ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: INFORMATYKA z. 38 Nr kol. 1451

Janusz KARMAŃSKI
Politechnika Śląska, Instytut Informatyki

ANALIZA PRACY APLIKACJI MULTIMEDIALNYCH CZASU RZECZYWISTEGO W SYSTEMIE LINUX

Streszczenie. Artykuł prezentuje analizę pracy aplikacji multimedialnych czasu rzeczywistego w systemie Linux. Przedstawione zostały wykresy opóźnień uzyskiwanych przy przetwarzaniu i transmisji informacji multimedialnej, w zależności od obciążenia systemu. Następnie wskazano, o jakie cechy powinno zostać wzbogacone jądro systemu, aby poprawić wydajność pracy aplikacji multimedialnych.

ANALYSIS OF REAL-TIME MULTIMEDIA APPLICATIONS WORKING IN LINUX

Summary. This paper presents analysis of distributed multimedia application working in Linux. Delays observed in computing and transmission of multimedia information in function of system load are presented. Finally methods of improving performance of multimedia applications are proposed.

1. Wstęp

Aplikacje multimedialne czasu rzeczywistego stawiają nowe wymagania dla systemów operacyjnych, pod kontrolą których pracują. Przykładem mogą być wideokonferencje pozwalające na komunikację w czasie rzeczywistym pomiędzy ludźmi znajdującymi się nieraz w znacznym oddaleniu od siebie, np. w domu i w biurze. Ten typ aplikacji jest bardzo wrażliwy na opóźnienia powstające w rozproszonym systemie multimedialnym, ponieważ informacja multimedialna jest generowana w czasie rzeczywistym (np. z kamery) i musi być na bieżąco konsumowana (np. wyświetlana na monitorze). Konieczne jest zapewnienie stałego czasu transmisji, czyli stałego opóźnienia każdej porcji informacji multimedialnej transmitowanej w sieci. Inaczej mówiąc, dane multimedialne są ważne tylko przez pewien

krótki okres czasu, po upływie którego stają się bezużyteczne, i jeżeli nie zostały odtworzone na komputerze użytkownika, muszą być porzucone.

W ostatnich latach prowadzonych jest wiele prac badawczych na polu zapewnienia gwarancji QoS dla aplikacji multimedialnych. Większość z tych prac koncentruje się na infrastrukturze sieciowej, czyli na zapewnieniu jakości połączeń w sieci. Najlepszym przykładem jest tu sieć ATM, od początku zawierająca wbudowane odpowiednie mechanizmy zarządzania jakością nawiązywanych połączeń.

Podczas gdy systemy transportowe zajmują się przesyłaniem danych, systemy końcowe odpowiedzialne są za ich przetwarzanie, jak np. kompresja i dekompresja, oraz prezentowanie użytkownikowi – między innymi odtwarzanie obrazu i dźwięku. Systemy te powinny więc przetwarzać strumienie multimedialne w czasie rzeczywistym, tak aby zapewnić wymaganą jakość usług aplikacjom multimedialnym. Ważną kwestią jest wyposażenie samego systemu operacyjnego w zmodyfikowane mechanizmy zarządzania zasobami, pozwalające aplikacjom negocjować ilość przydzielonych im zasobów, takich jak czas procesora, pamięć fizyczna czy interfejs sieci, oraz dającym pewne gwarancje, że wynegocjowane dla aplikacji parametry QoS będą realizowane.

W niniejszej pracy podjęta została próba znalezienia odpowiedzi na pytanie, jak zminimalizować czas przetwarzania danych multimedialnych w systemie, tak aby np. efektywnie wykorzystywać połączenie w sieci o zagwarantowanej przepustowości i parametrach QoS.

Przy ocenie jakości pracy aplikacji multimedialnych wykorzystane zostaną w dalszej części pracy między innymi terminy: opóźnienie oraz wahania opóźnienia [1]. Całkowite opóźnienie jest to odstęp czasu między rejestracją pewnej porcji informacji multimedialnej w punkcie wejściowym systemu a odtworzeniem tej informacji w punkcie wyjściowym. Przykładowo, dla wideokonferencji jest to czas między zeskanowaniem jednej klatki obrazu przez kamerę a wyświetleniem tej klatki na monitorze innego użytkownika. Można tu mówić zarówno o średnim opóźnieniu w pewnym przedziale czasu, jak i o opóźnieniu minimalnym i maksymalnym.

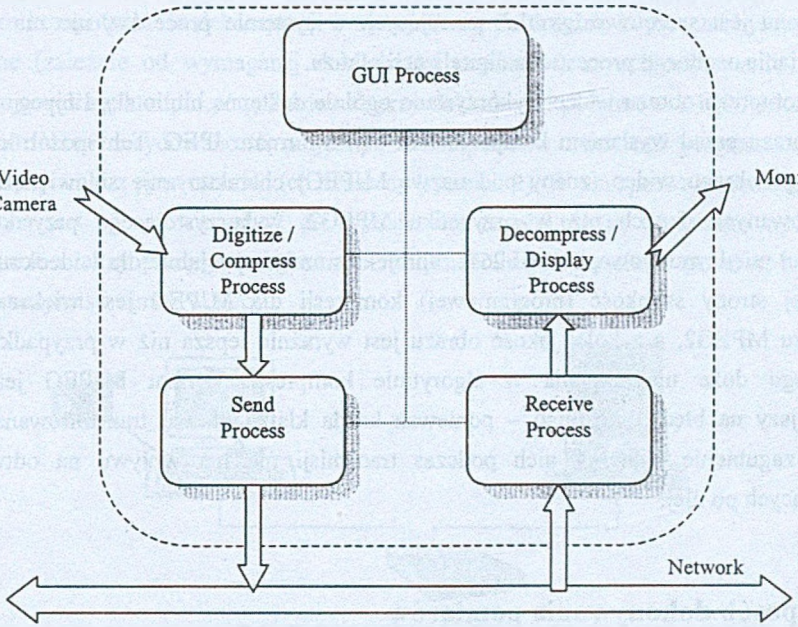
Wahania opóźnienia można zdefiniować jako różnicę między maksymalnym i minimalnym opóźnieniem. Występowanie znacznych odchyleń od uzgodnionego średniego opóźnienia powoduje przerwy w ciągłym odtwarzaniu informacji multimedialnej.

2. Budowa testowej aplikacji multimedialnej dla systemu Linux

Jako platformę badań zdecydowano się wykorzystać system operacyjny Linux pracujący na komputerach PC. Wybór systemu Linux podyktowany został dużą jego popularnością,

dostępnością wielu aplikacji oraz kompletnego kodu źródłowego systemu niezbędnego przy wprowadzaniu planowanych zmian do jądra, mających na celu wzbogacenie mechanizmów zarządzania zasobami o elementy gwarancji jakości usług.

Testowana aplikacja składa się z kilku niezależnych procesów realizujących różne zadania związane z przetwarzaniem i transmisją danych multimedialnych. Schemat aplikacji przedstawiony jest na rysunku 1.



Rys. 1. Schemat aplikacji testowej
Fig. 1. Test application schema

Interfejs użytkownika stworzony został przy użyciu funkcjonalnej biblioteki QT dostępnej bezpłatnie dla systemu Linux. Centralny proces wyświetla okienko główne aplikacji wyposażone w menu, z którego użytkownik wybiera polecenia sterujące pracą aplikacji. Polecenia te są następnie przekazywane do właściwych procesów odpowiedzialnych za ich realizację za pomocą mechanizmu przesyłania komunikatów. Przez większość czasu proces ten jest zawieszony w tzw. kolejce komunikatów w oczekiwaniu na akcję użytkownika, nie konsumuje on więc czasu procesora. Dopiero po wybraniu przez użytkownika np. polecenia z menu proces otrzymuje odpowiedni komunikat od systemu i po przetworzeniu polecenia użytkownika przesyła odpowiednie komunikaty do właściwych procesów, aby następnie powrócić do stanu oczekiwania.

Operacje kompresji/dekompresji oraz przesyłania i odbierania danych z sieci zostały rozbite na odrębne procesy. Dzięki temu wstrzymanie pracy jednego procesu na blokującej

operacji wejścia-wyjścia nie powoduje zatrzymania pracy pozostałych – np. dekompresja odebranej z sieci klatki obrazu wideo może odbywać się jednocześnie z odbieraniem następnej. Porcje przetworzonej informacji multimedialnej przekazywane są pomiędzy procesami odpowiedzialnymi za kompresję i wysyłanie oraz odbieranie i dekompresję za pomocą standardowego mechanizmu strumieni dostępnego w systemach Unix'owych. Ponieważ operacja digitalizacji każdej kolejnej klatki obrazu wideo przez kartę wejściową realizowana jest sprzętowo „w tle" pracujących w systemie procesów, nie ma potrzeby uruchamiania osobnego procesu dla digitalizacji obrazu.

