Dokumentacja SKJ4.1

Zadanie 3 tunel – wstępna dokumentacja

Co zostało zrealizowane:

* Struktura programu:
  + App
    - Klasa startowa, odpowiada za trzymanie zmiennych używanych przez interface i identyfikacji danej instancji aplikacji.
  + Chat
    - Klasa odpowiedzialna za obsługę czatu oraz komend.
  + Gate
    - Klasa odpowiedzialna za obsługę całej komunikacji z zewnątrz, zarówno przez UDP jak i TCP.
  + NetPath
    - Klasa służąca jako opakowanie dla adresu sieciowego (ip-port).
  + Packet
    - Klasa wykorzystywana jako medium do komunikacji pomiędzy klientami, jej obiekty są wysyłane i odbierane zarówno przez TCP jak i (opakowane w DatagramPacket) UDP. Jej metody statyczne natomiast służą ukryciu przed resztą aplikacji sposobu działania celem uproszenia pisania programu.
* Funkcjonalność:
  + Protokół zawarty w aplikacji pozwala na przesyłanie pakietu przez dowolnie dużo pośredników dowolnymi metodami, (jakkolwiek nie jest to w tym stopniu z poziomu interface’u użytkownika). W chwili obecnej dostępne dwie komendy pozwalają na:
    - Nawiązanie połączenia z aplikacji do agenta końcowego za pośrednictwem najpierw UDP a następnie TCP (Aplikacja<=UDP=>Pośrednik<=TCP=>Agent).
    - Nawiązanie połączenia na „odwrót” (Aplikacja<=TCP=>Pośrednik<=UDP=>Agent).

Co nie zostało zrealizowane:

Program nie wykorzystuje funkcji Exec celem optymalizacji zarządzania wątkami oraz ich bezpiecznego przerywania (co zostało skompensowane przy pomocy TimeOut gdzie konieczne). Ograniczona funkcjonalność dopuszcza tylko komunikacje w IPv4. Czat jest bardzo wrażliwy na błędy składni czy znaki białe przy wprowadzaniu poleceń. Aplikacja nie ma zabezpieczeń na wypadek prób więcej niż jednego połączenia przez klientów końcowych (Pośrednik może przepuszczać dowolnie wiele linii komunikacyjnych ale każdy agent końcowy może mieć tylko jedno połączenie, czy to UDP czy TCP).

Ponadto program realizuje funkcjonalność odpowiadającą tej z dokumentacji a nie dokładnie taką, tak na przykład jest nie wrażliwy na argumenty startowe, wykorzystuje interface graficzny, nie kończy swojej pracy w przypadku niepowodzenia nawiązania połączenia tylko czeka dalej na kolejne polecenie itd.

Jak zainstalować:

Program jest mobilny t.j. gdziekolwiek się nie znajdzie będzie funkcjonował poprawnie.

Program powinien (z racji zwracania wyjątków nie-krytycznych do czatu) wyjaśniać potęcjalne błędy składni (jednak NIE błędy logiczne poleceń). O ile nie nastąpi nadużycie komend połączenia /Lew/Rig (np. połączenie z dwoma klientami na raz; połącznie z samym sobą) wszysko powinno dziłac bezbłędnie.

Ewentualne błędy:

Program wykorzystuje JDK 1.9 (ze względu na część FX aplikacji) ponadto w chwili obecnej obsługuję komunikacje tylko przy pomocy IPv4. Błędy jeżeli jakieś się pojawią z zasady jeżeli nie zakończyły działania programu (co zostało przewidziane w wszystkich znalezionych błędach krytycznych) mają charakter: (w konsoli) logu, oraz (w okienku czatu) informacyjny (celem wyjaśnienia np. dlaczego dana komenda się nie wykonała). Daję się zauważyć znaczna (z nieznanych mi powodów) ilość błędów przy starcie programu i losowaniu jego unikalnego adresu.

