż w tym prototypie. Struktura typu `struct timer_list`, wraz z opisanymi polami, jest inicjowana z użyciem makra o nazwie `timer_setup`. Licznik jest aktywowany przez funkcję `add_timer()`.

Liczniki czasu niskiej rozdzielczości

Czas, po którym aktywny licznik ma być uruchomiony może być zmieniony przy użyciu funkcji `mod_timer()`. To jedyny bezpieczny sposób modyfikacji tej wartości po aktywacji licznika. Jeśli ta funkcja zostanie użyta dla nieaktywnego licznika, to spowoduje jego aktywację. W systemach uniprocesorowych aktywny licznik można usunąć używając funkcji `del_timer()`. W przypadku systemów wieloprocesorowych do tego celu należy użyć funkcji `del_timer_sync()`. Funkcja `timer_pending()` zwraca `1` jeśli zostanie wywołana dla aktywnego licznika lub `0` w przeciwny przypadku. Jeśli funkcja implementująca czynności uruchamiane w ramach licznika używa współdzielonych zasobów, to powinna stosować odpowiednie środki synchronizacji do ich ochrony. Przed pojawieniem się wersji 4.8 jądra liczniki czasu niskiej rozdzielczości były grupowane w nieposortowaną listę, ale podzieloną na pięć części. O przynależności licznika do danej części decydował jego czas uruchomienia.

Liczniki czasu niskiej rozdzielczości

W miarę upływu czasu wszystkie liczniki były przenoszone sukcesywnie między tymi częściami, aż do ich uruchomienia. Kiedy nadszedł ten moment funkcja związana z licznikiem była uruchamiana w ramach przerwania programowego wyzwolonego przez procedurę obsługi przerwania zegarowego, związanego z lokalnym urządzeniem zdarzeń czasowych. To rozwiązanie powodowało niedokładności w czasie uruchomienia licznika maksymalnie rzędu okresu zegara systemowego, ale było kosztowne ze względu na operację przenoszenia liczników. W wersji 4.8 została dodana poprawka autorstwa Thomasa Gleixnera, która eliminuje potrzebę przeprowadzania tej operacji, ale powoduje, że niedokładności w przypadku liczników o bardzo długim czasie uruchomienia mogą być nawet rzędu godzin. Działanie nowego mechanizmu liczników niskiej rozdzielczości jest zależne od wartości stałej `HZ` i opisane będzie dla przypadku, gdy wynosi ona 250.

Liczniki czasu niskiej rozdzielczości

Dla takiej wartości tej stałej jądro tworzy hierarchię 9 tablic (w innych przypadkach może to być 8) o 64 elementach będących wskaźnikami na listy liczników. Najwyższa w tej hierarchii tablica zawiera liczniki o czasach aktywacji między 0 a 255 ms i rozdzielczości 4 ms, czyli 2^2 ms. Niedokładność w czasie ich uruchomienia, to maksymalnie 4 ms. Rozdzielczości oferowane przez kolejne tablice to odpowiednio: 32 ms (2^5), 256 ms (2^8), 2048 ms (2^{11}), 2^{14} ms, 2^{17} ms, 2^{20} ms, 2^{23} ms, 2^{26} ms. Niedokładności liczników umieszczonych w najniższej w hierarchii tablicy to około 18 h, ale ich czasy uruchomiania są z przedziału od 6 do 49 dni. Dlaczego zastąpiono relatywnie dokładne liczniki mniej precyzyjnymi? Ponieważ większość z nich jest używana jako programowy mechanizm typu *watchdog*, czyli zazwyczaj są usuwane, zanim się uruchomią. Precyzja jest wymagana tylko w przypadku tych, które mają krótki czas uruchomienia, w pozostałych nie jest wymagana, a operacja przenoszenia liczników była w poprzednim mechanizmie zbyt ▸ kostowna.

