```
void RemoveThread(wxThread* th)
{
   wxCriticalSectionLocker lock(cs);
   thread_pool.Remove(th);
   if (semaphore) semaphore->Post();
}
```

Obiekt klasy ***wxCriticalSectionLocker*** działa analogicznie do *wxMutexLocker* i wchodzi do sekcji krytycznej na czas do chwili jego zniszczenia (czyli w tym przypadku do końca pliku funkcji).

Warunek wxCondition

Sekcje krytyczne oraz *muteksy* mogą być używane z warunkami ***wxCondition***, które służą do uwalniania wątków oczekujących z chwilą, gdy warunek staje się prawdziwy, a jego stan zostanie zasygnalizowany przez aplikację. Prosty i niezwykle przejrzysty przykład korzystania z warunku *wxCondition* znajdziesz w dokumentacji tej klasy.

21.3. Wątki i ich zdarzenia

Komunikacja między wątkami aplikacji to zagadnienie dość problematyczne, zwłaszcza, gdy w grę wchodzi kontrola stanu wątku. Początkujący programiści wxWidgets często popełniają błąd polegający na nadgorliwym obciążaniu aplikacji mechanizmami sprawdzającymi wątki, opartymi na wywoływaniu metod blokujących zasoby. Tymczasem cała interakcja między wątkami może zostać obsłużona w bardzo elegancki sposób, to znaczy za pomocą zdarzeń.

Odpowiednio przygotowanym zestawem zdarzeń możesz informować aplikację o wszystkim, co dotyczy pracy wątku. Od jego utworzenia, przez aktualizację wartości, nad którymi pracuje, po poinformowanie aplikacji o zakończeniu pracy, a nawet zamknięciu danego wątku. Oprócz tego zdarzenia wątków są doskonałym rozwiązaniem wszędzie tam, gdzie jest potrzebna aktualizacja elementów GUI, co w całości można przenieść na wątek główny aplikacji i uniknąć niepotrzebnych konfliktów.

Definiowanie i obsługa zdarzeń wątków to zadania niezwykle łatwe do realizacji. Najlepiej przekonać się o tym, analizując prosty przykład. Niech będzie to wątek, który będzie wykonywać cyklicznie jakąś operację aż do osiągnięcia pewnego limitu, po czym zostanie on zamknięty. Zarówno informację o zmianach wartości przetwarzanych przez wątek, jak i informację o jego zamknięciu przekażemy w formie zdarzeń do głównego okna aplikacji, które odpowiednio je przetworzy i zaktualizuje GUI, na przykład przez dodanie odpowiednich informacji w logu.

Na początek, przed wykonaniem jakichkolwiek innych czynności, trzeba zdeklarować rodzaje zdarzeń wątków:

```
wxDECLARE_EVENT(wxEVT_MYTHREAD_UPDATED, wxThreadEvent); // Aktualizacja stanu
wxDECLARE_EVENT(wxEVT_MYTHREAD_CLOSED, wxThreadEvent); // Zakończenie wątku
```

Zwróć uwagę, że korzystamy tutaj z klasy ***wxThreadEvent***, która została wyposażona w funkcje ułatwiające komunikację między wątkami. Ma ona kilka przydatnych metod, dzięki którym możesz ustawiać i pobierać typowe dane z użyciem jej obiektów – ich listę sprawdź w dokumentacji. Gdyby zakres danych możliwych do przetworzenia okazał się zbyt mały, możesz wyprowadzić klasę własnego zdarzenia z *wxThreadEvent* i wzbogacić ją o dodatkowe pola, zgodne z Twoimi oczekiwaniami.

Gdy mamy już deklaracje zdarzeń, możemy przystąpić do napisania klasy wątku realizującego – zgodnie z przyjętym założeniem – proste odliczanie:

```
class MyCountThread: public wxThread
{
   public:
      MyCountThread(MyFrame *hdlr)
         : wxThread(wxTHREAD_DETACHED), handler(hdlr) {}
```

```
      ~MyCountThread() {}

   protected:
      virtual ExitCode Entry() wxOVERRIDE
      {
         // Symulacja odliczania
         for (int i = 10; i != 0; i--)
         {
            wxThreadEvent* evt = new wxThreadEvent(wxEVT_MYTHREAD_UPDATED);
            evt->SetExtraLong(i);
            evt->SetId(this->GetId());
            wxQueueEvent(handler, evt);
            Sleep(1000);
         }

         return (wxThread::ExitCode)0;
      }

      virtual void OnExit() wxOVERRIDE
      {
         wxGetApp().RemoveThread(this);

         wxThreadEvent* evt = new wxThreadEvent(wxEVT_MYTHREAD_CLOSED);
         evt->SetId(this->GetId());
         wxQueueEvent(handler, evt);
      }

      MyFrame *handler;
};
```

Właśnie tutaj objawił się prawdziwy sens przechowywania wskaźnika do okna głównego w obiekcie wątku. Dzięki temu, że mamy do niego dostęp, możliwe jest sprawne złożenie obiektów zdarzeń, przypisanie do nich danych i w końcu umieszczenie w kolejce zdarzeń za pomocą funkcji **void *wxQueueEvent(wxEvtHandler* dest, wxEvent* event)***. Można również zrezygnować z przechowywania wskaźnika okna głównego i każdorazowo uzyskiwać do niego dostęp za pomocą funkcji *wxApp::GetTopWindow()*:

```
wxQueueEvent(wxGetApp().GetTopWindow(), evt);
```

Gdy mamy tak napisaną klasę wątku, za jej pomocą możemy już tworzyć nowe wątki aplikacji, jednak klasa okna głównego nie będzie jeszcze nic wiedzieć o zdarzeniach, jakie są przez wątek wysyłane. Należy zatem wyposażyć ją w odpowiedni mechanizm obsługi zdarzeń pochodzących z wątków.

Konieczne jest najpierw zdefiniowanie wcześniej zadeklarowanych zdarzeń:

```
wxDEFINE_EVENT(wxEVT_MYTHREAD_UPDATED, wxThreadEvent);
wxDEFINE_EVENT(wxEVT_MYTHREAD_CLOSED, wxThreadEvent);
```

Kolejnym krokiem jest zadeklarowanie odpowiednich metod klasy okna głównego (czyli w większości przypadków przez nas rozpatrywanych – naszej własnej klasy *MyFrame*):

```
void OnThreadUpdate(wxThreadEvent& event);
void OnThreadClose(wxThreadEvent& event);
```

Następnie trzeba połączyć zdarzenia z odpowiednimi funkcjami, co można wykonać, umieszczając właściwe wpisy w tablicy zdarzeń lub dołączając ich obsługę dynamicznie w konstruktorze *MyFrame*:

```
Bind(wxEVT_MYTHREAD_UPDATED, &MyFrame::OnThreadUpdate, this, wxID_ANY);
Bind(wxEVT_MYTHREAD_CLOSED, &MyFrame::OnThreadClose, this, wxID_ANY);
```

Na koniec pozostaje napisanie metod obsługujących zdarzenia: