

PREZENTACJA SYSTEMU OPERACYJNEGO SCOUT

Paweł Pisarczyk

Plan prezentacji:

- 1. Wstęp**
- 2. Scout – system operacyjny oparty o ścieżki.**
- 3. Moduły**
- 4. Ścieżki**
- 5. Stopnie ścieżek**
- 6. Interfejsy**
- 7. Wydajność**
- 8. Możliwości zastosowań**
- 9. Podsumowanie**

Warszawa, 1999

1. Wstęp.

Początek systemów operacyjnych datuje się na połowę lat 50, gdy pojawiły się komputery drugiej generacji. Rosnący poziom komplikacji ówczesnych komputerów i stawianych im zadań obliczeniowych powodował, że pisanie oddzielnych programów w kodzie maszynowym dla oddzielnych problemów obliczeniowych było bardzo pracochłonne i pozbawione większego sensu. Pojawiła się więc konieczność stworzenia oprogramowania rdzennego - zarządzającego wykonywaniem się programów obliczeniowych i wspomagającego ich tworzenie. Zauważono, że zadania obliczeniowe składają się ze zbliżonych do siebie partii kodu, odpowiadających za obliczanie wartości funkcji matematycznych, pobieranie i zapisywanie danych, obsługę pamięci masowej itp. i postanowiono by należały one do oprogramowania systemowego. Rozpoczęto także prace nad stworzeniem języków abstrakcyjnych, umożliwiających wygodny opis formalny obliczeń i zamianę tego opisu na kod maszynowy. Tak narodziły się języki takie jak np. FORTRAN i Algol i gałąź informatyki zwana teorią systemów operacyjnych.

Burzliwy rozwój elektroniki i pojawianie się coraz to szybszych komputerów powodował, że prowadzono dalsze badania nad systemami operacyjnymi, koncentrując się nie tylko nad tym jakie usługi powinien dostarczać system, ale również nad tym, jak powinien to robić. Dążono do maksymalnego wykorzystania czasu drogiej jednostki obliczeniowej, tak by mogło z niej skorzystać jak najwięcej użytkowników w tym samym czasie i by cały czas wykonywała programy. W dążeniach do maksymalnego wykorzystania jednostki obliczeniowej można wyróżnić kilka przełomowych etapów, które miały znaczący wpływ na architekturę współczesnych systemów operacyjnych:

- a) przetwarzanie wsadowe
- b) wieloprogramowość
- c) powstanie systemów z podziałem czasu (1962 – system CTSS – w systemie tym krótsze zadania były przetwarzane przed zadaniami dłuższymi poprzez nadanie im wyższych priorytetów niż zadaniami dłuższymi)
- d) umożliwienie jednoczesnej, interaktywnej pracy na komputerze wielu użytkowników – powstanie systemu MULTICS i systemu UNIX.

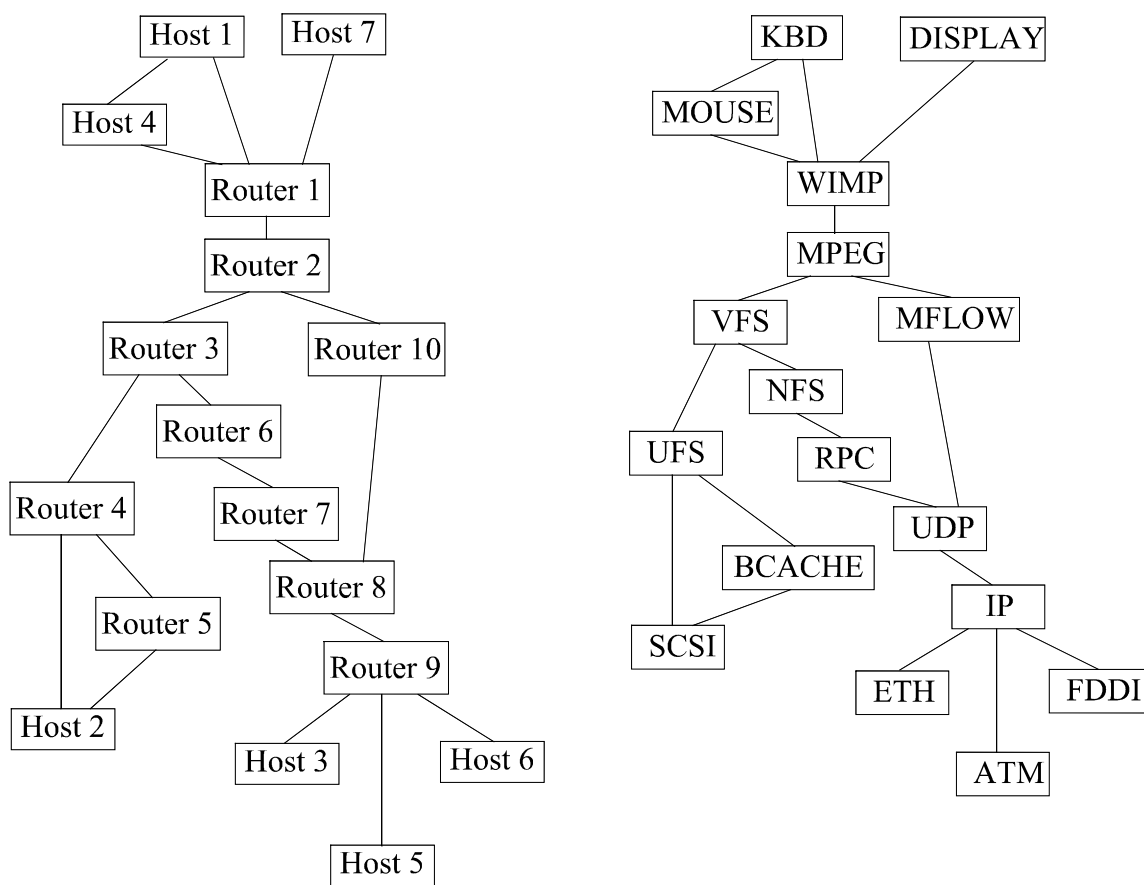
Wraz z upowszechnianiem się mini komputerów i rozwojem sieci komputerowych oprogramowanie systemowe stawało się coraz bardziej skomplikowane. Pojawił się kolejny problem. Jak sprawić, by na komputerach o różnej architekturze mógł działać ten sam system operacyjny, zapewniający ten sam interfejs programowy i to samo wygodne środowisko pracy? Odpowiedź na to pytanie wydaje się teraz prosta – należy napisać system w języku wysokiego poziomu. Jednak jak postąpić, gdy nie ma języka wysokiego poziomu, który umożliwiłby wygodne i wydajne zaprogramowanie w nim systemu? Przy głębszym zastanowieniu dochodzi się do wniosku, że bodaj największym osiągnięciem w całej dotychczasowej historii systemów operacyjnych było powstanie systemu UNIX(1973) autorstwa D. Ritchiego i K. Thomsona - pierwszego systemu napisanego w języku wysokiego poziomu C. Język C został opracowany także przez D. Ritchiego na podstawie języka B opracowanego przez K. Thomsona, który powstał z języka BCPL. Napisanie systemu w języku C było posunięciem bardzo jak na owe czasy nietypowym – jednak przyczyniło się do szerokiej akceptacji systemu przez użytkowników i jego popularności. System mógł być przenoszony, po dokonaniu niewielu zmian, na komputery o różnej architekturze. W roku 1984 na świecie istniało około 100000 instalacji Unixa na komputerach różnych producentów o bardzo różnych możliwościach - od mikroprocesorów do dużych maszyn.

Postępujący i burzliwy rozwój przemysłu komputerowego, nie spotykany w żadnej innej gałęzi przemysłu, sprawił, że istnieje wiele doskonałych systemów operacyjnych, które wiele „wyniosły” z paradygmatu uniksowego. Jednak ze względu na ich ogólność i „słabą” modularność nie jest możliwe ich efektywne wykorzystanie przy realizacji komunikujących się systemów specjalizowanych. Należy zauważyć, że rozwój Internetu i sieci lokalnych sprawił, że większość współczesnych komputerów osobistych wykorzystywana jest głównie do komunikacji i pozyskiwania informacji o różnym charakterze: pliki, obrazy, głos, dokumenty hipertekstowe. Rosnąca popularność komputerów, stawiająca je na równi z innymi urządzeniami powszechnego użytku, powoduje, że konieczne jest teraz tworzenie systemów operacyjnych, które mogą być łatwo dopasowywane do konkretnych zastosowań, pracują z szeroką gamą urządzeń, nie potrzebują serwisu, a ich głównym przeznaczeniem jest komunikacja. Dodatkowo system taki w konkretnym wcieleniu daje użytkownikowi tylko pewien wąski podzbiór swoich funkcji, dzięki czemu jest bardzo prosty w użytkowaniu (rozważmy komputer, który jako jedyny interfejs użytkownika dostarcza jedynie przeglądarkę, a dojdziemy do popularnej koncepcji komputera sieciowego lansowanej przez największe firmy informatyczne). Współczesne systemy ogólnego przeznaczenia nie mogą być z uwagi na swoją ogólność w pełni konfigurowane przez niewykwalifikowany personel, a próby idące w kierunku poprawienia wygody ich eksploatacji, jak wiadomo, nie są zbyt zadowalające. Wyraz ogólny można z powodzeniem zastąpić wyrażeniem: - wymagający profesjonalnego serwisu. Pewną nową próbą podejścia dotego problemu jest system operacyjny Scout stworzony przez Davida Mosbergera z Uniwersytetu w Arizonie.

2. Scout - system operacyjny oparty o ścieżki

System operacyjny Scout został napisany w języku C i zaimplementowany dla komputerów opartych o procesory rodziny Alpha AXP (DECchip 21064, DECchip 21164). Autorem tego systemu jest David Mosberger, który przez ostatnie lata zajmował się także rozwojem systemu Linux dla architektury Alpha AXP. System Scout został zaprojektowany w taki sposób, by wspierać konstruowanie modularnych systemów specjalizowanych. System Scout jest systemem z podziałem czasu z szeregowaniem pobłażającym tzn. zadanie w sposób jawny musi zrzec się procesora. Dopiero wtedy program szeregujący może wybrać kolejne zadanie do wykonania. Autor systemu argumentuje wybór pobłażającej strategii szeregowania, tym, że system jest jeszcze w fazie eksperymentów. Dodatkowo przy takiej strategii szeregowania nie jest konieczne zachowywanie pełnego kontekstu procesu. Wystarczy bowiem zapamiętywać tylko niektóre rejestry procesora używane przez zadanie, ponieważ znany jest moment kiedy sterowanie zostanie przekazane do programu szeregującego. Przy strategii wyłączeniowej - program szeregujący uruchamiany jest zazwyczaj wraz z każdym „tikiem” zegara systemowego, a więc moment wyłączenia nie jest znany przez aktualnie wykonywane zadanie i muszą być zachowane wszystkie rejestry procesora. System Scout nie wspiera wielu domen ochrony tzn. wszystkie zadania wykonują się w jednej wspólnej przestrzeni adresowej. Nie wspiera też symetrycznej wieloprocessorowości. Cechy te można uznać za niewątpliwe wady systemu, jednak głównym celem stworzenia systemu było praktyczna realizacja systemu wspierającego koncepcję ścieżek Autor zapewnia, że braki systemu zostaną w przyszłości usunięte.

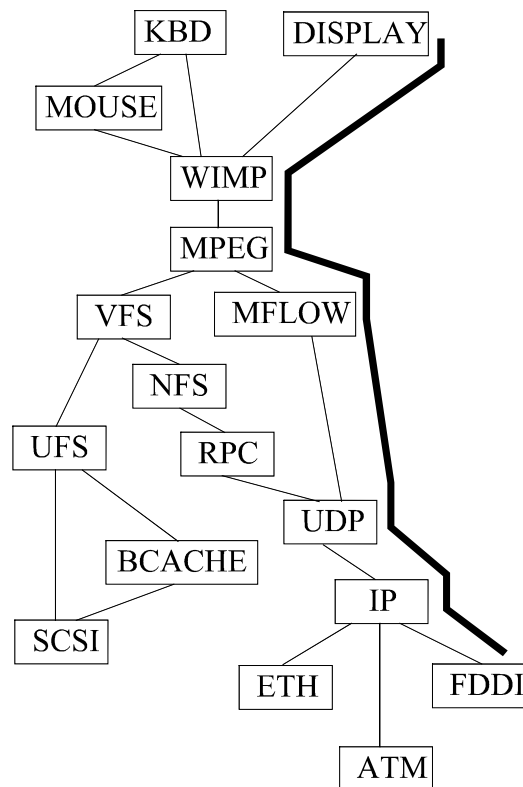
Scout jest systemem opartym o tzw. ścieżki. Koncepcja ścieżek jest rozszerzoną koncepcją strumieni znanych z systemu Unix. Opiera się ona głównie na analogi pomiędzy heterogenicznym systemem sieciowym, a systemem jednokomputerowym (rys. 1).



