ROZPROSZONE SYSTEMY OPERACYJNE
Dokumentacja końcowa
Realizacja niezawodnych usług w OpenSSI

Marcin Najs

16 czerwca 2005

Spis treści

1 Cel i zakres realizowanego projektu

Celem projektu było wykorzystanie mechanizmów OpenSSI do realizacji niezawodnych usług. Podstawowym założeniem było wykorzystanie demona ***spawndaemon*** dla funkcji ***keepalive*** w celu monitorowania życia przykładowego serwera. Ze różnych względów nie powiodła się jednak konfiguracja i prawidłowe uruchomienie demona, dlatego też założenia projekowe uległy znacznym zmianom. Aby zademonstrować działanie niezawodnej usługi został zaimplementowany prosty serwer czasu oraz aplikacja kliencka. W celu komunikacji serwera z klientem zostaje zestawione połączenie TCP. Wysyłane zostają żądania klienta co do informacji, którą powinien otrzymać, a następnie klinent otrzymuje odpowiedz w postaci godziny, daty, roku, miesiąca, dnia roku lub dnia miesiąca. W programie serwera działa funkcja, która tworzy procesy na wszystkich innych węzłach klastra. W momencia przerwania dzialania serwera lub fizycznej awarii węzła w klastrze programy serwera działające juz na innych węzłach, wybierają spośród siebie tego, który przejmie rolę serwera świadczącego usługi dla klienta. Dla programu klienta sytuacja awaryjna, która zdarzyła się na węzle serwera jest niezauważalna, przezroczysty jest również sam proces wyboru nowego węzła, na którym działał będzie serwer. Klient powinien mieć możliwość dalszej komunikacji z serwerem i bez przeszkód odbierać informacje. Jeden serwer jest w stanie obsługiwać wielu klientów.

2 Implementacja

2.1 Program klienta

- Przy pomocy funkcji ***socket()*** tworzone jest gniazdo do zestawienia komunikacji z serwerem.
- Przy pomocy funkcji ***connect()*** program kliencki zestawia połączenie z serwerem, na określonym przez użytkownika adresie oraz porcie, na którym nasłuchuje serwer.
- Jeśli utworzenie gniazda lub nawiązanie połączenia nie powiedzie się, program klienta kończy działanie podając odpowiedni błąd na wyjście.
- Po nawiązaniu połączenia z serwerem klient rozpoczyna proces wysyłania i odbioru informacji przy wykorzystaniu funkcji ***send()*** oraz ***recv()***.
- Program kliencki znajduje się w pliku time_client.c.

2.2 Program serwera

- Przy pomocy funkcji ***socket()*** tworzone jest gniazdo do zestawiania połączenia z klientem.
- Przy pomocy funkcji ***bind()*** dokonane zostaje powiązanie adresu z gniazdem.
- Serwer blokuje się na funkcji ***listen()***, oczekując na połączenia od klientów.
- Serwer przyjmuje połączenie przychodzące od klienta używając do tego funkcji ***accept()***.

- Kiedy połączenie z klientem zostaje nawiązane, przy pomocy funkcji ***fork()***, dla każdego łączącego się z serwerem klienta zostaje utworzony nowy proces.
- Gdy proces zostaje utworzony rozpoczyna się proces wymiany informacji między serwerem a klientem za pomocą funkcji ***send()*** oraz ***recv()***.
- Na początku procedury obsługi procesu potomnego zostaje wywołana funkcja ***podtrzymuj()***, która pełni kluczową rolę jeśli chodzi o zapewnienie niezawodnej pracy serwera.
- Funkcja podtrzymuj() na początku swojego działania sprawdza stan węzłów w klastrze. W programie serwara z góry założona jest liczba określająca numer węzła, na który ma zostać zmigrowany proces serwera. Liczba ta jest podawana jako argument dla funkcji ***clusternode_avail()***, która sprawdza czy dany węzeł klastra jest w stanie aktywnym. Jeśli węzeł nie jest aktywny, następuje wyjście z funkcji. Następnie przydzielana jest pamięć dla tablicy węzłów za pomocą funkcji ***malloc()***. Dla określenia liczby węzłów wykorzystana została funkcja ***cluster_maxnodes()***. Następnie za pomocą funkcji ***cluster_membership()*** otrzymywana jest tablica zawierająca numery wszystkich węzłów znajdujących się w stanie aktywnym. Wszyskie powyższe czynności służą do pobrania podstawowych informacji o stanie węzłów w klastrze.
- Dla każdego aktywnego procesu tworzona jest tablica par sprzężonych ze sobą deskryptorów plików przy pomocy funkcji ***pipe()***, które służyć będą do komunikacji między procesami (replikami serwera).
- Przy pomocy funkcji ***rfork()*** na każdym aktywnym węzle tworzony jest proces. Jeśli proces potomny zostanie utworzony wywoływana jest dla niego funkcja ***Wezel()*** realizująca komunikację między procesami na węzłach oraz wybór między nimi procesu, który przejmie rolę serwera.
- Każdemu procesowi przyporządkowana jest liczba określająca jego pozycjie w hierarchi procesów. Dany proces wysyła informacjie do wszystkich procesów o niższym priorytecie. Proces, któremu przydzielona jest najmniejsza liczba nie jest w stanie nawiazac komunikacji z procesem o niższym numerze. Wtedy dany proces przejmuje rolę serwera. Sytuacja odłączenia serwera sygnalizowana jest poprzez wysłanie sygnału **SIGHUP**. Serwer posiada procedurę obsługi tego sygnału ***handlerS()***. Obsługa sygnału zrealizowana jest dzięki użyciu funkcji ***signal()***.
- Wywoływana jest następnie funkcja ***migrate()***, która przenosi proces na wybrany węzeł w klastrze.
- Program serwera znajduje się w pliku time_server.c.