Do kompresji obrazu wideo wykorzystano ogólnie dostępną bibliotekę libjpeg – kolejne klatki obrazu przed wysłaniem kompresowane są do formatu JPEG. Ten sposób kompresji ruchomego obrazu wideo (znany pod nazwą MJPEG) charakteryzuje się większą ilością transmitowanych danych niż w przypadku MPEG2, wykorzystującego przy kompresji zależności międzyramkowe, oraz H.261, zaprojektowanego specjalnie dla wideokonferencji. Z drugiej strony szybkość (programowej) kompresji dla MJPEG jest większa niż w przypadku MPEG2, a z kolei jakość obrazu jest wyraźnie lepsza niż w przypadku H.261 stosującego duże uproszczenia w algorytmie kompresji. Format MJPEG jest także odporniejszy na błędy transmisji – ponieważ każda klatka obrazu transmitowana jest w całości, zagubienie jednej z nich podczas transmisji nie ma wpływu na odtwarzanie następujących po niej.

3. Sposób dokonywania pomiarów

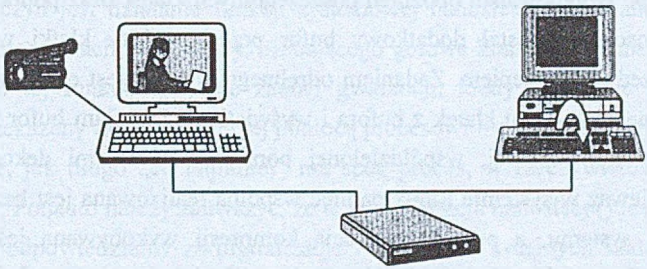
Tak stworzoną aplikację należało następnie wyposażyć w możliwość rejestracji czasów realizacji poszczególnych zadań, czasów przydziału procesora dla poszczególnych procesów, oraz poszerzyć o funkcje generacji odpowiednich wykresów i statystyk.

Pierwszy napotkany problem dotyczył możliwości (a raczej jej braku) precyzyjnego pomiaru czasu w systemie Linux. Maksymalną możliwą standardowo rozdzielczość oferuje funkcja systemowa times() zwracająca ilość „tyknięć" zegara systemowego od momentu uruchomienia systemu. Ponieważ zegar systemowy w Linux'ie pracującym na platformie PC zlicza impulsy dokładnie co 1/100 sekundy (z taką częstotliwością generowane jest przerwanie zegarowe), można wykonywać w ten sposób pomiary z dokładnością nie większą niż wspomniana 1/100 sekundy. Aby uzyskać większą dokładność pomiaru można między innymi odczytywać bezpośrednio wartość odpowiedniego rejestru układu czasowego generującego przerwania zegarowe.

Ponieważ ze względów bezpieczeństwa w systemie Linux dostęp do urządzeń (portów we-wy) możliwy jest jedynie przez jądro systemu oraz moduły ładowane dynamicznie

pracujące w jego przestrzeni, stworzony został odpowiedni moduł zwracający bieżącą wartość wspomnianego rejestru za pośrednictwem pliku specjalnego /dev/timer. Docelowe rozwiązanie powinno polegać na wyposażeniu samego jądra systemu w łatwiejszą w użyciu funkcję systemową.

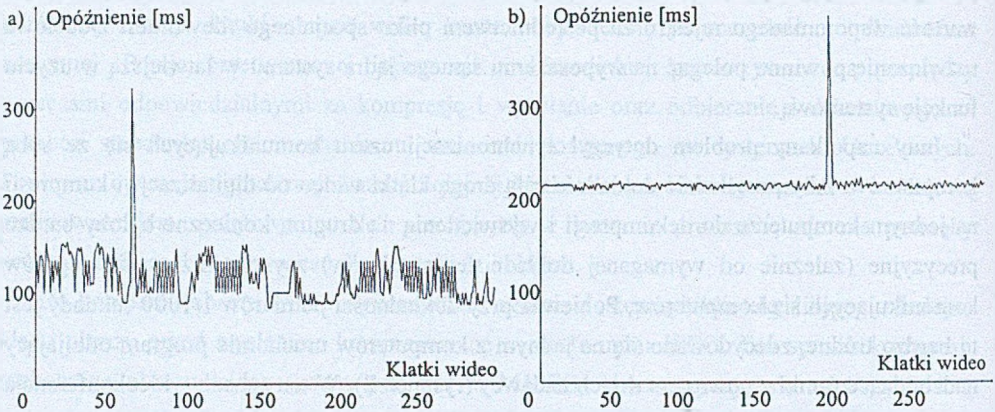
Inny napotkany problem dotyczył synchronizacji czasu komunikujących się ze sobą komputerów. Aby prześledzić dokładnie całą drogę klatki wideo od digitalizacji i kompresji na jednym komputerze do dekompresji i wyświetlenia na drugim, konieczne byłoby bardzo precyzyjne (zależnie od wymaganej dokładności pomiarów) zsynchronizowanie zegarów komunikujących się komputerów. Ponieważ przy dokładności pomiarów 1/1000 sekundy jest to bardzo trudne, zdecydowano się na jednym z komputerów uruchomić program odbijający nadchodzące ramki z powrotem do ich nadawcy (rysunek 2). W ten sposób wideokonferencja prowadzona jest „z samym sobą", ale wydaje się, że nie ma to większego wpływu na wykonywane analizy przepływu informacji multimedialnej, oczywiście poza wpływem na sam czas transmisji ramek w sieci.



