Projekt aplikacji „Klucznik” (architektura klient-serwer)

Projekt: Obliczenia rozproszone
Wykonawcy: Damian Hintz, Łukasz Tomczyk, Jarosław Grzyb
Temat: Łamanie haseł metodą brute-force

Cel i założenia

Naszym celem stworzenie aplikacji do złamania zakodowanego hasła przy pomocy jakiejś znanej funkcji np. MD5. W tym celu będziemy generować wszystkie możliwe hasła aż do znalezienia rozwiązania lub do przeszukania całej badanej przestrzeni. Z opisu można słusznie wywnioskować, że zostanie wykorzystana metoda brute-force. W tym celu będziemy musieli ustalić alfabet użyty do generowania haseł oraz hasła jakie będziemy badać (w tym jak długie powinny być hasła). Zakładamy, że w alfabecie będzie mógł znaleźć się każdy normalny znak ASCII, który można wprowadzić z klawiatury. Długość haseł natomiast będzie ograniczona do 255. Jednak w praktyce tak długich haseł nie będziemy spotykać więc hasła jakie będziemy badać będą prawdopodobnie o maksymalnej długości kilkunastu znaków.

Założenia dodatkowe

W trakcie realizacji projektu doszliśmy do wniosku, że aplikacja nie musi ograniczać się jedynie do wyszukiwania haseł, ale wykonywać również inne zadania o podobnej właściwości do wyszukiwania haseł. Wprowadzane zmiany do projektu miały na uwadze właśnie umożliwienie rozszerzenia funkcjonalości aplikacji o dodawanie również innych zdań.

Wymagania funkcjonalne

Serwer

1. Sterowanie działaniem serwera, tj. uruchamianie oraz zatrzymywanie.
2. Dodawanie i usuwanie zadań do listy zadań wykonywanych na serwerze.
3. Sterowanie stanem poszczególnych zadań tj. aktywowanie i wstrzymywanie.

Klient

Od klienta wymagamy jedynie połączenia się z serwerem, rozpoczęcia wykonywania zadań oraz możliwość zatrzymania w dowolnej chwili wykonywanych działań. Klient ma za zadanie wykonanie części zadania przekazanego przez serwer i zwrócenie jego wyniku na serwer.

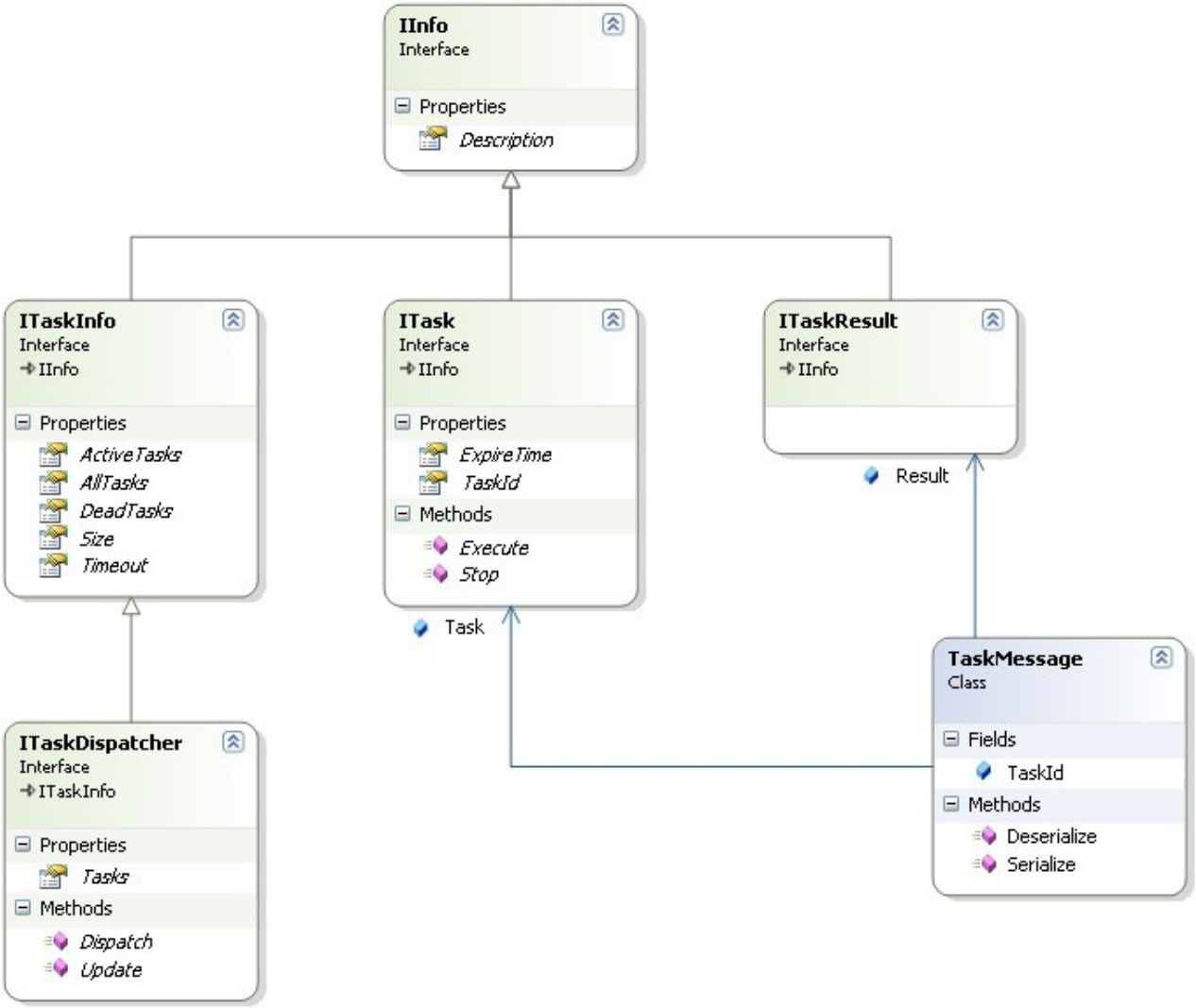
Projekt

Projekt aplikacji został podzielony na trzy główne podprojekty. Projekt serwera i klienta, oraz projekt wspólny. Aplikacja została zaprojektowana w taki sposób, aby była możliwość prostego rozszerzenia jej o wykonywanie nowego rodzaju zadań.

Projekt wspólny składa się z głównego modułu oraz modułu realizującego wykonywanie zadań poszukiwania haseł. Możemy również dodać do projektu nowe moduły implementujące wybrane interfejsy zdefiniowane w głównym module, w celu dodania możliwości realizacji zadań nowego rodzaju.

Moduł „Główny”

