Jakub Dutkiewicz

Inżynieria oprogramowania

Asynchroniczne wzorce programowe – zadania laboratoryjne

Do realizacji poniższych zadań posłuż się środowiskiem Visual Studio. Zadania połączone są z prezentacją, która jest przedstawiana na zajęciach. Jeżeli wykonujesz zadania w domu, posłuż się dokumentacją MSDN.

1. Napisz program, który doda dwa zadania do puli wątków. Każde z zadań po czasie przekazanym w parametrze (stateInfo) wypisze na konsoli wiadomość, która zawiera informacje o tym, ile poczekało. Wykorzystaj operację uśpienia wątku głównego. Wykorzystaj ThreadPool.
2. Napisz prosty program realizujący zadanie klient-serwer. Metody obsługujące klienta i serwer umieść w osobnych wątkach. Dodaj jeden serwer oraz dwóch klientów do puli wątków. Do analizy swojego rozwiązania użyj trybu debugowania.

   Zdefiniuj problemy oraz błędy takiego rozwiązania. Kod źródłowy oraz wnioski zapisz w swoich notatkach.

3. Bazując na rozwiązaniu Zadania 2 napisz program realizujący zadanie klient-serwer. Tym razem, za każdym razem gdy serwer zaakceptuje nowe połączenie utwórz nowy wątek dla tego połączenia. Dodatkowo, każda ze stron po otrzymaniu wiadomości niech wypisze wiadomość „Otrzymałem wiadomość: treść wiadomości”. Wiadomości wysyłane przez serwer powinny mieć kolor czerwony, a wiadomości wysyłane przez klienta powinny mieć kolor zielony.
4. Popraw działanie programu z Zadania 3 używając wyrażenia „lock” (poszukaj sposobu wykorzystania takiej metody w dokumentacji MSDN; statement lock).

   Jaki problem próbujemy rozwiązać? Jakie są problemy związane z tym rozwiązaniem. Wnioski oraz kod źródłowy umieść w swoich notatkach.
5. Napisz program, który pozwala na sumowanie liczb w znajdujących się w tablicy. Rozmiar tablicy podany jest jako argument programu (dla uproszczenia można w ramach laboratorium zapisać ją w zmiennej). Liczby wybierane są losowo. Sumowanie odbywa się na przestrzeni wielu wątków. Liczba wątków zależy od wielkości fragmentu tablicy, która jest również podana jako argument programu. Wykorzystaj następujące narzędzia: AutoResetEvent, WaitHandle.WaitAll, lock, ThreadPool.QueueUserWorkItem, WaitCallback.

   Poeksperymentuj z rozmiarem tablicy oraz jej przetwarzanego przez jeden wątek fragmentu. Wnioski oraz kod źródłowy zapisz.

6. Zaimplementuj w trybie Callback narzędzie czytające dane z pliku. Callback powinien być odpowiedzialny za zamykanie strumienia oraz wypisywanie wiadomości na ekranie. Podpowiedź: Użyj następujących narzędzi: FileStream, BeginRead, AsyncCallback, WaitHandle, AutoResetEvent, WaitOne.

   Czy wątek główny czeka na zakończenie operacji czytania? Czy wątek główny czeka na zakończenie metody callback?

7. Zaimplementuj w trybie EndInvoke narzędzie czytające dane z pliku. Tym razem wątek główny powinien być odpowiedzialny za zamykanie strumienia oraz wypisywanie wiadomości na ekranie. Podpowiedź: nie potrzebujesz metody callback, w jej miejsce możesz podstawić null.