Rys. 2. Stanowisko badawcze
Fig. 2. Research configuration

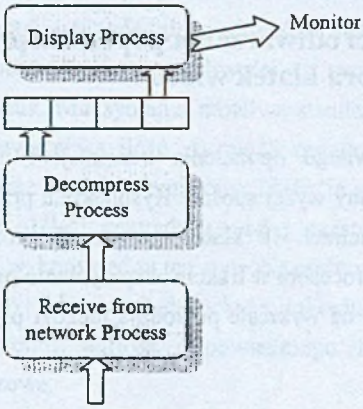
4. Poprawa płynności odtwarzania poprzez wprowadzenie dodatkowego bufora klatek wideo

W celu zbadania całkowitego opóźnienia transmisji i przetwarzania klatek wideo nawiązano połączenie w opisany wyżej sposób. Rysunek 3.a przedstawia wykres opóźnienia podczas konferencji o parametrach 10 klatek na sekundę, rozmiar klatki 300x200, 256 odcieni szarości. Obciążenie procesora w trakcie eksperymentu oscylowało wokół 70%. Duże wahania opóźnienia widoczne na wykresie powodują niezbyt płynne odtwarzanie obrazu na ekranie monitora.



Rys. 3. Opóźnienie a) bez bufora, b) po wprowadzeniu bufora klatek wideo
Fig. 3. Latency a) without buffer, b) with video frames buffer

Aby zniwelować wpływ wahań opóźnień poszczególnych klatek oraz poprawić płynność odtwarzania, zastosowany został dodatkowy bufor przechowujący klatki wideo po ich dekompresji, a przed odtworzeniem. Zadaniem odrębnego procesu jest cykliczne pobieranie w ustalonych momentach czasu klatek z bufora i wyświetlanie ich. Sam bufor zrealizowany jest w bloku pamięci wspólnej, współdzielonej pomiędzy procesami dekompresji oraz wyświetlania. Ponieważ w systemie Linux pamięć wspólna realizowana jest bezpośrednio w pamięci fizycznej systemu, a ponadto operacja kompresji wykonywana jest wprost do wspomnianego bufora, zaś wyświetlanie obrazu bezpośrednio z niego, nie ma żadnego dodatkowego narzutu czasowego związanego z kopiowaniem danych do i z bufora. Do synchronizacji dostępu procesów do bufora wykorzystywany jest mechanizm semaforów.



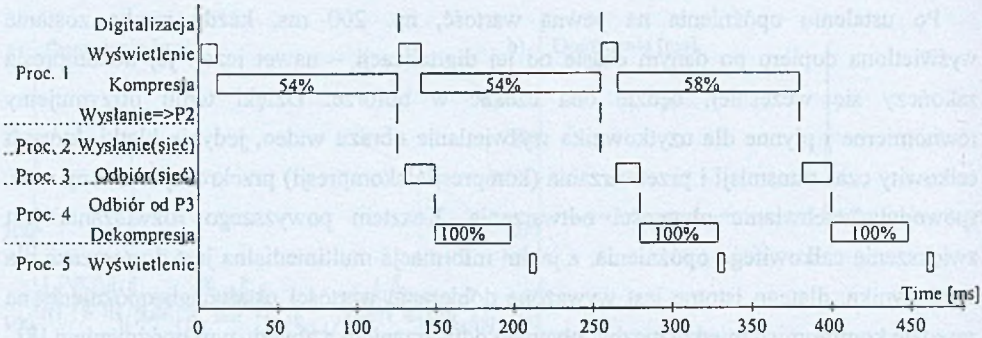
Rys. 4. Bufor wideo
Fig. 4. Video buffer

Po ustaleniu opóźnienia na pewną wartość, np. 200 ms, każda ramka zostanie wyświetlona dopiero po danym czasie od jej digitalizacji – nawet jeżeli jej dekompresja zakończy się wcześniej, będzie ona czekać w buforze. Dzięki temu otrzymujemy równomierne i płynne dla użytkownika wyświetlanie obrazu wideo, jedynie klatki, których całkowity czas transmisji i przetwarzania (kompresji/dekompresji) przekroczy ustalony czas, spowodują zachwianie płynności odtwarzania. Kosztem powyższego rozwiązania jest zwiększenie całkowitego opóźnienia, z jakim informacja multimedialna jest dostarczana dla użytkownika, dlatego istotne jest wyważone dobieranie wartości ustalonego opóźnienia na zasadzie kompromisu między niezbyt płynnym odtwarzaniem a zbyt dużym opóźnieniem [8].