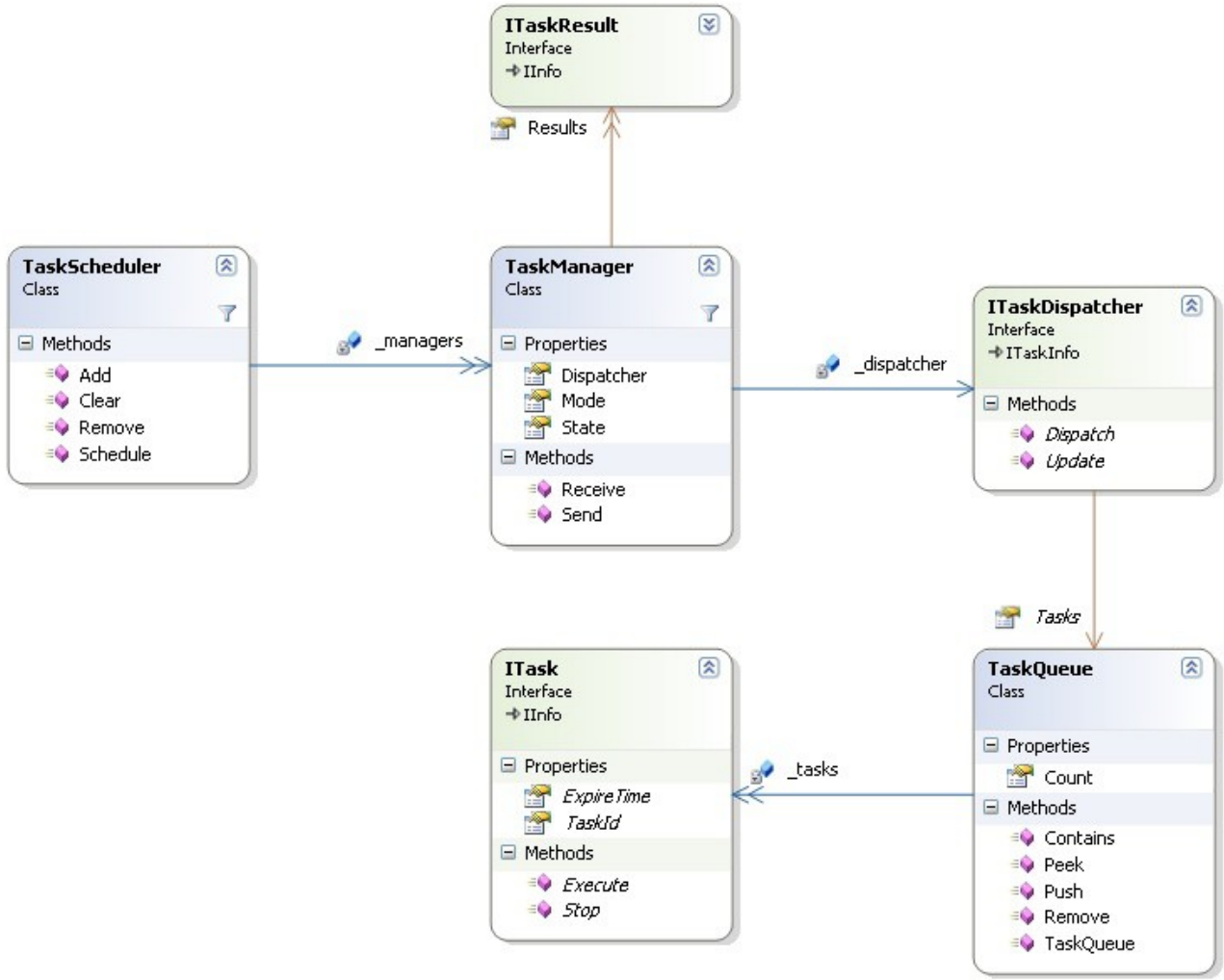
Poniżej znajduje się diagram klas z głównymi interfejsami w głównym module.



Interfejs Iinfo nie ma żadnych właściwości poza Description, które ma umożliwić nam uzyskanie szczegółowego opisu o stanie danego obiektu. Jak widać kilka różnych interfejsów korzysta z tego interfejsu. Na diagramie widzimy również jeszcze trzy inne interfejsy, które powinny nas zainteresować tj. Itask, ItaskResult, oraz ItaskDispatcher. Te trzy interfejsy są

interesujące z tego powodu, że aby dodać nowy rodzaj zadań powinniśmy wprowadzić do projektu właśnie klasy realizujące te trzy interfejsy. Itask zawiera pola określające czas jaki dajemy zadaniu na wykonanie oraz jego identyfikator. Zawiera również dwie metody Execute, do rozpoczęcia wykonania zadania oraz Stop, do jego zatrzymania. Obiekt klasy realizującej ten interfejs otrzyma klient w celu wykonania zadania. Po wykonaniu zadania, jeżeli był jakiś wynik zadania, klient odsyła do serwera rezulat w obiekcie realizującym interfejs ItaskResult, który zawiera jedynie opis rozwiązania. Obiekty obu klas są przekazywane między klientem oraz serwerem w klasie TaskMessage. Klasa ma trzy pola TaskId, będący identyfikatorem zadania, Task będący obiektem zadania oraz możliwy rezultat, obiekt Result. Ostatnim najważniejszym interfejsem jest ItaskDispatcher, który dokonuje podziału głównego zadania. Obiekt realizujący ten interfejs zawiera kolejkę zadań aktywnych o nazwie Tasks, które zostały wysłane do klienta, oraz dwie metody Update, do aktualizacji stanu zadania oraz Dispatch, która wybiera kolejne zadanie do wykonania. ItaskDispatcher ma też kilka pól opisujących właściwości zadania tj. liczbę wykonanych, aktywnych oraz wszystkich zadań, a także rozmiar zadania i czas na wykonanie zadania przez klienta.