Rys. 1 Analogia pomiędzy systemem sieciowym, a systemem jednokomputerowym

Na rysunku z lewej przedstawiony jest heterogeniczny system sieciowy, który jest zbiorem komputerów połączonych siecią. Pakiety pomiędzy komputerami przekazywane są przy pomocy routerów. Router rozpoznaje adres odbiorcy i podejmuje decyzję, którym interfejsem sieciowym ma wysłać odebrany pakiet, by jak najszybciej dotarł on do odbiorcy. Komputer wysyłający nie wie nic o tym jaką drogą pakiet zostanie wysłany oraz przez ile routerów zostanie przekazany dalej. Z punktu widzenia komputera wysyłającego i odbierającego nie jest istotne jaką drogą pakiet zostanie dostarczony. Komputery wysyłają i odbierają pakiety tak, jakby znajdowały się w tej samej sieci. Na rysunku z prawej przedstawiony jest klasyczny system jednokomputerowy, składający się z urządzeń wejściowych i wyjściowych (np. klawiatura, monitor), interfejsów sieciowych i podsystemu dyskowego. Gdy na system jednokomputerowy spojrzysz tak, jak na heterogeniczny system sieciowy, a więc potraktuje się go jako zbiór niezależnych modułów, które tylko odbierają dane, przetwarzają je i przekazują dalej na podstawie zdefiniowanych wcześniej połączeń, dochodzi się do koncepcji ścieżek. Ścieżką jest właśnie owe zdefiniowane wcześniej powiązanie między modułami, wyznaczające drogę przekazywania danych (rys. 2). System oparty o ścieżki jest więc niezwykle modularny i cechują go bardzo dobrze zdefiniowane powiązania pomiędzy modułami, które tworzą kolejne warstwy systemu. Systemy modularne charakteryzują się zazwyczaj dużo gorszą wydajnością niż systemy monolityczne - jednak pionowa integracja warstw systemu i wynikająca z tego mnogość powiązań między nimi określana jako linia spaghetti powodują, że systemy monolityczne są trudniejsze w dopasowaniu ich do konkretnych zastosowań, pielęgnacji i rozbudowie. Koncepcja ścieżek wydaje się

rozwiązywać ten problem. Dodatkowo system oparty o ścieżki jest modułowy i tak samo, a może nawet bardziej wydajny niż system monolityczny. Jest on bowiem zbiorem modułów wiązanych przed startem. Z uwagi na dobrze zdefiniowane interfejsy pomiędzy modułami możliwa jest wyrafinowana optymalizacja, a nawet ręczne kodowanie modułów w assemblerze. Ścieżki w systemie Scout mogą być wydłużane w trakcie pracy systemu. Dodatkową zaletą wynikającą ze stosowania ścieżek jest spójny interfejs komunikacyjny dostarczany programom działającym w systemie. Wątek żąda od systemu otwarcia ścieżki poprzez wywołanie systemowe i w odpowiedzi dostaje jej uchwyt w systemie. Przy wykorzystaniu tego uchwytu np. tak jak przy wykorzystaniu deskryptora pliku lub strumienia w systemie Unix do zapisu odczytu danych do pliku, wątek wysyła lub odbiera dane. Przy otwarciu ścieżki można przekazać jej specjalne parametry, którymi może być np. adres komputera, z którym chcemy się połączyć. Jak widać proces może w ten sam sposób komunikować się z innym procesem poprzez sieć przy użyciu dowolnego protokołu, którego obsługa byłaby włączona do ścieżki. Interfejs pozostaje zawsze ten sam.

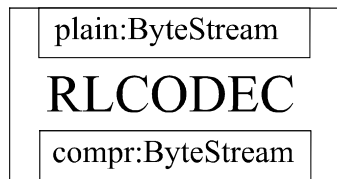


Rys. 2 Ścieżka

3. Moduły

Jak już wspomniano system składa się z modułów, które są łączone ze sobą przy starcie. Moduł realizuje określone funkcje, które są dostępne przy pomocy serwisów. Serwisy powinny być kompatybilne tzn. typ danych wyjściowych serwisu powinien być kompatybilny z typem danych wejściowych kolejnego serwisu w ścieżce. Serwisy są odrębnymi modułami napisanymi w języku C. Powiązania serwisów z określonymi modułami dokonuje się przy pomocy specjalnego języka opisu modułu. Moduł musi zawierać co najmniej dwa serwisy.

