**Lab4** - synchronizacja dwóch odzielnych programow za pomocą Mutex'u. Mutex jest to prymitywna synchornizacja, zapewniajaca wyłączny dostęp udostępnionego zasobu do tylko jednego wątku.

Po uruchomieniu programu głównego definiuje mutex o określonej nazwie, podając nazwę flagi jako fałsz. Dzięki temu program może uzyskać muteks, jeśli już jest stworzony. Następnie, jeśli nie uzyskano muteksa, program po prostu wyświetla „Dziala” i czeka na wcisniecie dowolnego klawisza, aby zwolnić muteks i wyjść. Jeśli uruchomimy drugą kopię programu, będzie czekał przez 5 sekund, próbując zdobyć muteks. Jeśli naciśniemy dowolny klawisz w pierwszej kopii programu, rozpocznie się wykonywanie drugiego. Jednak, jeśli będziemy czekać przez 5 sekund, druga kopia programu nie zdobędzie muteksu.

**Lab5** - SemaphoreSlim jest lekką wersją Semafora. Ogranicza liczbę wątków, które mogą jednocześnie uzyskać dostęp do zasobu.

Po uruchomieniu programu głównego, tworzy się instancję SemaphoreSlim, podając numer współbieżnych wątków dozwolonych w jego konstruktorze. Następnie zaczyna sześć wątków z różnymi nazwami i czasami rozpoczęcia. Każdy wątek próbuje uzyskać dostęp do bazy danych, ale ograniczamy liczbę równoczesny dostęp do bazy danych przez cztery wątki za pomocą semafora. Kiedy cztery wątki uzyskują dostęp do bazy danych, pozostałe dwa wątki czekają aż jeden z poprzednich wątkiów skończy swoją pracę i zasygnalizuje, wywołując metodę \_emaphore.Release.

**Lab6** - przykład wysyłania powiadomień z jednego wątku do drugiego przy pomocy konstrukcji AutoResetEvent. AutoResetEvent powiadamia o oczekującym wątku, że zdarzenie miało miejsce.

Po uruchomieniu programu głównego definiuje dwie instancje AutoResetEvent. Jeden z nich jest

do sygnalizowania z drugiego wątku do głównego wątku, a drugi do sygnalizacji od głównego wątku do drugiego wątku. Ustawimy false w AutoResetEvent, określając początkowy stan obu instancji jako unsignaled. To znaczy że każdy wątek wywołujący metodę WaitOne jednego z tych obiektów zostanie zablokowany do czasu wywołania metody Set. Jeśli zainicjalizujemy stan zdarzenia na wartość true, stanie się on sygnalizowany i pierwszy watek wywołujący WaitOne nastąpi natychmiast. Musimy ponownie wywołać metodę Set, aby pozwolić drugiemu wątkowi kontynuować. Następnie tworzymy drugi wątek, który wykona pierwszą operację przez 10 sekund i poczeka na sygnał z drugiego wątku. Sygnał oznacza, że ​​pierwsza operacja została zakończona. Teraz drugi wątek czeka na sygnał z głównego wątku. Wykonujemy dodatkową pracę nad głównym wątkiem i wysłamy sygnał, wywołując metodę \_mainEvent.Set. Wtedy czekamy na kolejny sygnał z drugiego wątku.

**Lab7** – ManualResetEventSlim – elastyczniejsza sygnalizacja między wątkami.

Po uruchomieniu programu głównego, tworzy się instancja ManualResetEventSlim. Następnie rozpoczynamy trzy wątki, które będą czekać na zdarzenie, aby zasygnalizować kontynuację wykonania.

**Lab8 –** CountdownEvent - sygnalizuje by czekać aż pewna liczba operacji zostanie zakończona.

Po uruchomieniu głównego programu tworzymy nową instancję CountdownEvent, określając, że chcemy, aby sygnalizował, kiedy dwie operacje zakończą się w konstruktorze. Następnie rozpoczynamy dwa wątki które zasygnalizuja kiedy skończą. Jak tylko drugi wątek zostanie ukończony, główny

wątek powraca z oczekiwania na CountdownEvent i kontynuuje dalej.

**Lab9 –** Barrier - pomaga zorganizować kilka wątków do spotkania w pewnym momencie, zapewniając wywołanie zwrotne, które będzie wykonywane za każdym razem, gdy wątki będą wywoływać

Metodę SignalAndWait.

**Lab10** – program przedstawia sposób oczekiwania na wątek bez angażowania kernel-mode oraz SpinWait - hybrydową konstrukcję synchronizacyjną, do która została zaprojektowana by czekać przez jakiś czas w trybie użytkownika, a następnie przejść do kernel-mode, aby zaoszczędzić czas procesora.