Dokumentacija WCF Raspodeljene Aplikacije Radnika

# 1. Pregled sistema

Ova aplikacija simulira distribuirani skup od 10 klijenata koji komuniciraju sa centralnim WCF servisom. Najviše 5 radnika istovremeno radi (Active), ostali su u režimu pripravnosti (Standby). Klijenti šalju heartbeat poruke svakih 5 s. Ukoliko servis ne primi heartbeat u roku od 15 s, radnik se smatra mrtvim (Dead), a jedan iz Standby liste preuzima taj posao.

Glavne komponente:

* **ClientApp** – klijentska aplikacija sa radnicima
* **HostApp** – WCF host koji pokreće servis
* **WcfService** – implementacija servisnog interfejsa i koordinacije

# 2. Struktura projekta

Solution 'Wcf'

├── ClientApp

│ ├── ConsoleLogger.cs

│ ├── ServiceProxy.cs

│ ├── TaskWorkerFactory.cs

│ ├── TaskWorker.cs

│ └── Program.cs

├── HostApp

│ └── Program.cs

└── WcfService

├── IService.cs

└── WorkerCoordinator.cs

# 3. Opis .cs fajlova

## ClientApp

### *ConsoleLogger.cs*

* Jednostavan logger za konzolu s podrškom za boju teksta.
* Metoda Log(message) ispisuje poruku belom bojom.
* Metoda Log(message, color) ispisuje u zadatoj boji.

### *ServiceProxy.cs*

* Definiše DataContract tipove: MessageStatus, MessageError, Message, WorkerState.
* Sadrži WCF duplex proxy factory (ServiceProxyFactory) koja otvara kanal prema servisu na http://localhost:8080/Service/.

### *TaskWorkerFactory.cs*

* Metod Create(int workerId) pravi instancu TaskWorker, poziva Register(), vraća radnika ako je uspešno registrovan.

### *TaskWorker.cs*

* Implementira ICallback interfejs za prijem callback-a od servisa.
* **Stanja radnika**:
  + Standby – ne radi, samo čeka callback.
  + Active – pokreće DoWork() petlju i timer za heartbeat.
  + Dead – prekida rad i heartbeat.
* **Heartbeat timer** šalje SendHeartbeat(id) svakih 5 s.
* **DoWork()** petlja: ako je aktivan i nalazi se u dobijenoj listi aktivnih, ispisuje  
  "[Worker X] WORKING... <vreme>" svake 5 s.
* Na prvo aktiviranje, planira lokalni *control-task* koji simulira krash ili pauzu heartbeat-a (10–30 s kasnije).

### *Program.cs*

* Generiše nasumični redosled ID-jeva 0–9.
* Registruje sve radnike (TaskWorkerFactory.Create).
* Ispisuje plavom bojom početne poruke *“[Main] Sve registracije završene…”* i čeka da svi pređu u Dead, pa završava.

## HostApp

### *Program.cs*

* Kreira ServiceHost nad tipom WorkerCoordinator na adresi http://localhost:8080/Service/.
* Dodaje endpoint s WSDualHttpBinding.
* Pokreće host i čeka pritisak tastera.

## WcfService

### *IService.cs & ICallback*

* **IService** (ServiceContract) definiše RPC metode:
  + Message Register(int registrationWorkerId)
  + void SendHeartbeat(int workerId)
* **ICallback** (CallbackContract) definiše:
  + void ChangeWorkerState(WorkerState newState)
  + void ShutdownWorker()
  + void UpdateActiveWorkers(int[] activeIds)

### *WorkerCoordinator.cs*

* **Konstruktor**: pokreće timer (CheckInterval = 1 s) za proveru heartbeata.
* **Register(int id)**:
  1. Zaključava sinhronizaciju.
  2. Dodaje WorkerInfo u \_workers.
  3. Ako je manje od 5 aktivnih, postavlja radnika na Active (šalje callback).
  4. Ispisuje log i vraća Status = Ok.
* **SendHeartbeat(int id)**: ažurira LastHeartbeat ili, ako je mrtav, šalje ShutdownWorker().
* **MonitorWorkers** (svake 1 s):
  1. Prolazi sve radnike, proverava kašnjenje > 15 s.
  2. Ako je prešao, markira ga Dead i ispisuje log.
  3. Birа nasumični radnik iz Standby liste i aktivira ga (Active).
  4. Ako je bilo promena, poziva BroadcastActiveList().
* **SetState** i **BroadcastActiveList** šalju callback-ove ka klijentima.

# 4. Tok izvršavanja

1. **Host pokreće servise**
   * Otvara ServiceHost, vezuje URL, čeka konekcije klijenata.
2. **Klijenti se pokreću**
   * Program.Main() nasumično registruje 10 radnika.
   * Servis u Register() dodeljuje prvim 5 aktivan status, ostali Standby.
3. **Heartbeat + radna petlja**
   * Klijenti koji dobiju ChangeWorkerState(Active):
     + Pozivaju StartHeartbeat() → timer šalje SendHeartbeat na servis svake 5 s.
     + Pokreću DoWork() petlju: ako su aktivni i u dobijenoj listi, ispisuju WORKING...
   * Ako stanju postane Standby ili Dead, zaustavljaju heartbeat i rad.
4. **Servisova provera**
   * Svakog 1 s: MonitorWorkers markira neaktivne radnike mrtvim (Dead).
   * Iz liste Standby bira novog radnika, aktivira ga.
   * Šalje svima aktuelnu listu aktivnih ID-jeva preko UpdateActiveWorkers.
5. **Simulacija krah/pauze**
   * Svaki klijent na svom kontrolnom tasku nasumično posle 10–30 s:
     + Ili poziva StopWork() (sim. krah), što prekida petlju i heartbeat.
     + Ili pauzira heartbeat na 20 s, pa ga ponovo startuje.
6. **Završetak**
   * Klijentski Main čeka da svi radnici pređu u Dead.
   * Kada su svi mrtvi, ispisuje završnu poruku.

# 5. Kako sve radi zajedno

1. **Registracija**
   * Klijent → Register(id) → Servis unosi u \_workers i eventualno šalje ChangeWorkerState(Active).
2. **Duplex kanal**
   * ServiceProxyFactory kreira WCF duplex kanal uz ICallback implementaciju u TaskWorker.
3. **Sinkronizacija stanja**
   * Servis održava jedinstveni kontekst (Single instance), sinhronizuje pristup kolekciji radnika.
4. **Heartbeat mehanizam**
   * Klijenti šalju SendHeartbeat svakih 5 s kad su aktivni.
   * Servis detektuje mrtve koji ne šalju > 15 s.
5. **Promocija Standby radnika**
   * Nakon timeout‑a markira mrtvog, bira jednog iz Standby i šalje ChangeWorkerState(Active).
6. **Ažuriranje liste**
   * Servis po svakoj promeni stanja aktivnih radnika poziva UpdateActiveWorkers sa trenutnom listom.
   * Na klijentu samo onaj sa najmanjim ID‑jem ispisuje listu (u zelenoj boji).
7. **Krah i oživljavanje**
   * Klijenti sami “pokrču” sopstvene krahove ili pauze heartbeat-a (StopWork ili StopHeartbeat).
8. **Finalna sinhronizacija**
   * Klijentski Program sinhrono čeka kraj svih radnika, zatim ispisuje završetak.

# 6. Zaključak

Ova aplikacija demonstrira:

* Postavljanje i korišćenje WCF **duplex** servisa
* Upravljanje radnicima kroz **state machine** (Standby → Active → Dead)
* Mehanizam heartbeat/timeout za **otkrivanje krahova**
* Promociju standby radnika radi **load balancinga**
* Sinkronizaciju stanja i **callback**–ove kroz WCF
* Jednostavnu **logiku simulacije** krahova i pauza na klijentskoj strani

Sve u svemu, sistem pokazuje kako se može na jednostavan način implementirati tolerancija na greške i dinamičko preraspoređivanje poslova u malom distribuiranom okruženju.