Liczniki czasu wysokiej rozdzielczości

Liczniki czasu wysokiej rozdzielczości (ang. *high-resolution timers*) oferują rozdzielczość nanosekundową i stosowane są wszędzie tam, gdzie liczniki niskiej rozdzielczości są niewystarczające, jak w przypadku przetwarzania danych multimedialnych. Początkowo programiści Linuksa chcieli zastąpić liczniki niskiej rozdzielczości tymi o wysokiej rozdzielczości, ale okazało się to zadaniem trudnym, więc pozostawiono oba mechanizmy. Liczniki wysokiej rozdzielczości są dostępne, jeśli ich obsługa została dodana do jądra w trakcie kompilacji, a sprzęt dostarcza co najmniej dwóch zegarów, które one mogą używać. Jeśli ten drugi wymóg nie jest spełniony to API tych liczników jest dostępne, ale działają one tak samo jak liczniki niskiej rozdzielczości. Wymagane zegary to zegar monotoniczny i rzeczywistego czasu. Oba oferują rozdzielczość nanosekundową, ale wartość pierwszego jest zawsze zwiększana w określonych momentach, a wartość drugiego może w pewnych przypadkach być zmniejszana i jądro musi kompensować te zmiany. W systemach wieloprocesorowych zazwyczaj każdy procesor ma po parze takich zegarów.

Api Liczników czasu wysokiej rozdzielczości

Licznik czasu wysokiej rozdzielczości jest reprezentowany przez strukturę typu `struct hrtimer`. Od 2017 roku[4] w ramach aktywacji jest on dodawany do drzewa czerwono-czarnego lub do kolejki. Kiedy zegar sprzętowy powiązany z licznikami wysokiej rozdzielczości generuje przerwanie, to funkcje liczników w drzewie, które należy uruchomić są wywoływane przez procedurę obsługi tego przerwania (górną połówkę). Następnie sprawdza ona, czy należy uruchomić pierwszy licznik w kolejce. Jeśli tak, to wyzwala przerwanie programowe, które uruchamia funkcję związaną z tym licznikiem i sprawdza pozostałe liczniki. Struktura typu `struct hrtimer` oprócz pól, które pozwalają ją dodać do listy i drzewa czerwono-czarnego, zawiera również składową `expires`, która przechowuje czas, po którym licznik powinien zostać uruchomiony, wyrażony w nanosekundach. Innym polem zawartym w tej strukturze jest `function`, które przechowuje adres funkcji licznika, która implementuje czynności wykonywane po jego uruchomieniu.

[4] W tym roku została dodana poprawka autorstwa Anny Marii Gleixner.

API liczników czasu wysokiej rozdzielczości

Prototyp tej funkcji jest następujący:

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

Nazwa funkcji nie musi być taka sama, jak ta użyta w prototypie. Wyliczenie, które definiuje typ wartości zwracanej przez tę funkcję ma dwa elementy: `HRTIMER_NORESTART`, który oznacza, że licznik nie będzie automatycznie odnowiony i `HRTIMER_RESTART`, który oznacza, że licznik będzie cykliczny. W tym ostatnim przypadku funkcja musi modyfikować wartość pola `expires` struktury typu `struct hrtimer` wskazującej na tę funkcję. Dlatego otrzymuje ona jako argument adres tej struktury. Wspomniana modyfikacja musi się odbywać tylko przy pomocy funkcji `hrtimer_forward()`. Jedno z pól struktury typu `struct hrtimer` przechowuje również bieżący stan licznika wyrażony jedną z następujących stałych:

`HRTIMER_STATE_INACTIVE` licznik jest nieaktywny,

`HRTIMER_STATE_ENQUEUED` licznik jest aktywny i oczekuje na uruchomienie.

API liczników czasu wysokiej rozdzielczości

Funkcje obsługujące liczniki wysokiej rozdzielczości przypominają te od liczników niskiej rozdzielczości. Funkcja `hrtimer_init()` inicjuje strukturę typu `struct hrtimer`. Jeden z jej argumentów określa, czy wartość pola `expires` wyraża czas bezwzględny (tzn. mierzony względem momentu uruchomienia komputera), czy względny (mierzony względem momentu aktywacji licznika). Za aktywację licznika odpowiada funkcja `hrtimer_start()`. Usuwaniem licznika zajmują się dwie funkcje: `hrtimer_cancel()` i `hrtimer_try_to_cancel()`. Obie zwracają 0 jeśli licznik był już nieaktywny lub 1 jeśli był aktywny. Ostatnia z nich zwraca także -1, jeśli funkcja licznika była już w trakcie wykonania. Reaktywacji licznika można dokonać funkcją `hrtimer_restart()`. Często te liczniki są wykorzystywane do budzenia wątku, który jest umieszczony w kolejce oczekiwania. Jądro zapewnia strukturę typu `struct hrtimer_sleeper`, która pozwala powiązać taki licznik z deskryptorem wątku i ułatwia obsługę opisywanego przypadku. Więcej szczegółów dotyczących API liczników wysokiej rozdzielczości znajduje się w 7. instrukcji laboratoryjnej.