   Czy wątek główny czeka na zakończenie operacji czytania? W porównaniu do czytania synchronicznego, co zyskujesz dzięki takiemu rozwiązaniu?
8. Napisz program, który asynchronicznie wywoła metody rekurencyjnego oraz iteracyjnego obliczania silni oraz kolejnych elementów ciągu Fibonacciego. Sprawdź, które metody zostaną wykonane jako pierwsze. W swoim rozwiązaniu użyj metod BeginInvoke, EndInvoke oraz delegatów. Samodzielnie dobierz tryb APM.
9. Tworzenie narzędzia do mnożenia macierzy zgodnie ze wzorcem EAP
   A) Zaprojektuj klasę CompletedEventArgs dla zdarzenia obsługującego asynchroniczną operację MatMulAsync w ramach operacji asynchronicznego mnożenia macierzy. Rozważ implementację klasy ProgressChangedEventArgs. Przyjmij założenie, że mnożone będą macierze o rozmiarze $n \times n$.
   B) Utwórz klasę MatMulCalculator i rozpocznij budowę klasy zgodnie z wzorcem EAP (MSDN)
      a. Zadeklaruj delegat dla zdarzenia MatMulCompleted
      b. Zadeklaruj zdarzenie MatMulCompleted
      c. Zadeklaruj HybridDictionary, przechowujący informacje o wykonywanych operacjach
      d. Zadeklaruj callback onCompletedCallback (typu SendOrPostCallback)
      e. Zdefiniuj metodę CalculateCompleted, która, w razie, gdy zostało w jakiś sposób zdefiniowane zdarzenie MatMulCompleted, wywołuje to zdarzenie
      f. W konstruktorze przypisz operację CalculateCompleted do callbacku onCompletedCallback
   C) Na podstawie dokumentacji MSDN uzupełnij podany wzorzec EAP służący do mnożenia macierzy o definicje poszczególnych funkcji (większość z metod jest po prostu zgodna ze wzorcem, kreatywnością musisz się wykazać tylko w metodach MatMul oraz getVal):
      a. public void CalculateCompleted(object state)
      b. void Completion(int size, double[] mat, Exception ex, bool cancelled, AsyncOperation ao)
      c. bool TaskCancelled(object taskID)
      d. void CalculateWorker(double[]mat1, double[]mat2, int size, AsyncOperation asyncOp)
      e. double getVal(double[] mat, int column, int row, int size)
      f. double[] MatMul (double[] mat1, double[] mat2, int size)
      g. public virtual void MatMulAsync(double[] mat1, double[] mat2, int size, object taskId)
      h. public void CancelAsync(object taskId)
   D)
10. Posługując się narzędziem opracowanym w zadaniu 9 utwórz klienta oprogramowania mnożącego macierze. Zaimplementuj zdarzenie Completed, zdarzenie jest odpowiedzialne za wyświetlenie wyniku na ekranie komputera.

Zadanie domowe : wykorzystaj to samo oprogramowanie aby zrealizować strategię dziel i rządź w celu zrównoleglenia operacji mnożenia. Czy jest to optymalna strategia? Co należałoby zrobić, aby zoptymalizować program?

11. Wykorzystaj obiekt typu BackgroundWorker w celu uruchomienia procesu obsługującego TcpListener w osobnym wątku. Zwróć uwagę na budowę obiektu BackgroundWorker – jest on typowym przypadkiem wykorzystania wzorca EAP. Zaimplementuj wszystkie zdarzenia, na których implementacje pozwala BackgroundWorker.
12. Dokładnie przeanalizuj przedstawiony kod. Zwróć uwagę na asynchroniczne wyrażenie lambda – jak długo to wyrażenie jest „wywoływane”?. Dodatkowo, przemyśl do czego służy metoda ContinueWith.

```
static async Task serverTask(){

            TcpListener server = new TcpListener(IPAddress.Any, 2048);

            server.Start();

            while(true){

                TcpClient client = await server.AcceptTcpClientAsync();

                byte[] buffer = new byte[1024];

                client.GetStream().ReadAsync(buffer, 0, buffer.Length).ContinueWith(

                    async (t) =>

                    { int i =  t.Result;

                        while (true)

                        {   client.GetStream().WriteAsync(buffer, 0, i);

                            i = await client.GetStream().ReadAsync(buffer, 0, buffer.Length);

                        }});}}
```

13. Zaimplementuj clientTask zgodnie ze wzorcem TAP. Posłuż się w implementacji kodem źródłowym z zadania 12. Posługując się metodą Task.WaitAll uruchom jedno zadanie serwera i kilka zadań klienta.