Na rysunku 3.b przedstawiono opóźnienia uzyskane po wprowadzeniu do aplikacji bufora klatek wideo. Można zauważyć, że po użyciu bufora wykres poprawił się, choć nadal daleko mu do ideału. Widoczne na wykresie chwilowe duże skoki opóźnienia spowodowane są dużym czasem transmisji niektórych klatek wideo w sieci. Natomiast zamiast oczekiwanych 200 ms opóźnienie wynosi średnio prawie 230 ms, wykres nie jest również całkiem gładki. Jedną z przyczyn jest działanie funkcji systemowej nanosleep() używanej do budzenia w określonych momentach procesu wyświetlającego gotowe klatki pobierane z bufora. Sposób implementacji tej funkcji nie daje żadnej gwarancji, kiedy dokładnie procesor zostanie ponownie przekazany uśpionemu za jej pomocą procesowi – czas podawany jako jej parametr mówi jedynie, jak długo „co najmniej" ma spać proces, w rzeczywistości może to trwać jednak dłużej. Ponadto należy zauważyć, że ta sama funkcja nanosleep() jest wykorzystywana przez proces odpowiedzialny za digitalizację i kompresję kolejnych klatek obrazu z karty wejściowej wideo do usypiania w oczekiwaniu na moment digitalizacji kolejnej klatki. W efekcie już na samym początku momenty digitalizacji kolejnych klatek są obarczone pewnym błędem. Rozwiązaniem powyższych niedokładności wydaje się być wyposażenie systemu w możliwość automatycznego cyklicznego aktywowania niektórych procesów, w zależności od potrzeb aplikacji multimedialnych, np. dokładnie x razy na sekundę.

5. Analiza czasów przydziału procesora dla poszczególnych zadań

Aby określić faktyczny czas realizacji poszczególnych etapów przetwarzania i transmisji informacji multimedialnej, zmierzono czasy ich rozpoczęcia i zakończenia. Otrzymane wyniki różnią się znacznie w zależności od obciążenia procesora oraz parametrów pracy programu. Wynik dla trzech kolejnych klatek wideo przy rozmiarze klatki 300x200, 24bit owym kolorze, i 8 ramkach na sekundę (pełne obciążenie procesora) przedstawiony jest na rysunku 5.



Rys. 5. Czasy transmisji i przetwarzania informacji multimedialnej
Fig. 5. Multimedia information transmission and processing times

Ponieważ procesy są podczas pracy wywłaszczane (co widać wyraźnie na rysunku 5), pojawia się pytanie, ile faktycznie czasu procesora wykorzystują. Pomocne będą tu statystyki gromadzone przez jądro systemu przy każdym przerwaniu zegarowym, mówiące, ile faktycznie czasu proces użytkował procesor oraz ile trwało wykonywanie funkcji systemowych wywoływanych przez ten proces. Procentowe wykorzystanie czasu wyliczone przy użyciu tych statystyk dla operacji kompresji i dekompresji podano na rysunku.

Należy zwrócić uwagę, że przedstawione wartości są jedynie wartościami przybliżonymi – statystyki systemowe uaktualniane są tylko przy każdym przerwaniu zegarowym, a więc uzyskana dokładność to co najwyżej 1/100 sekundy. Proces w systemie Linux może otrzymać lub utracić procesor nie tylko w wyniku działania standardowej procedury szeregowania wywoływanej przy obsłudze przerwania zegarowego, ale także w wyniku wykonywania blokujących operacji we-wy, lub podczas wykonywania niektórych funkcji systemowych. Dlatego też nie ma większego sensu obliczanie procentowego czasu wykorzystania procesora przy zbyt krótkich operacjach, jak wysyłanie danych przez sieć czy wyświetlanie obrazu, dla których wynik obarczony jest zbyt dużym błędem – przykładowo dla procesu 3 (odbiór danych z sieci) wykorzystanie procesora wyliczone z wykorzystaniem odpowiednich statystyk wynosi 0%.

Należy zadać tu sobie pytanie, czy poprzez lepsze zarządzanie czasem procesora możliwe jest zmniejszenie opóźnienia spowodowanego przetwarzaniem informacji multimedialnej w systemie. Zwróćmy uwagę, że sam czas przetwarzania danych w systemie przed ich wysłaniem przez sieć nie jest optymalny – przy innym dzieleniu procesora między zadania czas kompresji można byłoby skrócić prawie o połowę, jeżeli procesor odbierany byłby procesowi pierwszemu dopiero po zakończeniu kompresji. W ten sposób można byłoby uzyskać mniejsze całkowite opóźnienie transmisji informacji multimedialnej. Wydaje się więc, że aplikacja współdziałając z systemem operacyjnym przy wykorzystaniu zasobów (w

tym przypadku CPU) mogłaby osiągnąć lepsze wyniki, a więc zapewnić użytkownikowi usługi o lepszej jakości.

