Wykład dziewiąty

Pomiar czasu i synchronizacja względem czasu w Linuksie

Wywołania wielu funkcji jądra są uzależnione nie od wystąpienia określonych zdarzeń lecz od upływu czasu, dlatego jądro systemu musi mieć mechanizm pozwalający na jego pomiar. Ten pomiar jest również potrzebny aplikacjom przestrzeni użytkownika. Te ostatnie mogą wymagać informacji na temat bieżącej daty i czasu, jak również mogą dokonywać pomiaru czasu wykonania pewnych czynności.

Generowanie taktów i pomiar upływu czasu

Głównym mechanizmem wyznaczającym dla jądra upływ kolejnych chwil jest *zegar systemowy*. Mechanizm ten generuje w określonych odstępach czasu przerwanie zwane przerwaniem zegarowym, które jest obsługiwane za pomocą specjalnej procedury obsługi przerwania. Zegar systemowy pracuje na podstawie pobudzeń zegarowych dostarczanych przez generator impulsów taktujących procesor lub przez inne urządzenia tego typu. Współczesne platformy sprzętowe mogą być wyposażone w kilka różnych urządzeń zegarowych, które mogą pełnić rolę zegara systemowego. Przykładem są komputery z procesorami z rodziny x86. Tradycyjnie rolę zegara systemowego w takich komputerach odgrywa układ PIT (ang. *Programmable Interval Timer*), ale nowsze komputery tej klasy są wyposażone także w układ HPET (ang. *High Precision Event Timer*), który pozwala na pracę z większą częstotliwością. Dodatkowo, urządzenia zegarowe w różnych platformach sprzętowych mogą się różnić obsługą i precyzją. Dlatego też, w trakcie rozwoju serii 2.6 jądra systemu, stworzono warstwę abstrakcji, która pozwala ukryć te różnice i ujednolicić obsługę zegarów. Ta warstwa abstrakcji tworzy za pomocą istniejących zegarów sprzętowych trzy rodzaje wirtualnych urządzeń zegarowych:

- źródła czasu (ang. *clock source*) - urządzenia, które odmierzają czas i pozwalają na jego odczyt,
- urządzenia zdarzeń czasowych (ang. *clock event devices*) - urządzenia, które generują sygnał oznaczający, że upłyną określony odcinek czasu,
- urządzenia odliczające (ang. *tick devices*) - urządzenia zbudowane na bazie urządzeń zdarzeń czasowych pozwalające określić, czy sygnał o danym zdarzeniu ma być powtarzalny, czy tylko jednorazowy.