Rozważmy na przykład moduł dokonujący kompresji i dekompresji RLE przedstawiony na rysunku 3.



Rys. 3 Przykładowy moduł filtrujący dokonujący kompresji i dekompresji RLE

Jak widać składa się on z jednego serwisu dokonującego kompresji RLE - serwis `plain` - i drugiego serwisu, dokonującego dekompresji - serwis `compr`. Dane przesyłane do modułu na wejście `plain` ulegają kompresji, natomiast dane przesyłane w odwrotnym kierunku ulegają dekompresji. Moduł będący częścią systemu i działający w trybie jądra może odwoływać się do sprzętu w związku z czym może dokonywać kompresji lub dekompresji przy pomocy sprzętowego kodera / dekodera. W przypadku kompresji RLE oczywiście pozbawione jest to większego sensu, ale na przykład dla kodowania MPEG może to być bardzo dobry pomysł. Moduł posiadający dwa serwisy jest tzw. modułem filtrującym. Moduł ładowany jest do pamięci w trakcie startu systemu, a odwołania do niego odbywają się przy pomocy specjalnej struktury `Module` zdefiniowanej poniżej:

```

typedef struct Module {
    String          name;
    String          module_name;
    Long            (*init)(Module m);
    CreateStageFunc createStage;
    DemuxFunc       demux;
    Struct ModuleLink {
        Module module;
        Int service;
    } links[];
} * Module;
  
```

Struktura `Module` wykorzystywana jest przez stopnie ścieżki do odnalezienia opracowanych wcześniej serwisów i uruchomienia ich z konkretnymi danymi. Powiązania pomiędzy serwisami modułów zdefiniowane są poprzez tablicę `links[][]`. Serwisy dostępne są poprzez tablicę `links[][]`. Każdy moduł posiada swój unikatowy numer w systemie, a każdy serwis posiada unikatowy numer w obrębie modułu. System wykonując ścieżkę, przechodząc do jej kolejnych stopni uruchamia konkretny serwis skojarzonego z konkretnym stopniem. Modułu przy wykorzystaniu tablicy `links[][]`.

4. Ścieżki

Ścieżka reprezentowana jest w pamięci jako specjalna struktura danych. Definicja przedstawiona jest poniżej:

```

typedef struct Path {
    long          pid;
    Stage         end[2];
  
```

```

    PathQueue    q[4];
    struct Attrs  attrs;
    bool         realtime;
    u_long       prio;
} * Path;

```

Ścieżka jest obiektem składającym się ze stopni (path stages), które powiązane są z określonymi modułami. Z każdą ścieżką skojarzone są 4 kolejki komunikatów określone tablicą `q[4]`. Po dwie kolejki dla każdego kierunku przekazywania danych tzn. jedna kolejka wejściowa i jedna kolejka wyjściowa. Każda utworzona ścieżka w systemie posiada swój własny priorytet uwzględniany przy szeregowaniu ścieżek. Procesy w systemie nie posiadają odrębnych priorytetów. Priorytet procesu określony jest przez wykonywaną ścieżkę. Ścieżkę tworzy się przy pomocy wywołania systemowego `createPath()`, które z kolei wywołuje funkcję `createStage()` skojarzoną ze strukturą `Module` i odpowiedzialną za tworzenie kolejnych stopni ścieżki.

3. Stopnie ścieżki

Definicja stopnia przedstawiona jest poniżej:

```

struct Stage {
    Iface    iface[2];
    Path     path;
    Module   module;
    Long     (*establish)(Stage s, Attrs a);
    Void     (*destroy)(Stage s);
    Stage    nextStage;
} * Stage;

```

Każdy ze stopni zawiera wskaźniki do funkcji `establish()` i `destroy()`. Funkcje te używane są przy tworzeniu i usuwaniu ścieżki. Stopień kojarzony jest z modułem poprzez pole `module`.