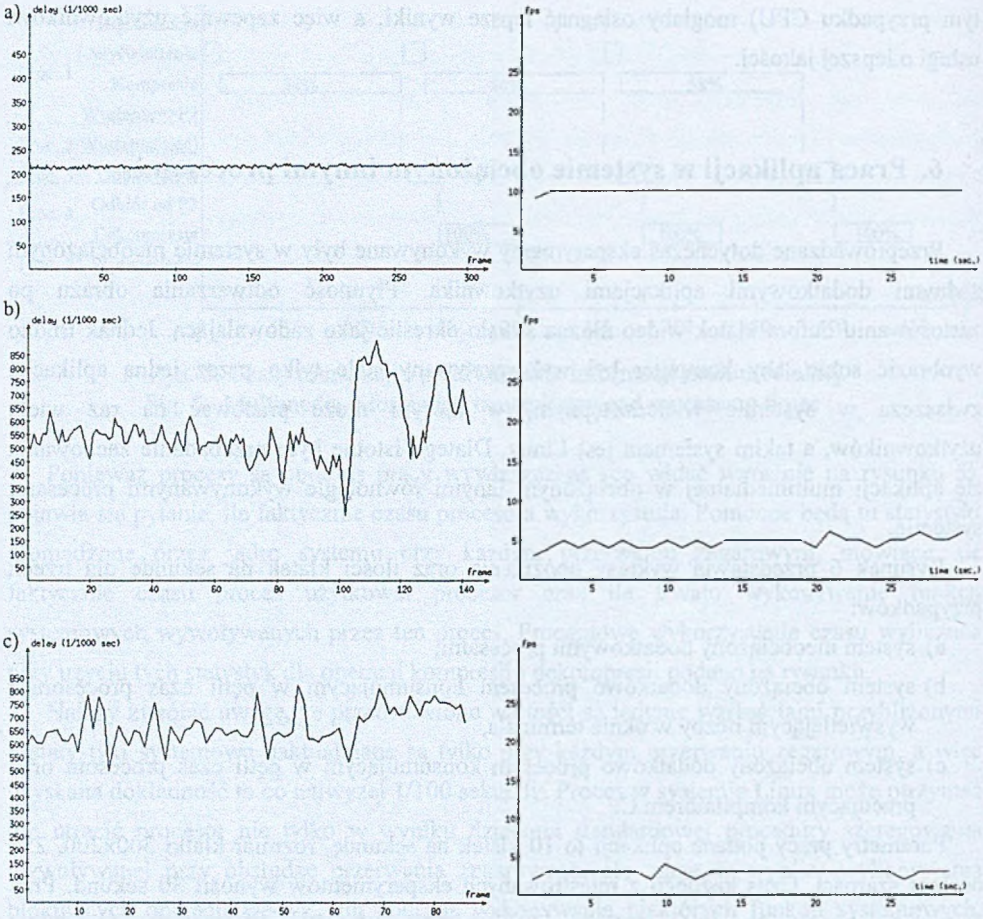
6. Praca aplikacji w systemie obciążonym innymi procesami

Przeprowadzane dotychczas eksperymenty wykonywane były w systemie nieobciążonym żadnymi dodatkowymi aplikacjami użytkownika. Płynność odtwarzania obrazu po zastosowaniu bufora klatek wideo można śmiało określić jako zadowalającą. Jednak trudno wyobrazić sobie, aby komputer był wykorzystywany stale tylko przez jedną aplikację, zwłaszcza w systemie wielodostępnym, w którym może pracować na raz wielu użytkowników, a takim systemem jest Linux. Dlatego istotne było przebadanie zachowania się aplikacji multimedialnej w obciążonym innymi równolegle wykonywanymi procesami systemie.

Rysunek 6 przedstawia wykresy opóźnienia oraz ilości klatek na sekundę dla trzech przypadków:

a) system nieobciążony dodatkowymi procesami,
b) system obciążony dodatkowo procesem konsumującym w pętli czas procesora i wyświetlającym liczby w oknie terminala,
c) system obciążony dodatkowo procesem konsumującym w pętli czas procesora oraz pracującym kompilatorem C.

