Konferencja naukowa KNWS'04
„Informatyka - sztuka czy rzemiosło"
23-25 czerwca 2004, Czocha

RTLINUX/BSD JAKO SYSTEM OPERACYJNY CZASU RZECZYWISTEGO

Kamil Mielcarek

Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski
65-246 Zielona Góra, ul. Podgórna 50

e-mail: K.Mielcareko@iie.uz.zgora.pl

STRESZCZENIE

Wprowadzenie do systemów czasu rzeczywistego z uwzględnieniem systemów opartych na jądrze Linuxa i BSD. Systemy RTLinux/BSD mogą być używane jako systemy RT w rozwiązaniach generowanych w ramach automatycznego współprojektowania sprzętowo-programowego. Ich znajomość może zwiększyć możliwości tworzonego automatycznie układu.

1. WPROWADZENIE

Rosnące zainteresowanie oprogramowaniem Open Source w ostatnim czasie skłania do zapoznania się z systemami czasu rzeczywistego realizowanymi na ogólnodostępnych i wolnych od opłat rozwiązaniach. Prym wiodą tu systemy oparte o jądro systemu Linux oraz systemy z Berkeley Software Distribution. Dostępność rozwiązań predestynuje systemy do masowych zastosowań nawet w bardzo wymagających sytuacjach. Zasadność wyboru można poprzeć analizując zachowanie rynku, gdzie pojawiają się komercyjne zastosowania systemów opracowanych na bazie jądra Linuksa. Mankamentem może być to, że dedykowane systemy czasu rzeczywistego (ang. real-time system) dość często objęte są dość wysokimi opłatami. Firma FSM Labs oferuje kilka wersji systemu RTLinux (podstawowa jest bezpłatna, oparta na licencji GPL) zależnie od wymagań użytkownika (jednocześnie oferując te same rozwiązania dla systemów BSD). RTLinux jest oczywiście systemem operacyjnym (a dokładniej RTLinux + Linux), ale to właśnie system operacyjny jest "sercem" systemu czasu rzeczywistego i to głównie od niego zależy spełnianie wymogów czasowych. System RTLinux był tworzony z myślą o prostocie, spełniając jednocześnie bardzo restrykcyjne (twarde) wymogi systemu czasu rzeczywistego. Jak twierdzi sam twórca - Victor Yodaiken: *„RTLinux daje wewnętrzne środowisko pthreads, dzięki niemu możliwe było zintegrowanie POSIXowych wątków i wywołań sygnałów z istniejącym zbiorem operacji. Dla programistów zaprzyjaźnionych z pthreads i POSIXowym programowaniem nie będzie trudne nauczenie się, jak pisać programy RT. Nie powinno być też trudne przerobienie sterownika Linux aby pracował z RTLinux. Chodziło o to, żeby nie zmuszać użytkowników do programowania [jądra] Linuksa. Programista nie musi nawet znać struktury jądra Linuksa, aby pisać programy real-time dla RTLinuksa."*

Gigant z Redmond (Microsoft) nie pozostając w tyle oferuje dwa systemy operacyjne "real-time". Jest to Windows CE .NET oraz Windows XP Embedded, gdzie Windows CE .NET jest twardym systemem operacyjnym czasu rzeczywistego, natomiast Windows XP Embedded nie jest zasadniczo systemem operacyjnym czasu rzeczywistego, ale użytkownik może łatwo dodać do niego możliwości "real-time" optymalizując Windows XP Embedded tak, aby spełniał on wymogi real-time, używając dostępnych rozwiązań firm trzecich. Na rynku znajdują się też inne alternatywy dla RTLinuksa, jak np. KURT (GPL) czy QNX oraz komercyjne - przewyższające możliwości systemu z FSM Labs. Wielu użytkowników jednak wybiera właśnie ten system ze względu na cenę i łatwość obsługi.

Systemy czasu rzeczywistego powszechnie stosuje się w dziedzinach takich jak: aeronautyka, multimedia, telekomunikacja, astronomia, robotyka, przemysł, medycyna, kontrola i pomiary

2. PODSTAWY SYSTEMÓW CZASU RZECZYWISTEGO

Przystępując do omawiania zagadnień systemów czasu rzeczywistego należy pamiętać o istniejącej różnicy pojęć system i system operacyjny czasu rzeczywistego.

System czasu rzeczywistego (ang. real-time system - zgodnie z IEEE/ANSI) to system komputerowy, w którym obliczenia są wykonywane współbieżnie z procesem zewnętrznym (otoczenie) w celu sterowania, nadzorowania lub terminowego reagowania na zdarzenia (występującego w otoczeniu).

System operacyjny czasu rzeczywistego (ang. real-time operating system) - jest tylko jednym z elementów kompletnego systemu. Musi on dostarczać odpowiedniej funkcjonalności, aby cały system mógł sprostać stawianym wymaganiom.

Definicja systemu czasu rzeczywistego często uzupełniana jest przez sformułowanie pożądanych cech systemu, do których należą:
ciągłość działania - system czasu rzeczywistego powinien pracować bez przerw
zależność od otoczenia - system jest uzależniony od zdarzeń i danych generowanych przez proces zewnętrzny (otoczenie)
współbieżność - otoczenie systemu składa się z obiektów działających współbieżnie, generujących zdarzenia lub dane wymagające obsługi
przewidywalność - system musi zachowywać się deterministyczne reagując na zdarzenie według określonych wymagań
punktualność - odpowiedź systemu musi być dostarczona do otoczenia w odpowiednich momentach czasowych

Systemy czasu rzeczywistego ze względu na limity czasowe można podzielić na dwie zasadnicze grupy:

Rygorystyczny (twardy) system czasu rzeczywistego - (ang. hard real-time system) gwarantuje terminowe (przed upływem czasu lub w dokładnie wyznaczonych chwilach) wypełnianie krytycznych zadań. System zawiedzie, jeśli któryś z limitów czasowych nie zostanie dotrzymany.

Miękki system czasu rzeczywistego - (ang. soft real-time system) może tolerować zauważalne odchylenia czasowe w usługach dostarczanych przez system operacyjny, takich jak przerwania, taktowanie czy planowanie.

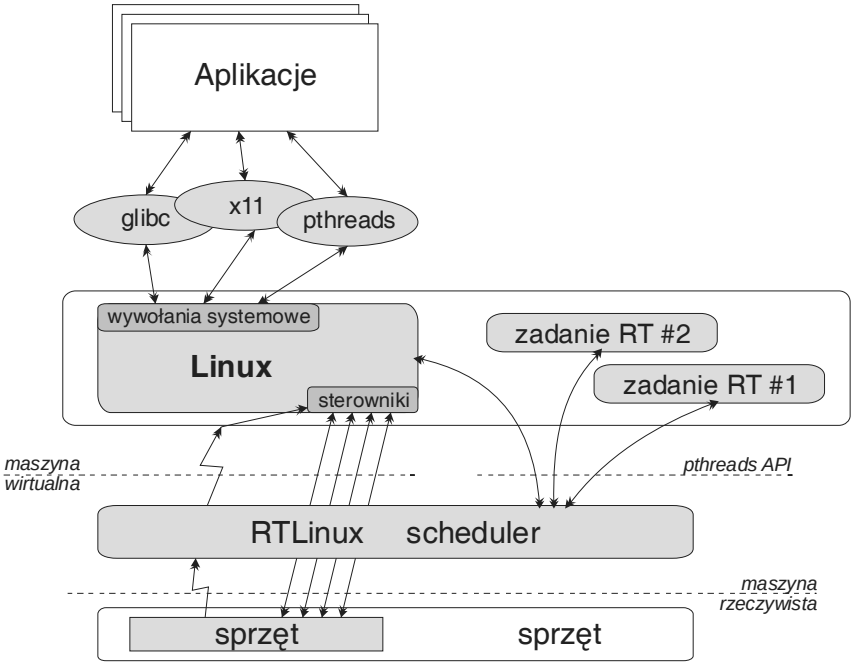
Dopuszcza się także istnienie **systemów mieszanych** nazywanych solidnymi (ang. firm real-time system), gdzie poszczególne wymagania czasu rzeczywistego mogą mieć zarówno charakter twardy, jak i miękki.

3. ARCHITEKTURA SYSTEMU RTLINUX

Istnieją dwa podejścia do zapewnienia działania systemu Linux: poprzez ulepszenie wywłaszczania jądra systemu oraz przez dodanie nowej warstwy poniżej jądra z zapewnieniem pełnej kontroli przerwań i możliwości procesora. Oba podejścia (odpowiednio) znane są pod angielskimi nazwami „*preemption improvement*" oraz "*interrupt abstraction*". W systemie zastosowano drugą z metod, gdzie RTLinux dodaje warstwę wirtualnego sprzętu między standardowym jądrem Linuksa a rzeczywistymi urządzeniami (rys. 1). RTLinux implementuje kompletny i przewidywalny system operacyjny czasu rzeczywistego bez interfejsu z Linuksa. Wątki są wykonywane bezpośrednio przez z góry narzucony harmonogram. Jądro Linuksa i wszystkie normalne procesy są zarządzane przez proces nadrzędny (RTLinux sheduler) jako zadanie tła. Dzięki temu uzyskano możliwość uruchomienia systemu ogólnego przeznaczenia w małym i przewidywalnym systemie operacyjnym czasu rzeczywistego. Dało to możliwość uruchomienia kodu Linuksa lub np. systemów z rodziny BSD (w zależności od wersji systemu z FSM Labs).

Uzyskanie pełnej kontroli przez wirtualizację urządzeń uzyskano poprzez trzy podstawowe modyfikacje. Warstwy RTLinuxa przejmują wszystkie przerwania sprzętowe, przerwania nie kontrolowane przez wątki RT są przekierowywane do systemu Linux. RTlinux przejmuje również kontrolę nad zegarem sprzętowym implementując zegar wirtualny dostępny dla zwykłego jądra systemu. Ostatnią modyfikacją jest podmiana wywołań *cli* oraz *sti* (włączanie i wyłączanie flagi przerwań) z jądra Linuksa, aby uniknąć rzeczywistego wyłączenia na rzecz odpowiednika wirtualnego.

Umieszczenie środowiska wykonawczego poniżej jądra Linuksa powoduje, że nie jest możliwe używanie jego usług z uwagi na możliwość wystąpienia blokady. Rozwiązaniem jest podział systemu na dwie części i wykonanie niezależnie fragmentu odpowiedzialnego za rygorystyczny i miękki system czasu rzeczywistego. Daje to możliwość uzyskania rygorystycznego systemu czasu rzeczywistego będącego jednocześnie typowym systemem.



Rys. 1. Architektura systemu RTLinux

Poniższy przykład prezentuje sposób komunikacji między programami system RT a programami uruchomionymi w systemie Linux. Przykład składa się z dwóch programów: pierwszy jest to cześć RT, mająca za zadanie zapełnić kolejki FIFO tak wielką ilością danych jak to jest możliwe, druga część uruchamiana z poziomu zwykłego systemu odczytuje dane.

Do kompilacji programu potrzebny jest plik make rtl.mk obecny w katalogu z kodem źródłowym. Do poprawnego uruchomienia niezbędne jest wystartowanie jądra RTLinux oraz obecności modułów rtl_time, rtl_sched, rtl_posixio, rtl_fifo.

Kolejność postępowania (wykonanie jako root): *modprobe rtl_fifo*

Weryfikacja czy wszystkie moduły zostały załadowane: *lsmod*

Kompilacja programu: *gcc -o read read.c*

Uruchomienie programów: *insmod fifo.o; ./read*

Zatrzymanie programów czytającego i zapisującego: *Ctrl+c*, *rmmod fifo*

fifo.c - część RT

```
#include <rtl.h>
#include <rtl_sync.h>
#include <time.h>
#include <rtl_fifo.h>

#define BUFSIZE 1024
char buf[BUFSIZE];
```