Każde ze źródeł czasu i urządzeń zdarzeń czasowych ma przypisaną jakość. Zazwyczaj to, które ma największą miarę jakości przyjmowane jest za domyślne. Domyślne urządzenie zdarzeń czasowych pełni rolę zegara systemowego. Częstotliwość generowania kolejnych przerwań przez zegar systemowy jest określona stałą HZ. Znając częstotliwość można wyznaczyć długość okresu z jakim działa zegar systemowy. W większości współczesnych platform sprzętowych stała HZ jest równa 100 (100 herców, czyli okres = 10ms). W przypadku komputerów opartych na procesorach x86 wartość tej stałej zmieniała się w trakcie rozwoju jądra serii 2.4 i 2.6. Przez pewien czas jej wartość wynosiła 1000 herców (okres = 1 ms). Obecnie jej wartość wynosi 250 herców (4 ms). Stała ta ma oczywiście wpływ na szybkość działania systemu. Zwiększenie jej ma wiele zalet, ale również może powodować problemy. Do zalet należy zaliczyć zwiększenie rozdzielczości przerwania zegarowego i zwiększenie dokładności sterowania czasem, co pociąga za sobą lepszą obsługę wszystkich zdarzeń zależnych od czasu. Niestety, w starszych wersjach jądra nie można było tej stałej w dowolny sposób modyfikować. Należy również uwzględnić fakt, że zwiększenie tej stałej pociąga za sobą zwiększenie częstotliwości występowania przerwania zegarowego i częstszego wywoływania jego procedury obsługi. Poza tym mogą wyniknąć problemy związane z obsługą sieciowych źródeł czasu działających w oparciu o protokół NTP. Główną przesłanką dla zwiększenia wartości HZ była poprawa działania aplikacji mulitmedialnych, głównie z zakresu audio. Okazało się, że to jednak niewystarczającym zabiegiem. Dlatego nastąpił powrót do niższej wartości, która obecnie wynosi 250, a problem programów dźwiękowych rozwiązano za pomocą liczników czasu wysokiej rozdzielczości. Nowsze wersje jądra pozwalają także na dynamiczne określanie częstotliwości zegara. Jest to konieczne w przypadku niektórych platform sprzętowych, takich jak systemy wbudowane, czy choćby laptopy pracujące w trybie oszczędności energii. Ponieważ każdy sygnał (przerwanie) czasowy wymaga obsługi, która niesie za sobą pewien wydatek energii, to aby ją zaoszczędzić komputer może ustawić zegar czasowy, tak aby nie generował sygnałów w stałych odstępach czasowych, ale dopiero wtedy gdy będą potrzebne, tzn. wtedy gdy trzeba będzie wykonać jakąś czynność. Liczbę taktów zegara od momentu uruchomienia systemu jądro przechowuje się w zmiennej *jiffies*, której wartość jest zwiększana o jeden podczas każdego taktu zegara. Czas pracy systemu (*ang. uptime*) mierzony jest jako iloraz *jiffies/HZ*. W zmienna *jiffies* przechowuje wartości będące 64 – bitowymi liczbami naturalnymi. We wcześniejszych wersjach jądra była ona 32 – bitowa. Skutkowało to problemami w wersjach jądra, w których zmieniana była wartość stałej HZ dla 32-bitowych platform sprzętowych opartych na procesorach o architekturze x86. Przy HZ=100 przepełnienie *jiffies* następowało co 497 dni. Kiedy w wersjach jądra 2.4 i 2.6 zaczęto stosować stałą HZ=1000, to okres ten skrócił się do 49,7 dnia, dlatego też twórcy jądra zastosowali rozwiązanie polegające na „nałożeniu" 32 – bitowej zmiennej *jiffies* na zmienną *jiffies_64*, która ma rozmiar 64 bitów. Odwołanie się więc do zmiennej *jiffies* w architekturach 32 – bitowych dawało wartość młodszego słowa zmiennej *jiffies_64*, a w architekturach 64 – bitowych jej pełną wartość. Dostęp do pełnej wartości tej zmiennej możliwy był w obu przypadkach poprzez funkcję *get_jiffies_64()*. Ta funkcja jest dostępna również we współczesnych wersjach jądra. Aby uniknąć problemów, jakie może spowodować przepełnienie zmiennej *jiffies*[1] programiści jądra wprowadzili makrodefinicje, uwzględniające to zjawisko przy pomiarze czasu. Są to *time_after*, *time_before*, *time_after_eq* i *time_before_eq*. Do wszystkich z nich przekazywane są dwa argumenty: (najczęściej) bieżąca wartość zmiennej *jiffies* i wartość, z którą jest ona porównywana. Pierwsza makrodefinicja zwraca wartość różną od zera, jeśli pierwszy argument reprezentuje moment występujący po momencie reprezentowanym przez drugi argument. Druga makrodefinicja działa odwrotnie. Dwie pozostałe dodatkowo uwzględniają przypadek, kiedy oba przekazane makrodefinicjom argumenty są równe.
Aplikacje użytkownika, pisane są z założeniem, że zmienna HZ ma wartość 100. Niestety, nie jest to prawdą dla wszystkich platform sprzętowych. Oprócz komputerów z procesorami o architekturze x86 również komputery z procesorami Alpha mają inną wartość tej stałej, w ich przypadku 1024 herce. Aby programy użytkowe mogły działać bez przeszkód na wszystkich platformach sprzętowych, wprowadzono stałą USER_HZ, która pełni rolę stałej HZ dla przestrzeni użytkownika. Ma ona zastosowanie głównie podczas skalowania wartości *jiffies* dla przestrzeni użytkownika.
Jądro obsługuje również mechanizm *zegara czasu rzeczywistego (RTC)* nazywanego także *zegarem czasu systemowego*, z którego pracy korzystają głównie aplikacje użytkowników. Zegar ten przechowuje i aktualizuje informacje o bieżącej dacie i godzinie. We wcześniejszych wersjach jądra jego zawartość była odczytywana i umieszczana w zmiennej *xtime* podczas rozruchu systemu. Jądro nie odczytywało już więcej zawartości zegara czasu rzeczywistego, ale samo aktualizowało zawartość tej zmiennej i ewentualnie mogło aktualizować zawartość samego RTC. Typ zmiennej *xtime* był strukturą posiadającą dwa pola. Pierwsze z nich przechowywało liczbę sekund, które upłynęły od 1 stycznia 1970 roku[2], a drugie liczbę nanosekund, które upłynęły od początku bieżącej sekundy. Zapis i odczyt tej zmiennej wymagał założenia blokady sekwencyjnej *xtime_lock*. W nowszych wersjach jądra rolę tej struktury przejęły dwie zmienne. Jedna typu *struct tk_read_base*, która przechowuje w jednym z pól wspomnianą wcześniej liczbę nanosekund oraz zmienna typu *struct timekeeper*, która przechowuje w jednym ze swych pól wspomnianą liczbę sekund. Obie te struktury pełnią również dodatkowe funkcje w jądrze systemu. Dla aplikacji użytkownika wskazania zegara czasu rzeczywistego dostępne są za pomocą wywołania *gettimeofday()*. Aktualizacja tego zegara, podobnie jak zmiennych *jiffies* i *jiffies_64* odbywa się w ramach obsługi przerwania zegarowego. Procedura obsługi tego przerwania jest również odpowiedzialna za uaktualnianie statystyk procesów użytkownika i systemowych.