Parametry pracy podane aplikacji to 10 klatek na sekundę, rozmiar klatki 300x200, 256 odcieni szarości. Czas każdego z rejestrowanych eksperymentów wynosił 30 sekund. Przy systemie nieobciążonym (rysunek 6.a) aplikacja nie ma żadnych problemów z zachowaniem płynności wyświetlania oraz utrzymaniem żądanych 10 klatek na sekundę – obciążenie procesora oscyluje wokół 70 %. Jednak w przypadku b) uzyskiwana ilość klatek na sekundę spada do 4-5, zaś opóźnienie transmisji poszczególnych klatek zwiększa się ponad dwukrotnie, nie ma także mowy o jakiejkolwiek płynności odtwarzania. Jeszcze gorzej wygląda sytuacja w punkcie c) – ilość klatek na sekundę spada do średnio trzech, co oczywiście nie może być akceptowalne przez użytkownika aplikacji.



Rys. 6. Opóźnienie oraz ilość klatek na sekundę w zależności od obciążenia systemu
Fig. 6. Delay and frames per second depending from system load

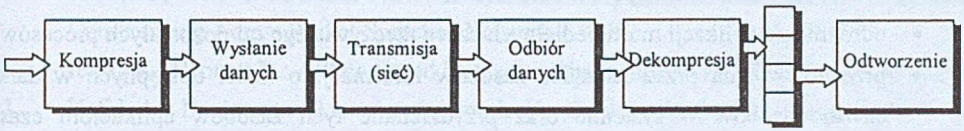
Przy większej ilości procesów pracujących w systemie procesor przydzielany jest wg standardowego „sprawiedliwego" unix'owego algorytmu szeregowania. Algorytm ten nie jest jednak zadowalający dla aplikacji multimedialnych. Użytkownik korzystający z usługi multimedialnej czasu rzeczywistego, np. wideokonferencji oraz z innych aplikacji jednocześnie, chciałby przypuszczalnie mieć zapewnioną równomierność przesyłania obrazu i dźwięku, w razie konieczności kosztem mniejszej ilości klatek na sekundę czy słabszej jakości obrazu, natomiast inne procesy – nie będące procesami czasu rzeczywistego – nie powinny zakłócać pracy wideokonferencji, nawet kosztem mniejszej własnej wydajności.

Okazuje się także, że zwiększanie priorytetu aplikacji multimedialnej nie przynosi tu oczekiwanych efektów [6]. Przy zbyt dużym priorytecie następuje „zagłodzenie" innych

procesów, interfejs użytkownika (myszka, klawiatura) przestaje prawidłowo reagować. Przy mniejszym priorytecie wymagania aplikacji nie są spełnione.

7. Analiza strumienia przepływu informacji multimedialnej

Każdy etap przetwarzania oraz transmisji informacji można przestawić jako pewien moduł pobierający dane z wejścia, przetwarzający je lub tylko transmitujący, oraz przekazujący przetworzone dane na wyjście. Całą drogę, jaką przechodzą dane multimedialne od momentu digitalizacji na jednym komputerze do wyświetlenia na drugim, przedstawia rysunek 7.



Rys. 7. Strumień przepływu informacji multimedialnej
Fig. 7. Multimedia information flow stream

Każdy moduł można scharakteryzować za pomocą takich parametrów, jak rozmiar danych wejściowych oraz rozmiar danych wyjściowych (mogą one różnić się np. wskutek kompresji czy dekompresji), czas (średni, minimalny, maksymalny) przetwarzania określonej porcji danych wejściowych. Parametry te można łatwo przekształcić w przepustowość, średnie opóźnienie, wahania opóźnienia. Innym nie analizowanym tu parametrem jest współczynnik gubienia danych, które może występować nie tylko podczas transmisji danych w sieci, ale także w modułach kompresji i dekompresji, w przypadku niewystarczającej ich wydajności.

Każdy moduł potrzebuje do swojej pracy pewnych zasobów. Dla kompresji i dekompresji będzie to przede wszystkim czas procesora, dla wysyłania i odbierania danych interfejs we-wy sieciowego, dla transmisji w sieci np. nawiązane połączenie o określonych parametrach w przypadku ATM. Warunkiem przetworzenia danych przez moduł jest więc nie tylko pojawienie się odpowiednich danych na jego wejściu, ale również przydzielenie mu wszystkich wymaganych zasobów. Tak więc całkowite opóźnienie powstające podczas przetwarzania i transmisji danych równe jest sumie opóźnień powstających w poszczególnych modułach oraz opóźnień w aktywowaniu odpowiednich modułów, kiedy na ich wejściu czekają dane, ale nie zostały im jeszcze przydzielone przez system wymagane zasoby. Wymusza to na systemie zintegrowane zarządzanie zasobami – np. przydział pasma w sieci wymaga również przydzielenia aplikacji interfejsu we-wy sieciowego, buforów w pamięci,

kanału DMA itp., oraz – aby zminimalizować czas przepływu danych – odpowiednie zasoby powinny być przydzielane modułom najpóźniej w momencie, kiedy na ich wejściu pojawiają się dane.