4. Interfejsy

Poszczególne stopnie zawierają po dwa interfejsy odpowiadające za przekazywanie danych do kolejnych stopni. Każdy z interfejsów służy do przekazywania danych w określonym kierunku do kolejnych stopni ścieżki. Interfejs może zawierać wiele innych pomocniczych funkcji np. służących do ustawiania konkretnych parametrów ścieżki np. może być to adres IP docelowego hosta, lub maksymalny rozmiar okna TCP skojarzony z interfejsem stopnia skojarzonego z modułem TCP. Najczęściej używanym typem interfejsu jest w systemie Scout interfejs asynchroniczny. Definicje interfejsu podstawowego i prostego interfejsu asynchronicznego, przedstawione są poniżej:

```

Typedef struct Iface {
    Iface    next;
    Iface    back;
}

```

```

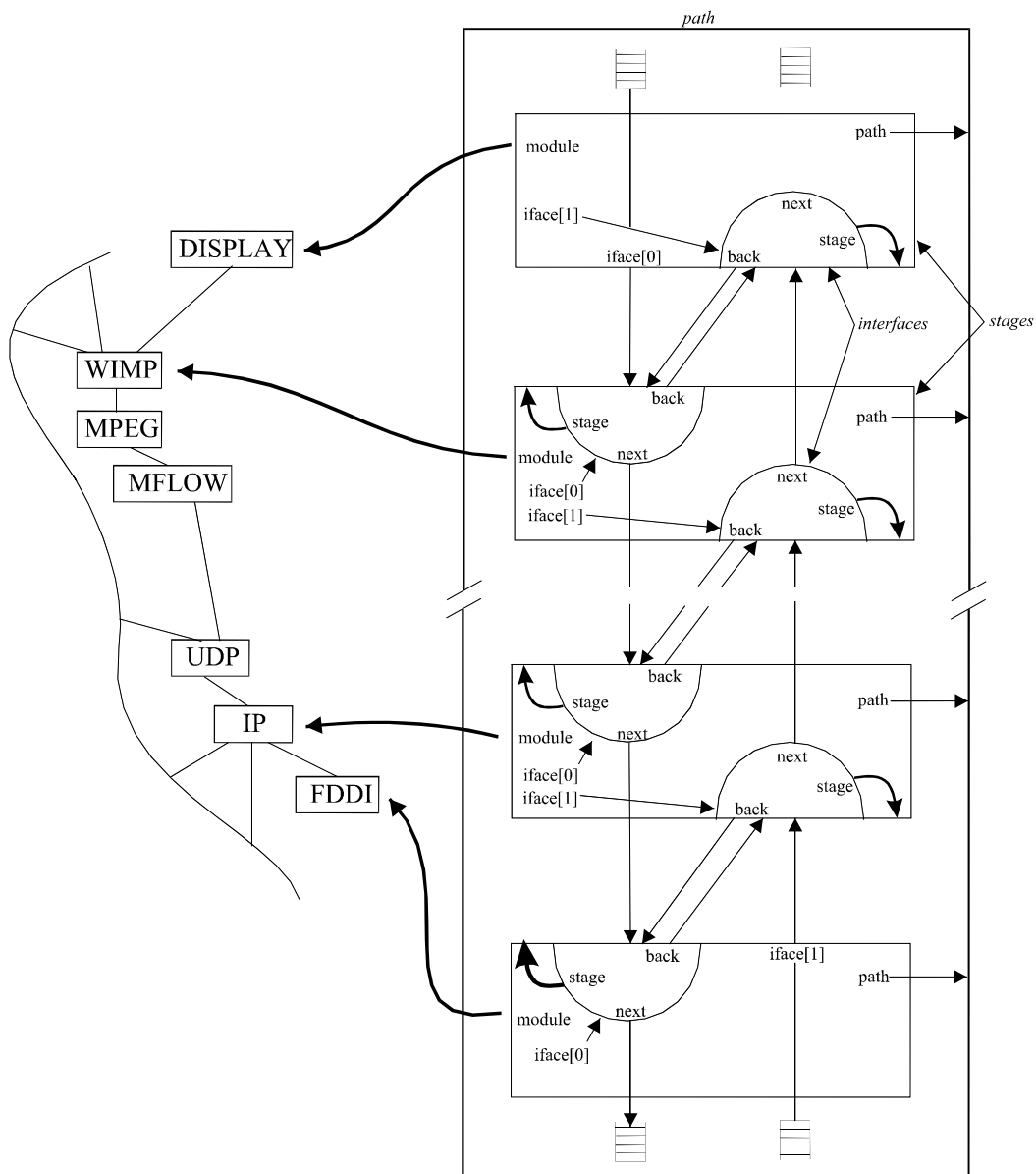
Stage    stage;
} * Iface;

typedef struct AioIface {
    struct Iface i;
    long    (*deliver)(Iface i, Msg m);
} * AioIface;

```

Jak widać przy użyciu funkcji dostarczania `deliver()` można przekazywać dane między stopniami. Z interfejsu `AioIface` można wyprowadzić interfejs zawierający funkcję modyfikującą maksymalny rozmiar okna nazwany np. `WindowedAioIface`, który może być wykorzystany do przekazywania danych do modułu TCP. Możliwe jest więc pojedyncze dziedziczenie.