Liczniki niskiej rozdzielczości

Liczniki, zwane licznikami dynamicznymi lub licznikami jądra, pozwalają opóźnić wykonanie określonych czynności o ustalony przedział czasu, począwszy od chwili bieżącej. Są dwie kategorie takich liczników: liczniki o niskiej i wysokiej rozdzielczości. Te pierwsze nie są mechanizmem precyzyjnym i mogą zdarzać się im niedokładności rzędu długości trwania okresu zegara systemowego, więc nie powinny one być stosowane do zadań czasu rzeczywistego, niemniej w większości przypadków sprawdzają się one w innych zastosowań. Należy również pamiętać, że liczniki te nie są cykliczne, tzn. po upływie określonego dla nich czasu są uruchamiane, zwalniane i nie jest automatycznie wznawiana ich praca. Liczniki dynamiczne są reprezentowane strukturą *timer_list*. Aby skorzystać z licznika należy zadeklarować zmienną tego typu, zainicjować ją przy pomocy *init_timer*, wypełnić bezpośrednio pola *expires, data* i *function*, oraz uaktywnić przy pomocy *add_timer()*. Pierwsze z wymienionych pól określa czas, po którym ma zostać wywołana związana z licznikiem funkcja, drugie jest argumentem wywołania tej funkcji, a trzecie zawiera jej adres. Wymieniona wcześniej funkcja musi mieć następujący prototyp:

void my_timer_function(unsigned long data);

Ponieważ funkcja ta może być związana równocześnie z kilkoma licznikami, to wartość parametru *data* pozwala jej ustalić na rzecz którego została wywołana. Do inicjacji wspomnianych pól struktury *timer_list* można użyć także makra *setup_timer*. Począwszy od wersji 4.15 jądra makrodefinicje *init_timer* i *setup_timer* zostały zastąpione makrodefinicją *timer_setup*, która wykonuje czynności obu z nich. Jeśli zaistnieje konieczność modyfikacji momentu wyzwolenia licznika to można użyć