8. Proponowane rozszerzenia mechanizmu szeregowania procesów

System operacyjny powinien oddzielać procesy czasu rzeczywistego (w przypadku aplikacji multimedialnych mówimy o tzw. „soft realtime") od pozostałych oraz zawierać mechanizmy pozwalające im współdziałać przy zarządzaniu zasobami. W wyniku przeprowadzonych badań dla CPU można zaproponować wzbogacenie mechanizmu szeregowania procesów o następujące cechy:

- odróżnienie aplikacji multimedialnych czasu rzeczywistego od pozostałych procesów,
- przechowywania przez zarządcę zasobów informacji o ilości dostępnych w danej chwili zasobów w systemie oraz przydzielanie tych zasobów aplikacjom czasu rzeczywistego w wyniku przeprowadzonej z nimi negocjacji, co pociąga również za sobą konieczność opracowania interfejsu negocjacji QoS,
- zapewnienie aplikacjom czasu rzeczywistego pewnego uzgodnionego (wynegocjowanego przy starcie aplikacji) czasu przydziału procesora (w ogólności również innych zasobów) niezależnie od obciążenia systemu innymi procesami. Jeżeli nie jest to możliwe – kiedy dostępność zasobów w systemie zmienia się – przeprowadzenie adaptacji aplikacji multimedialnych do nowej dostępności zasobów,
- pozostawienie aplikacji decyzji o momencie zwrotu procesora (lub innego zasobu), w ramach uzgodnionego wcześniej przydziału czasu dla aplikacji,
- udostępnienie mechanizmu okresowego cyklicznego uruchamiania procesu,
- aktywowanie kolejnych procesów na ścieżce przepływu informacji multimedialnej w momencie, kiedy na ich wejściu pojawiają się dane, co wymaga powiązania mechanizmów komunikacji z szeregowaniem procesów.

LITERATURA

1. Karmański J.: Jakość usług w rozproszonych systemach multimedialnych. VI Konferencja Sieci Komputerowych, ZNPS seria Informatyka z. 36.
2. Hafid A., Bochmann G.v., Dssouli R.: Distributed multimedia Applications and Quality of Service.

3. Hafid A., Bochmann G.v.: Some Principles for Quality of Service Management.
4. Wajda K., Błasiak J., Grzybacz M., Marek C.: Budowa sieci komputerowych w technologii ATM. Wydawnictwo Fundacji Postępu Telekomunikacji, Kraków 1997.
5. Mallikarjun Tatipamula, Bhumip Khasnabish: Multimedia Communication Networks: Technologies and Services. Artech House. Boston, London 1998.
6. Krishnamurthy L.: AQUA: An Adaptive Quality of Service Architecture for Distributed Multimedia Applications.
7. Yau D., Lam S.: An Architecture Towards Efficient OS Support for Distributed Multimedia.
8. Stone D.L., Jeffay K.: An Empirical Study of Delay Jitter Management Polices. Multimedia Systems, Volume 2, Number 6 (January 1995) pages 267-279.
9. Jeffay K., Stone D.L, Smith F.D.: Transport and Display Mechanisms For Multimedia Conferencing Across Packet-Switched Networks.
10. Jeffay K., Bennett D.: A Rate-Based Execution Abstraction For Multimedia Computing.
11. Hafid A., Bochmann G.v.: An Approach To Quality Of Service Management For Distributed Multimedia Application.

Recenzent: Dr inż. Maciej Bargielski

Wpłynęło do redakcji 18 listopada 1999 r.

Abstract

Important area of research on Quality of Service relates to resource allocation within real-time operating systems in order to be able to guarantee specific performance guarantees for real-time processing of multimedia data. In order to meet the user and application requirements operating system have to provide some mechanisms quality of service. To find requirements of distributed real-time multimedia applications program for Linux was created. Schema of the application is presented on figure 1 and 4, research configuration is presented on figure 2. Chart on figure 3 shows delays before and after adding video frames buffer to tested application.

Multimedia systems contain a number of stream processing components (figure 7), responsible for video compression and decompression, data transmission etc. The resulting end-to-end quality of multimedia service must be considered to be the concatenation of the QoS characteristics of the individual transmission and computing components through which

the information flows. Figure 5 presents delay generated for tested application in every system component.

Other important problem is co-operating real-time multimedia application with others programs working in system. Figure 6 presents achieved delays and frames per second in function of system load.

Finally, some methods of improving performance of multimedia applications are proposed in point 8. Most important are: special support in operating system for multimedia real-time applications, QoS negotiation, enforce, adaptation.