Makefile

```
all: fifo.o
include rtl.mk
clean: rm -f *.o
```

```
pthread_t thread;

void * start_routine(void *arg)
{
  struct sched_param p;
  int status;
  p.sched_priority = 1;
  pthread_setschedparam(pthread_self(), SCHED_FIFO, &p);
  pthread_make_periodic_np(pthread_self(),gethrtime(),10000000); //10ms
  while(1)
  {
    status = rtf_put(0, buf, 1024);  //zapis tak często jak to możliwe
    if (status <= 0)                 //1kB danych do rt_fifo
      pthread_wait_np();             //jeśli fifo pełne: czekaj 10ms
    }
  return 0;
}

int init_module(void)
{
  rtf_create(0, 1024*1024);  //stworzenie rt fifo o rozmiarze 1MB
  return pthread_create (&thread, NULL, start_routine, 0);
}

void cleanup_module(void)
{
  rtf_destroy(0);
  pthread_delete_np (thread);
}
```

read.c - część odbiorcza

```
#include <stdio.h>
#include <fcntl.h>

#define BUFSIZE 1024
char buf[BUFSIZE];

int main()
{
  int fd0;
  int n, c, d;
  c = d = 0;
  if ((fd0 = open("/dev/rtf0", O_RDONLY)) < 0)
  {
    fprintf(stderr, "Error opening /dev/rtf0\n");
    exit(1);
  }
  while (1 == 1)
  {
    n = read(fd0, buf, BUFSIZE);
    c++;
    if (c == 10000)
    {
      printf("odczyt 10000 * 1024 bajtow z koleki fifo - ");
      c = 0;
      d++;
      printf("nr. %d\n", d);
    }
  }
}
```

Tab. 1. Wyniki testu przeprowadzone na przykładowym programie

Processor	Memory (RAM)	przepustowość MB/s
Pentium II / 300 MHz	64 MB	69
486 DX2 / 66MHz	32 MB	6.5

W systemach czasu rzeczywistego czas jest liczony z dokładnością do mikrosekund (μs). W praktyce np. pesymistyczne opóźnienie obsługi przerwania przez RTLinux na komputerze PC 486/33MHz plasuje się poniżej 30 μs. Taktowanie 33MHz to ok. 1000 taktów na 30 μs - jest to bardzo bliskie limitu sprzętowego.

Przy wyznaczaniu limitów czasowych mamy na myśli dwie rzeczy: opóźnienia obsługi przerwań sprzętowych (tzn. czas od momentu wystąpienia zdarzenia na urządzeniu do początku jego obsługi przez sterownik) oraz wahania planisty (ang. scheduler jitter). Proces czasu rzeczywistego może być uruchamiany jako odpowiedź na przerwanie sprzętowe w określonym wcześniej momencie lub też cyklicznie, w stałych odstępach czasu (co niejako odpowiada reakcji na przerwania sprzętowego zegara czasu rzeczywistego).

Wszelkiego rodzaju pamięć pomocnicza jest na ogół bardzo mała albo nie występuje wcale. Wszystkie dane są przechowywane w pamięci o krótkim czasie dostępu lub w pamięci ROM. Systemy czasu rzeczywistego nie mają również większości cech nowoczesnych systemów operacyjnych, które oddalają użytkownika od sprzętu, zwiększając niepewność odnośnie ilości czasu zużywanego przez operacje. Na przykład prawie nie spotyka się w systemach czasu rzeczywistego pamięci wirtualnej.

Rosnący krąg zastosowań systemów czasu rzeczywistego o łagodnych wymaganiach na przetwarzanie w czasie rzeczywistym powoduje, że większość współczesnych systemów operacyjnych, łącznie z przeważającą częścią wersji systemu UNIX, czyni zadość wymaganiom miękkich systemów czasu rzeczywistego.

LITERATURA

[1] Źródła FSM Labs. Inc., http://fsmlabs.com

[2] M. Holler: *Przykłady programów testowych systemu czasu rzeczywistego RTLinux*, http://midas.psi.ch/rtlinux

[3] K.Lal, T.Rak, K.Orkisz: *RTLinux - system czasu rzeczywistego*, Wyd. Helion 2003

[4] Preprints of 27th IFAC/IFIP/IEEE Workshop on Real-time Programming WRTP'03, University of Zielona Góra

[5] V. Yodaiken, C. Dougan, M. Barabanov: *RTLinux/RTCore dual kernel real-time operating system*, FSMLabs Inc.

[6] K. Sacha: *Systemy czasu rzeczywistego*, Oficyna wydawnicza Politechniki Warszawskiej, Warszawa, 1999