Poza opisanymi interfejsami znajdują również klasy zarządzające zadaniami. Poniżej mamy diagram z tymi klasami.



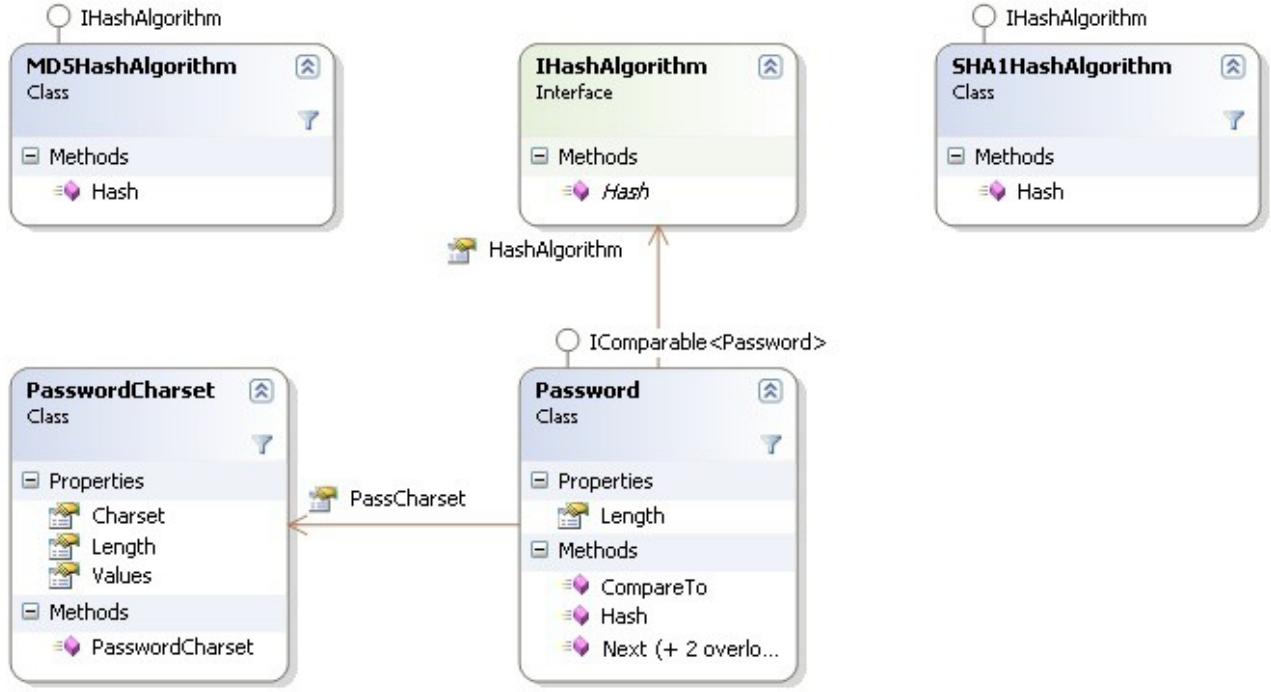
Klasa TaskScheduler zajmuje się wybieraniem kolejnego zadania do wykonania. Tutaj możemy dodawać nowe zadania oraz je usuwać, do czego służą metody Add oraz Remove. Jednak najbardziej interesującą jest Schedule, która rzeczywiście wybiera kolejne zadanie, które zostanie wysłane do klienta.

Każde zadanie jest zarządzane przez klasę TaskManager, która otrzymuje komunikat Receive od klasy TaskScheduler o tym że ma zostać wykonane, a następnie odsyła komunikat Send z kolejnym podzadaniem jakie będzie wykonywane. Kolejne podzadanie zostaje przekazane przez obiekt TaskDispatcher, który jedynie generuje kolejne podzadania do wykonania, a TaskManager zarządza stanem całego zadania. Zarządca zadań może działać w dwóch trybach, SearchOne i SearchAll. SerachOne oznacza, że będzie interesować nas tylko pierwsze znalezione rozwiązanie, a zarządca przejdzie wtedy w stan Completed, określający że zadanie zostało zakończone. W trybie SearchAll, zarządca będzie kontynuował działanie nawet pomimo znalezienia rozwiązania, aż do wykonania wszystkich zadań. W tym celu utrzymuje listę znalezionych rozwiązań Results.

Należy jeszcze wspomnieć o klasie TaskQueue, która zawiera listę aktywnych podzadań. Z każdym zadaniem powiązany jest czas jaki klient ma na jego wykonanie. Każde wysyłane zadanie jest generowane lub pobierane właśnie z kolejki zadań. Jednak zadania są przydzielane najpierw w takiej kolejności, że wybierane są te które przekroczyły przydzielony im czas oraz wybierane jest to zadanie, które jest najstarsze. Jeżeli nie ma takich zadań to generowane są nowe.

Moduł „Password”

Głównym zadaniem aplikacji miało być łamanie haseł, więc w tym celu zostały zaimplementowane potrzebne klasy, które możemy zobaczyć na diagramie poniżej.



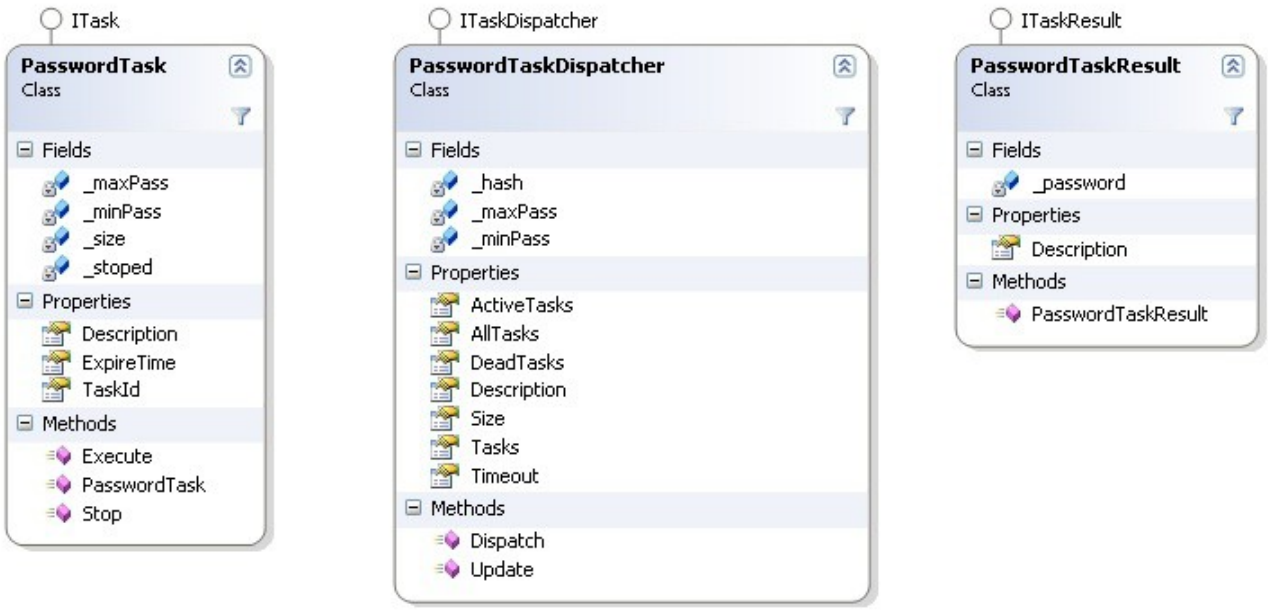
Na diagramie możemy zobaczyć główną klase Password do generowania kolejnych haseł, metodą Next, której możemy przekazać parametr będący liczbą całkowitą określającą liczbę haseł jakie klient będzie musiał sprawdzić. Metoda Next generuje nam po prostu hasło oddalone od bieżącego o daną odległość. Kolejne hasła generowane są na podstawie zbioru znaków zdefiniowanego w klasie PasswordCharset, a do obliczania wartości hash hasła używamy metody Hash. W tym celu klasa Password korzysta z klasy implementującej odpowiedni algorym haszujący, i realizującej interfejs IhashAlgorithm. W projekcie zostały zaimplementowane dwie kalsy haszujące tj. MD5HashAlgorithm oraz SHA1HashAlgorithm.