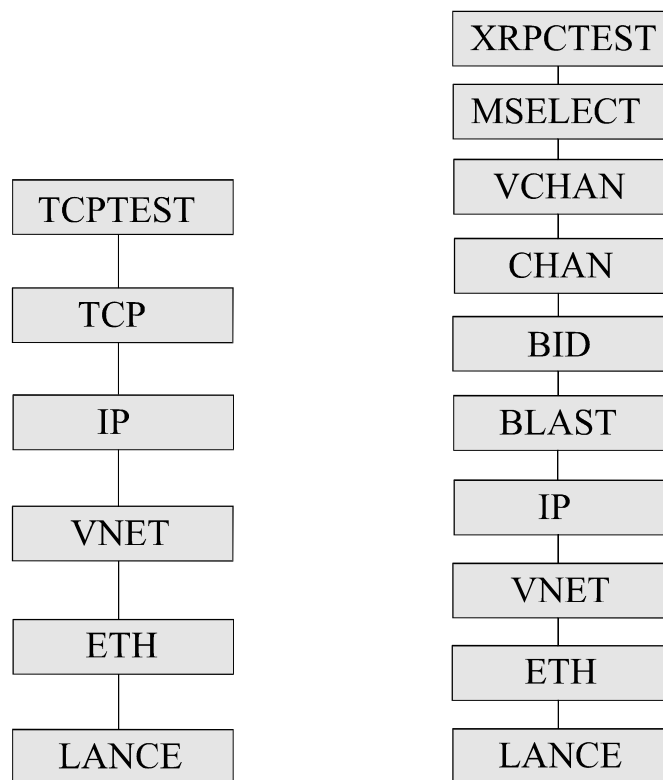
Powiązania pomiędzy modułami, ścieżkami, stopniami i interfejsami przedstawione są na rysunku 4.



Rys. 4 Powiązania między modułami, ścieżkami, stopniami ścieżek i interfejsami

5. Wydajność systemu Scout.

Chcąc pokazać zalety systemu Scout i koncepcji ścieżek autor wykorzystał go do realizacji wydajnej implementacji TCP i RPC. Stos protokołów używany w tej implementacji przedstawiony jest na rysunku 5.



Rys. 5 Stosy protokołów używane w testach

Implementacja bazuje na źródłach BSD. W implementacji autor tej zastosował wiele wyrafinowanych metod optymalizacji, ale do najważniejszej z nich należy niewątpliwie zapobieganie czasochłonnemu kopiowaniu buforów między warstwami stosu protokołów. Zapobieganie takiej sytuacji było możliwe dzięki ścieżkom. Inne metody optymalizacji związane są z dążeniem do maksymalnego wykorzystania pamięci podręcznej pierwszego poziomu procesora. Procesor Alpha bardzo aktywnie współpracuje z pamięcią cache. Jednak optymalizacja związana z maksymalizacją wykorzystania cache'u nie będzie tutaj omawiana. Na potrzeby testów użyto dwóch stacji roboczych DEC 3000/600, używających jednych z pierwszych procesorów Alpha – DECchip 21064 taktowanych zegarem 175MHz. Każdy z procesorów wyposażony jest w 8KB pamięci podręcznej pierwszego poziomu dla instrukcji i 8KB dla danych. Stacje robocze posiadały po 2MB pamięci cache drugiego poziomu. Użyte procesory są superskalarne i umożliwiają wykonanie do 2 instrukcji w jednym cyklu zegarowym. Stacje robocze zostały połączone wyizolowaną siecią 10Mbit Ethernet. Test wydajnościowy polegał na wysłaniu 1 bajtowego komunikatu do partnerskiej stacji roboczej i zmierzeniu czasu, który opłynął do momentu otrzymania odpowiedzi. Czas mierzono dopiero po 100000 takich operacji przy pomocy 1KHz zegara, a więc z rozdzielczością 1[ms]. Generowaniem pakietu ping-pong zajmował się moduł TCPTEST i XRPCTEST. Wyniki testów dla dwóch stosów protokołów przedstawione są w poniższej tabeli. Nie stosowano tutaj żadnej optymalizacji związanej z pamięcią podręczną:

Stos protokołów	T _c [μs]
TCP/IP	351.0±0.28
RPC	399.2±0.29

Autor proponuje by od każdej z wartości odjąć po 200[μs], gdyż jest to opóźnienie wprowadzane przez stosowany interfejs sieciowy. Jak widać czas przetwarzania protokołu jest w tej implementacji bardzo krótki.

6. Możliwości zastosowań.

Z uwagi na modułarną budowę system Scout może być użyty w wielu przypadkach wymagających wydajnej komunikacji i specjalizowanego systemu. Do najciekawszych należą:

- a) Zastosowanie systemu Scout do budowy routera IP. Jest to chyba pierwsza nasuwająca się możliwość. Z uwagi na wydajne przetwarzanie protokołu, można stosować system nie tylko do budowy routera, lecz także do budowy firewalla
- b) Końcówka skalowalnego serwera plików (scalable storage server node). Zastosowanie to wydaje się być bardzo ciekawe z uwagi na zapotrzebowanie na rozproszone i dzięki temu niezawodne systemy komputerowe. Obciążenie serwera może być rozkładane pomiędzy kilka komputerów.
- c) Sieciowy firewall. Z uwagi na modułarną i przejrzystą budowę można użyć systemu Scout do budowy bardzo bezpiecznego firewalla odpornego na wszelkiego typu ataki typu buffer-overflow, mające na celu przejęcie kontroli nad systemem przy wykorzystaniu istniejących błędów oprogramowania systemowego, czy ataki DoS (Denial of Service) pozwalające na unieruchomienie systemu poprzez „zalewanie” go dużą ilością pakietów.
- d) Zastosowanie systemu Scout do realizacji koncepcji komputera sieciowego
- e) Użycie systemu Scout do budowy urządzeń bezpośrednio podłączonych do sieci np. kamera sieciowa.
- f) Zastosowania w systemach mobilnych np. personal information manager, telefonia komórkowa

9. Podsumowanie

System Scout jest niewątpliwie systemem ciekawym. Zastosowana tu koncepcja ścieżek jest bardzo interesującą i wygodną metodą realizacji modularności. Zaletą tej metody jest to, że nie zakłada ona żadnej klasy modułów, które muszą istnieć w systemie, tak jak ma to miejsce we współczesnych uniksowych systemach operacyjnych, czy w sztanowym systemie firmy Microsoft MS Windows NT. System może być składany z pojedynczych modułów jak z klocków, a ścieżki sprawiają, że system modułarny nie jest wcale mniej wydajny od systemu monolitycznego, a nawet odwrotnie. Dzięki dobrze zdefiniowanym zależnościom pomiędzy elementami systemu możliwa jest wyrafinowana optymalizacja każdego z modułów. Co więcej, każdy moduł może być tworzony i optymalizowany

oddzielnie przez różne osoby. Koncepcja ścieżek jest otwarta i może być rozszerzona o nowe elementy, jak np. o ścieżki rozproszone tzn. takie, które przebiegają przez moduły na wielu komputerach.

Z punktu widzenia systemów operacyjnych należy powiedzieć, że system Scout posiada pewne braki, z powodu, których nie może konkurować z innymi współczesnymi systemami operacyjnymi do zastosowań specjalizowanych takimi jak np. OS/9, QNX, VxWorks stosowanymi powszechnie w przemysłowych, specjalizowanych systemach komputerowych. Do braków systemu można zaliczyć pobłażającą strategię szeregowania, brak domen ochrony, brak możliwości doładowywania modułów w czasie działania systemu. Jednak należy pamiętać o tym, że Scout powstał w celu praktycznej realizacji koncepcji ścieżek i z tego punktu widzenia spełnia wszystkie wymagania. Autor systemu David Mosberger deklaruje, że wszelkie niedoróbki systemu zostaną usunięte w przyszłych jego wersjach. Scout z uwagi na nowoczesną architekturą zasługuje niewątpliwie na uznanie.

Literatura:

D. Mosberger „SCOUT: A PATH-BASED OPERATING SYSTEM”, The University of Arizona, 1997