1 Dotyczy to głównie jej wersji 32-bitowej.
2 Tę datę przyjmuje się za początek „epoki Uniksa".

w tym celu funkcji *mod_timer()*. Funkcja ta może zostać użyta zarówno dla aktywnych, jak i nieaktywnych liczników, przy czym te ostatnie są przez nią aktywowane. Jeśli chcemy usunąć aktywny licznik możemy to uczynić funkcją *del_timer()*. W systemach wieloprocesorowych, celem uniknięcia sytuacji hazardowych należy użyć *del_timer_sync()*. We wszystkich systemach należy unikać innej modyfikacji aktywnych liczników, niż przez funkcję *mod_timer()*. Należy także pamiętać, aby zabezpieczyć zasoby do których dostęp współbieżny mają funkcje wywoływane przez liczniki oraz inne części kodu. Do określenia tego, czy licznik jest aktywny (oczekuje na wykonanie) można użyć funkcji *timer_pending()*, która zwraca 1, jeśli dany licznik jest aktywny. Liczniki są powiązane w listę. Funkcje przez nie wywoływane są uruchamiane w kontekście dolnej połówki, z poziomu przerwania programowego (czyli w kontekście przerwania), po zakończeniu obsługi przerwania zegarowego. Aby uniknąć sortowania listy liczników i kosztownego przeglądania jest ona podzielona na pięć grup, pod względem czasu, po jakim trzeba będzie wywołać funkcje zgromadzonych na niej liczników.

Liczniki wysokiej rozdzielczości

W zastosowaniach, gdzie nie wystarczają liczniki o niskiej rozdzielczości (głównie wspomniane wcześniej aplikacje multimedialne) należy zastosować liczniki wysokiej rozdzielczości. Działają one w oparciu o zegary oferujące pomiar czasu z rozdzielczością nanosekundową. Linux przyjmuje, że są co najmniej dwa takie zegary - zegar monotoniczny i zegar czasu rzeczywistego. Pierwszy odlicza czas w równych odstępach od momentu uruchomienia komputera. Drugi może być przestawiany w trakcie działania systemu komputerowego, stąd jądro uwzględnia w takiej sytuacji odpowiednie poprawki pomiaru czasu. W systemach wieloprocesorowych na każdy procesor przypada jedna taka para zegarów. Struktury typu *struct hrtimer*, opisujące liczniki wysokiej rozdzielczości, są zorganizowane jednocześnie w drzewo czerwono-czarne i listę. Lista jest prostsza do przeglądania, ale musi być posortowana. Ta ostatnia operacja efektywniej jest realizowana przy pomocy drzewa czerwono-czarnego. Kiedy urządzenie zdarzeń czasowych powiązane z obsługą liczników wysokiej precyzji wygeneruje przerwanie, to liczniki, których czas realizacji został osiągnięty są wykonywane w ramach dolnej połówki tego przerwania, która jest przerwaniem programowym. Struktura *hrtimer* zawiera pola, które pozwalają jej instancje umieszczać zarówno w drzewie czerwono-czarnym, jak i na liście liniowej. Podobnie jak *timer_list* zawiera ona także pole *expires*, które określa w nanosekundach czas, po którym ma zostać wykonany licznik, czyli wykonana związana z nim funkcja. Adres tej funkcji jest przechowywany w polu *function*. Jej prototyp jest następujący:

*enum hrtimer_restart my_hrtimer(struct hrtimer *);*

Ta funkcja może zwrócić dwie wartości: HRTIMER_NORESTART, która oznacza, że wykonanie licznika nie będzie automatycznie ponawiane i HRTIMER_RESTART, która oznacza, że licznik będzie ponownie wykonany. Aby tak się stało, funkcja musi odpowiednio zmodyfikować pole *expires* struktury *hrtimer*. Może to uczynić, gdyż wskaźnik na strukturę licznika, z którym ona jest związana jest jej przekazywany przez parametr. Aby uprościć manipulowanie tym polem programiści jądra Linuksa stworzyli funkcję *hrtimer_forward()*. Stan licznika wysokiej rozdzielczości jest także przechowywany w tej strukturze i określają go następujące stałe:

- HRTIMER_STATE_INACTIVE - licznik nie jest aktywny,
- HRTIMER_STATE_ENQUEUED - licznik jest zaszeregowany i czeka na wykonanie,

Nazwy i działanie funkcji obsługujących liczniki wysokiej rozdzielczości są podobne do funkcji i makrodefinicji obsługujących liczniki niskiej rozdzielczości. Funkcja *hrtimer_init()* inicjuje instancję struktury *hrtimer*. Przyjmuje ona argument określający, czy czas w polu *expires* ma być traktowany jako bezwzględny (mierzony od początku pomiaru czasu w komputerze), czy jako względny (mierzony względem chwili aktywacji licznika). Funkcja *hrtimer_start()* aktywuje licznik do wykonania. Funkcje *hrtimer_cancel()* i *hrtimer_try_to_cancel()* anulują licznik. Obie zwracają wartość 0, jeśli licznik nie był aktywny i 1, jeśli był. Dodatkowo, druga funkcja zwraca -1, jeśli licznik był w trakcie wykonania. Ponownej aktywacji anulowanego licznika można dokonać za pomocą funkcji *hrtimer_restart()*.
Często liczniki jądra są używane do tego, aby wybudzić proces lub wątek, który został dodany do kolejki oczekiwania, a jego wykonanie zostało zawieszone. Jądro dostarcza struktury o nazwie *hrtimer_sleeper*, która ułatwia obsługę takiego przypadku, wiążąc strukturę licznika z deskryptorem zadania, które ma być obudzone.
Jeśli platforma sprzętowa nie dostarcza zegarów o rozdzielczości nanosekundowej, lub jeśli ich obsługa nie jest włączona na etapie kompilacji jądra, to nadal programiści mogą używać liczników wysokiej rozdzielczości, ale są one wykonywane przez ten sam mechanizm, co liczniki niskiej rozdzielczości. Zatem, w takiej sytuacji nie ma rozróżnienia między działaniem obu typów liczników.

Opóźnianie wykonania

Najprostszym sposobem opóźnienia wykonania kodu jest umieszczenie w nim pętli aktywnego oczekiwania. W Linuksie do określenia warunku zakończenia takiej pętli można skorzystać np. z makrodefinicji *time_before()*. Wymaga to oczywiście odczytu zmiennej *jiffies*, co może rodzić obawy, czy podczas kompilacji nie zostanie ta operacja zoptymalizowana w niepożądany sposób. Aby zapobiec takim sytuacjom, programiści jądra zadeklarowali ją jako *volatile*, co skutecznie powstrzymuje kompilator od optymalizowania operacji na tej zmiennej. Zalecanym jest, aby w takiej pętli wywołać funkcję *cond_reached()*, która powoduje przeszeregowanie zadań jeśli zachodzi taka potrzeba. Jeśli czas opóźnienia ma być krótki, możemy posłużyć się funkcjami *udelay(), mdelay()* i *ndelay()*. Pierwsza opóźnia wykonanie na określoną liczbę mikrosekund, wykonując określoną liczbę razy pętlę złożoną z odpowiednich instrukcji. Liczba iteracji tej pętli jest określona wartością zmiennej BogoMIPS obliczaną przy uruchamianiu jądra. Począwszy od wersji 3.6 jądra dla niektórych platform sprzętowych, bazujących na procesorach ARM, działanie funkcji udelay() jest zależne od specjalnego zegara sprzętowego. Funkcja mdelay() korzysta do odliczania czasu z ... funkcji *udelay*(). Ostatnia z wymienionych funkcji pozwala na opóźnienie wykonania o określoną liczbę nanosekund. Ponieważ aktywne oczekiwanie jest nieefektywne i niewskazane, wykonanie kodu jądra można opóźniać używając do tego funkcji *schedule_timeout()* lub pokrewnych, które ustawiają stan procesu na TASK_INTERRUPTIBLE, TASK_UNINTERRUPTIBLE lub TASK_KILLABLE.