Opis protokołu:

Protokół komunikacji opiera się w całości o klasę Packet z następującą strukturą danych:

//LOCAL VAR  
 public static final int *SNDMSG*=1,*SNDFED*=2,*DIS*=3,*CON*=4,*REL*=5;  
 public static NetPath *defRelay*=null,*defReciver*=null;  
 public static boolean *tcpSideOfLink*=false;  
//============================================================= CONTENT NEW  
 public final int whatDoIcarry;  
 public final NetPath sender, reciver;  
//============================================================= NAV  
 public final boolean killGateOnExitEnter,udp;  
 public final String message;  
 public final ArrayList payload = new ArrayList();

1 = wiadomość => wiadomość do danego adresata;

//2 = zawartość czatu => chwilowo nie w użyciu

3 = wiadomość o odłączeniu się od klienta/serwera;

4 = połączenie => pakiet trasujący konfigurujący połącznie;

5 = pośrednik => pakiet zawierający inny pakiet jako swoją zawartość, po dotarciu do celu wysyła go. Można go składać wielokrotnie i w ten sposób wybrać dowolną trasę oraz parametry transmisji.

defRelay pośrednik do którego wysyłane są wiadomości pośrednie;

defReciver klient końcowy będący odbiorcą pakietu zawartego w pakiecie pośredniczącym wysłanym do defRelay

Gdzie whatDoIcarry stanowi informację o tym co zwiera dany pakiet;

Sender, reciver są odpowiednio adresami nadawcy i odbiorcy wiadomości;

killGateOnExitEnter odpowiada za bezpieczne zamykanie połączenia TCP bez przerwania;

udp informuję klasę Gate o sposobie wysłania wiadomości;

message zawiera informacje w formacie String, w tej implementacji tylko wiadomości od czatu do czatu.

payload lista typu Object do transportu różnych rzeczy (adresów, pakietów)

Przykład inicjalizacji połączenia Aplikacji z serwerem (uproszczona wersja połączonych logów klienta i serwera):

* Klient 1 otrzymuję polecenie /Lew @K2 @K3
* K1 wysyła wiadomość: new Packet(*CON*,App.*me*,new NetPath(ip0,po0),false,false,null,true ,0,new NetPath(ip1,po1))

Gdzie *CON* to typ wiadomości (połączeniowa);

App.*me* to nadawca

new NetPath(ip0,po0) to odbiorca (pośrednik)

false to informacja czy zamknąć Gate po przejściu

false to informaca czy pakiet jest wysyłany UDP

null to wiadomość tekstowa

true to informacja czy to komenda Lew

0 to informacja który krok połączenia następuje w tym monecie

new NetPath(ip1,po1) to adres obiorcy końcowego

* K2 odbiera wiadomość, krok wiadomości zostaje zwiększony do 1, typ wiadomości (TCP/UDP) zostaje zanegowane; adres obiorcy-pośrednika i odbiorcy-końcowego zostają zamienione miejscami
* K2 wysyła wiadomość dalej
* K3 odbiera wiadomość, krok wiadomości zostaje zwiększony do 2, adres nadawcy zostaje zamieniony z adresem nadawcy, a adres nadawcy z payload (tj. nadawcą zostaje K3, odbiorcą pośrednim K2, a końcowym K3), JEŻELI odbiorca nie jest już przyłączony, zostaje skonfigurowany: *defRelay*, *defReciver*, *tcpSideOfLink*; a program jest kontynuowany w przeciwnym przypadku pakiet przepada nic nie robiąc.
* K3 wysyła wiadomość dalej
* K2 przekazuje wiadomość dalej, tak samo jak poprzednio
* K1 odbiera wiadomość, następuje konfiguracja połączenia: *defRelay*, *defReciver*, *tcpSideOfLink*;
* Połącznie skonfigurowane i aktywne. =============================================
* Przykład wysłania wiadomości z K1 do K3:
* K1 wysyła wiadomość:

new Packet(REL , App.me, defRelay, false, false, null,

new Packet(SNDMSG, App.me, defReciver, false, true , massage))

Tj. Do pośrednika zostaje wysłana wiadomość przekaźnikowa, zawierająca jako ładunek kolejną wiadomość, tym razem już właściwą.

* K2 odbiera pakiet, wyjmuje jego zawartość PayLoad i wysyła ją dalej.
* K3 odbiera wiadomość

-=Koniec przykładu=-

Aplikacja w miarę działania wypisuję w „logu” tj. terminalu z którego zostały uruchomione akcje podejmowane przez Gate tym samym stanowiąc wierny zapis komunikacji; Pozwala to na obserwację weryfikacje funkcjonowania protokołu.