Opóźnianie wykonania

Najprostszym sposobem opóźnienia wykonania kodu w przestrzeni jądra jest użycie aktywnego oczekiwania. W jądrze Linuksa można je zaimplementować poprzez odczyt w pętli zmiennej `jiffies` i porównywanie jej bieżącej wartości z oczekiwaną liczbą przerwań zegarowych. Do porównywania tych dwóch wartości można użyć np. makra `time_before`. Zmienna `jiffies` została przygotowana do tego zastosowania. Programiści w jej deklaracji użyli słowa kluczowego `volatile`, aby uniemożliwić kompilatorowi przeprowadzenie optymalizacji kodu polegającej na umieszczeniu wartości tej zmiennej w rejestrze. Taka zmiana spowodowałaby, że wspomniana pętla zawsze odczytywałaby tę samą wartość i nigdy się nie kończyła. Opisywane słowo kluczowe powoduje, że wartość zmiennej `jiffies` jest odczytywana bezpośrednio z pamięci. Aktywne oczekiwanie często jest postrzegane jako antywzorzec, zatem zaleca się aby we wspomnianej pętli wywoływać funkcję `condition_reached()`, która powoduje przeszeregowanie procesów.

Opóźnianie wykonania

Jeśli opóźnienie ma być krótkie, to można użyć jednej z funkcji: `udelay()`, `mdelay()` lub `ndelay()`. Pierwsza z nich opóźnia wykonanie o zadaną liczbę mikrosekund, druga o określoną liczbę milisekund, a trzecia o podaną liczbę nanosekund. Wszystkie z nich stosują aktywne oczekiwanie. Funkcja `udelay()` wykonuje pętlę, której liczba iteracji jest określana przez wartość argumentu tej funkcji i przez liczbę tak zwanych *BogoMIPS*. Ta ostatnia wielkość jest ustalana w trakcie startu systemu i określa ile razy w zadanym przedziale czasowym CPU może wykonać pewne rozkazy. Od wersji 3.6 jądra, w przypadku komputerów z procesorami o architekturze ARM, funkcja `udelay()` używa specjalnego sprzętowego licznika czasu. W przypadku innych platform sprzętowych jej implementacja nie uległa zmianie. Funkcja `mdelay()` wywołuje funkcję `udelay()`.

Opóźnianie wykonania

W przypadku długich opóźnień używanie aktywnego oczekiwania nie jest dobrym pomysłem. Zamiast niego można zastosować funkcję `schedule_timeout()`, która usypia wątek i budzi go po upływie zadanego czasu. Zanim funkcja zostanie wywołana należy zmienić stan wątku na `TASK_UNINTERRUPTIBLE` lub `TASK_INTERRUPTIBLE` lub `TASK_KILLABLE`. Aby uprościć jej użycie programiści jądra Linuksa zdefiniowali trzy inne funkcje:

`schedule_timeout_killable()` nadaje wątkowi stan `TASK_KILLABLE` i wywołuje funkcję `schedule_timeout()`,

`schedule_timeout_interruptible()` jak wcześniej, ale nadaje wątkowi stan `TASK_INTERRUPTIBLE`,

`schedule_timeout_uninterruptible()` jak wcześniej, ale nadaje wątkowi stan `TASK_UNINTERRUPTIBLE`.

Opóźnianie wykonania

Ostatnim sposobem na opóźnienie wykonania kodu, który zostanie wymieniony w tym wykładzie jest skorzystanie ze struktury typu `struct hrtimer_sleeper` i użycie licznika wysokiej rozdzielczości do obudzenia wątku po upływie określonego czasu.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!