Password jest klasą realizująca działania na haśle, które umożliwią nam generowanie kolejnych haseł do sprawdzenia (wykorzystanie tu zostanie między innymi arytmetyka dodawania liczb w systemie pozycyjnym o podstawie wyznaczonej przez rozmiar użytego alfabetu, czyli będziemy traktować hasła jak liczby).

Można sobie zadać pytanie jak Password realizuje generowanie kolejnych haseł. W tym celu każde hasło traktujemy jak liczbę. Na podstawie alfabetu dokonujemy konwersji między hasłem w postaci napisu do postaci numerycznej, na której będziemy wykonywać potrzebne działania i odwrotnie. Nie wnikając w szczegóły jak to wszystko będzie działać, dzięki takiemu podejściu będziemy mogli w szybki sposób wygenerować przedział haseł jaki klient będzie miał do sprawdzenia przez zwykłe dodawanie liczb(hasła będą naszymi liczbami).

Password dostarczać będzie także odpowiednich funkcji haszujących używanych do sprawdzania czy dane hasło jest naszym rozwiązaniem.

Aby zadanie poszukiwania hasła mogło być wykonywane musi jeszcze tylko mieć klasy które będą realizować wcześniej wspomniane i wymagane trzy interfejsy. Wszystkie zostały przedstawione na poniższym diagramie.



Klasa PasswordTask, jest zadaniem przekazywanym do klienta w celu jego wykonania i zawierającą niezbędne do tego celu dodatkowe pola tj. minimalne i maksymalne hasło. Mamy również dostosowaną do naszego zadania klasę PasswordTaskDispatcher, które będzie nam generować kolejne zadania, oraz klasę PasswordTaskResult, która będzie zawierać znalezione hasło.

Serwer oraz klient będą się komunikować przez wysyłanie między sobą obiektu klasy TaskMessage. Do generowania haseł posłuży nam klasa TaskPassword, z pomocą której będziemy mogli tworzyć także przedziały haseł, które będą badali klienci. Badane przestrzenie haseł mogą być ogromne i do przedstawiania ich złożoności nie wystarczą zwykłe typy danych. W tym celu wykorzystana zostanie klasa BigInteger, które pozwoli nam realizować arytmetykę bardzo dużych liczb całkowitych (mówimy tutaj o liczbach mających setki, a nawet tysiące cyfr). Serwer będzie musiał w jakiś sposób zarządzać przydzielanymi zadaniami. Zadania będą szeregowane na podstawie czasu wygenerowania. Kolejka taka pozwoli nam szybko odnaleźć zadanie, które było wygenerowane najdawniej, gdyż takie zadania będą w pierwszej kolejności przydzielane klientom (w kolejce mogą znaleźć się zadania, które nie zostały wykonane przez klienta i z tego powodu nie zostały z niej usunięte).

Moduł serwera

Zadaniem modułu serwra jest dostarczenie mechanizmów do przekazywania zadań do klienta i ich odbieranie, czyli realizujący komunikację między nimi oraz dostarczenie interfejsu do sterowania serwerem.

Zadaniem serwera jest podzielenie zadania pomiędzy podłączających się klientów. Każdy podłączający się klient będzie dostawał kolejną porcję zadania do rozwiązania.