3 Podstawowe informacje dotyczące problemów niezawodności i tolerowania uszkodzeń w systemie rozproszonym

3.1 Redundancja

Ogólnym podejściem do tolerowania uszkodzeń jest użycie redundancji. Możemy wyróżnić trzy podstawowe rodzaje redundancji:

- Redundancja informacji
- Redundancja czasu
- Redundancja fizyczna

Redundancja fizyczna powstaje poprzez dodanie specjalnego wyposażenia aby system jako całość mógł tolerować utratę pewnych składowych lub ich wadliwe działanie.

3.2 Aktywne zwielokrotnienie

Redundancja fizyczna może być osiągnięta poprzez zastosowanie replikacji. Jedną z metod replikacji jest **aktywne zwielokrotnienie** (ang.*active replication*). Działanie aktywnej replikacji można zobrazować krótkim przykładem. Jeśli mamy urządzenie składające się z trzech podzespołów pracujących szeregowo (sygnał wejściowy przetwarzany jest kolejno przez każdą składową) to aktywnym zwielokrotnieniem będzie powielenie każdego z podzespołów. Załóżmy, że każde z urządzeń zostało potrojone. Informacje wyjściowe pochodzące z każdej kopii urządzenia są porównywane przed wejściem do kolejnego podzespołu. Informacje wyjściową z podzespołu uznaje się za poprawną jeśli większość kopii tego podzespołu da tę samą informację. W ten sposób powstaje system, w którym awaria pojedyńczego podzespołu zostaje ukryta przez prawidłowe działanie jego kopii. Technika opisana nosi nazwę TMR (ang.*Triple Modular Redundancy* - **potrójna redundancja modularna**).

3.3 Zastosowanie zasobów rezerwowych

Kolejną metodą zastosowania redundancji może być wprowadzenie zasobów rezerwowych. Podejście to polega na tym, że w jednym czasie prace wykonuje jeden serwer podstawowy. Jeśli ulegnie on awarii, to jego funkcję przejmuje serwer rezerwowy. Do zalet tego rozwiązania można zaliczyć : uniknięcie problemów porządkowania komunikatów i zmniejszenie liczby wymaganych maszyn w stosunku do poprzedniego sposobu (replikacji). Wadą takiego systemu jest złe działanie w przypadku wystąpienia wad bizantyjskich.

4 Mechanizmy dla zapewnienia niezawodności usług w OpenSSI

4.1 Interfejsy obiektów

Oprócz standardowych interfejsów dla obiektów, OpenSSI dostarcza interfejsy dla przemieszczenia i lokacji procesów między węzłami klastra. Do takich mechanizmów można zaliczyć funkcje *migrate*, *rexec* i *rfork*.

4.1.1 Funkcja migrate

Funkcja *migrate()* używana jest do przenoszenia jednego lub więcej procesów na wybrany węzeł. Argumentami komendy są: węzeł oraz identyfikator procesu (pid). Jeśli podany pid jest ujemny, komenda przeniesie wszystkie procesy z grupy. Jeśli natomiast podany identyfikator procesu jest częścią grupy wątków, to cała ta grupa będzie przeniesiona. Migracja procesu wyspecyfikowanego przez identyfikator procesu jest wykonywana przez zapisanie numeru węzła podanego jako argument do pliku */proc/pid/goto*. Proces, który ma ulec migracji musi być procesem użytkownika chyba, że użytkownik jest uprzywilejowany. Listę rodzajów procesów, które nie mogą zostać przeniesione za pomocą funkcji migrate można znalezć w dokumentacji (*man migrate*).

4.1.2 Funkcja rexec

Funkcja *rexec* służy do wykonania zdalnego wywołania pliku na określnym węzle. Należy ona do rodziny funkcji, które zastępują dotychczasowy obraz procesu nowym obrazem procesu. Działa ona analogicznie do funkcji *exec* z tą różnicą, że specyfikuje ona węzeł, na którym obraz procesu zostanie uruchomiony. Jeśli proces, który próbujemy utworzyć działa już na wyspecyfikowanym węzle, to nowy obraz procesu będzie działał na tym samym węzle co oryginał, a funkcja *rexec* da ten sam efekt co jej odpowiednik *exec*.

4.1.3 Funkcja rfork

Funkcja *rfork* służy do utworzenia procesu potomnego na określonym węzle. Jest ona wersją wywołania systemowego *rfork*. Proces potomny tworzony jest w węzle określonym przez argument funkcji. Jeśli proces macierzysty działa już na określonym węzle, to proces potomny zostaje uruchomiony na tym samym węzle co proces macierzysty, a funkcja spełnia tę samą rolę co *fork*. Jeśli jako argument określający węzeł podamy CLUSTERNODE_BEST, wybrany zostanie najmniej obciążony węzeł.

4.2 Dostępność aplikacji

Do mechanizmów zapewniających dostępność aplikacji dostarczonych przez OpenSSI można zaliczyć demony: ***spawndaemon*** oraz ***keepalive***.

4.2.1 Keepalive

Demon *keepalive* monitoruje procesy i demony, które są zarejstrowane przy użyciu demona *spawndaemon*. Kiedy zarejestrowany proces lub demon zawodzi, *keepalive* zapisuje zdarzenie używając do tego funkcji *syslog* i wywołuje skrypt restartujący proces lub demon. Do podstawowych cech demona *keepalive* można zaliczyć:

- Wprowadzenie interfejsu wiersza poleceń (*spawndaemon*) z wieloma opcjami dla restartowania procesów/demonów.
- Monitorowanie procesów czasu rzeczywistego
- Monitorowanie demonów
- Restartowanie procsów/demonów na dowolnym węzle w klastrze, w sposób określony przez użytkownika

Demon *keepalive* używa standardowych skryptów powłoki do restartowania procesów/demonów, które zostały przerwane.

4.2.2 Spawndaemon

Komenda *spawndaemon* uruchamia interfejs wiersza poleceń dla demnona keepalive, który monitoruje procesy i demony, restartując je kiedy umierają. Keepalive może monitorować procesy i demony indywidualnie oraz w grupach. Spawndaemon wykonuje kilka podstawowych zadań:

- Przetwarza opcje z wiersza poleceń oraz związane z nimi pliki konfiuracyjne
- Wysyła komunikaty do demona *keepalive*, który następnie wykonuje wszystkie zadania związane z uruchomieniem i monitorowaniem procesów
- Czyta informacje z tablicy procesów monitorowanych przez *keepalive* i wyświetla te informacje dla administratora systemu

4.3 Mechanizmy replikacji

OpenSSI dostarcza usługę replikacji zwaną ***Kernel Data Replication Service***. Dostępne są jedynie kody źródłowe dla tej usługi. Narazie nie jest ona używana przez żadne podstystemy. W oparciu o dotychczasową wiedzę na temat tego rozwiązania nie jestem w stanie stwierdzić czy przyda się w realizacji projektu.

5 Podstawowe informacje dotyczące algorytmów elekcji

W przypadku awarii serwera, procesy monitorujące znajdujące się na innych węzłach powinny zdecydować między sobą, na którym z węzłów serwer zostanie ponownie uruchomiony. Do tego celu można wykorzystać algorytmy elekcji. Jeden proces po wyborze będzie pełnił rolę inicjatora usługi. Można założyć, że każdy z procesów monitorujących posiada swój unikalny numer. Algorytmy elekcji najczęściej próbują zlokalizować proces o najwyższym numerze i mianować go nowym inicjatorem. Różnią się sposobami dokonania tej lokalizacji. Zakłada się również, że każdy z tych procesów zna numery wszystkich pozostałych procesów. Procesy nie mają informacji o tym, które z nich w danym momencie działają, a które są unieruchomione. Przykładami takich algorytmów mogą być:

- ***Algorytm tyrana***. Proces, który zauważył, że koordynator przestał działać wysyła komunikat do wszystkich procesów z większymi numerami. Jeśli nikt nie odpowie to proces ten zwycięza w wyborach. Jeśli, któryś z procesów nadeśle odpowiedz to on przejmuje kontrolę
- ***Algorytm pierścieniowy***. Każdy proces z grupy zna swojego następce. Gdy brak działania koordynatora zostaje wykryty, procec wysyła komunikat do swojego następcy dołączając swój numer. Jeśli komunikat wróci do nadawcy to zostaje on nowym koordynatorem.