Ogólna idea działania serwera będzie polegać na tym, że do serwera będą się przyłączać klienci. Klient będzie wysyłał komunikat do serwera mogący zawierać w sobie już rowiązanie poprzednio otrzymanego zadania. Serwer w takim przypadku będzie aktualizował informację o stanie rozwiązania. Jeżeli klient znalazł rozwiązanie, to cały proces poszukiwania się kończy, jeżeli klient nie znalazł rozwiązania, to serwer oznacza dane zadanie jako zakończone i je usuwa z listy zadań wykonywanych przez klientów. Po czym serwer wyznacza kolejne zadanie, najpierw poszukuje zadań na liscie już wykonywanych przez klientow, takich które są wykonywane przez zbyt długi czas. Jeżeli jakieś znajdzie, to przydziela je nowemu klientowi do wykonania. Jeżeli nie znajdzie takich, to generuje nowe zadanie i je przydziela. Jeżeli nie ma także nowych zadań i wszysko zostalo już przydzielone to klient nie będzie miał przydzielanego żadnego zadania. Serwer utrzymuje listę wygenerowanych zadań przydzielanych klientom, kiedy zadanie zostaje zakończone jest usuwane z listy. Każde zadanie otrzymuje pewien czas jaki ma na wykonanie klient i po którym może być ono przydzielone ponownie innemu klientowi.

Moduł klienta

No cóz, klientowi nie pozostaje nic innego jak połączyć się z serwerem, odebrać zadanie, wykonać je i przekazać wynik z powrotem na serwer i powtarzać ten proces, aż serwer będzie miał aktywne zadania, którymi będzie mógł się dzielić. Do realizacji tego wszystkiego wykorzystywana jest jedynie główna klasa TaskClient.

UDP

UDP (Datagramowy protokół użytkownika) jest chyba jednym z najprostszych ze stosu protokołów TCP/IP. Jego specyfikacja jest długa tylko na trzy strony, format nagłówka zawiera tylko cztery pola(nie wliczając pola danych) i dwa z pól są opcjonalne. UDP dostarcza minimalnej funkcjonalności potrzebnej do komunikacji. Wiadomość UDP zawiera parę pól źródłowych i docelowych portów, pole długości i opcjonalną sumę kontrolną do weryfikacji całej wiadomości UDP wraz z danymi.

Format nagłówka UDP

0 15	16 31
Port źródłowy	Port docelowy
Długość	Suma kontrolna
Dane	

16-bitowe pole portu źródłowego jest opcjonalne, ustawiane na zero w razie braku powiązanego źródłowego procesu, do którego odbierający mógłby odpowiedzieć, jednak często używany nawet gdy pakiet z odpowiedzią nie jest oczekiwany.

16-bitowe pole portu docelowego jest wymagane i identyfikuje zdalny proces po stronie odbierającego.

16-bitowe pole długości określa długość wiadomości UDP włączając w to nagłówek i pole danych. Minimalną długością tego pola jest 8.

16-bitowa opcjonalna suma kontrolna. Aplikacje nie obliczające sumy kontrolnej wypełniają to pole zerami. Odbiorca musi zweryfikować sumę jeżeli jest obecna.

UDP jest bezpołączeniowym, zawodnym protokołem dostarczania komunikatów. To oznacza, że nie ma żadnej procedury do ustanawiania połączenia między dwoma komunikującymi się procesami i wysyłając wiadomość wysyłająca strona zapomina o niej. Po stronie odbierającego, wiadomość UDP jest przekazywana (multipleksowana) do odpowiedniego procesu bez zwracania uwagi na wysyłający host.

UDP nie dostarcza gwarancji wyższej warstwie dostarczenia wiadomości oraz strona wysyłająca nie utrzymuje żadnej informacji o stanie wysłanego komunikatu po jego wysłaniu (z tego względu UDP jest nazywany czasami Zawodnym Protokołem Datagramowym). Jeżeli jest wymagana jakakolwiek niezawodność co do transmitowanej informacji to musi być ona zaimplementowana w wyższych warstwach.

Wobec tego, komunikaty UDP mogą być gubione w wyniku błędów lub przeciążenia sieci bez powiadomienia wysyłającego lub odbierającego. UDP nie gwarantuje niezawodności ani porządku w jakim datagramy docierają, tak jak TCP to robi. Datagramy mogą docierać nie w kolejności w jakiej zostały wysłane, być duplikowane lub ginąć. Jednak unikanie sprawdzania czy każdy pakiet rzeczywiście dotarł do odbiorcy sprawia że UDP jest szybszy i efektywniejszy, przynajmniej dla aplikacji które nie potrzebują gwaracji dotarcia pakietu. UDP jest używane z tego względu w aplikacjach, gdzie porzucone pakiety są bardziej preferowane niż opóźnione. Bezstanowa natura UDP jest użyteczna dla serwerów, które odpowiadają krótkimi komunikatami do dużej liczby klientów.

Dlaczego UDP ?

Szybkość i prostota. A co z zawodnością? Jeżeli chodzi o możliwość docierania pakietów w złej kolejności, to problem ten odpada ze względu na wykorzystanie tutaj prostego modelu do

komunikacji między serwerem i klientem, w którym następuje tylko jedno zapytanie od klienta i jedna odpowiedź od serwera. No tak, ale do klienta i serwera mogą dotrzeć także zduplikowane datagramy. Klient oczekuje tylko jednej odpowiedzi od serwera, po jej otrzymaniu nie czeka na nic więcej, więc nie ma problemu. Jednak serwer w razie otrzymania powtórzonego komunikatu od klienta hmmmmm..(jak rozwiązać ten problem, jeszcze się pomyśli). Jednak najpoważniejszy jest chyba problem zagubionych pakietów (tu też trzeba by się zastanowić głębiej jak się przed tym zabezpieczyć :)).

W projekcie do komunikacji zostanie wykorzystany właśnie protokół UDP, jednak sam projekt powstanie z wykorzystaniem technologii .NET i zakodowany w języku c#.

Komunikacja UDP

Wracając do komunikacji UDP klient-serwer. Podczas komunikacji mogą wystąpić pewne nieprzewidziane sytuacje. Klient może nie otrzymać odpowiedzi od serwera z dwóch powodów. Komunikat może w ogóle nie dotrzeć do serwera lub odpowiedź serwera może nie dotrzeć do klienta. Jeżeli komunikat klienta zostanie zagubiony, to pozostaje ponowne wysłanie komunikatu. Jeżeli zostanie zgubiony komunikat serwera z odpowiedzią do klienta, to serwer będzie musiał go retransmitować.

Sytuacja z zagubionym komunikatem klienta jest prosta i wymaga tylko aby klient po pewnym czasie nie otrzymywania potwierdzenia od serwera, ponowił wysłanie komunikatu. Komunikat zwrotny serwera jest jednocześnie potwierdzeniem otrzymania przez serwer komunikatu od klienta. Jednak co się dzieje gdy serwer otrzymał komunikat od klienta, jednak komunikat zwrotny serwera nie dotarł do klienta. W takim przypadku serwer będzie mógł się spodziewać ponownego komunikatu od klienta. W takiej sytuacji możemy postąpić na dwa sposoby. Możemy potraktować klienta tak jakbyśmy wcześniej nie otrzymali od tego klienta komunikatu i traktować go jak każdego innego i przydzielić mu nowe zadanie lub w jakiś sposób rozpoznać, że ten klient już dostał od serwera zadanie. W takim przypadku serwer będzie musiał odnaleźć zadanie przydzielone temu klientowi i wysłać je ponownie. Pierwszy sposób nie wymaga od serwera żadnych specjalnych działań jednak może powodować gromadzenie na liście zadań, takich które nie będą wykonywane przez żadnego klienta. Jednak po głębszym zastanowieniu pierwszy sposób wydaję się być lepszym, gdyż i tak zadania odkładane na listę zadań wykonywanych w końcu będą przydzielone nowym klientom jak nie temu samemu, którego zadanie zostało zgubione.

Projekt interfejsu

1. Wprowadzenie

System Klucznik jest podzielony na dwa podsystemy (wersja „Serwer” i wersja „Klient”) ze względu na wykonywane przez nich zadania. Dla obydwu przypadków został stworzony graficzny interfejs użytkownika, którego wygląd w jak największym stopniu starano się ujednolicić. Jest to zabieg czysto ergonomiczny, który pozwala na zwiększenie czytelności i przejrzystości aplikacji oraz łatwiejsze „poruszanie się w niej” przez użytkownika. Zróżnicowana funkcjonalność przyczyniła się do bardziej rozwiniętego interfejsu dla podsystemu „Serwer” a zwłaszcza jego panelu konfiguracyjnego. W dalszej części tej dokumentacji zaprezentowano wygląd interfejsu użytkownika systemu Klucznik.