J 9. Übung

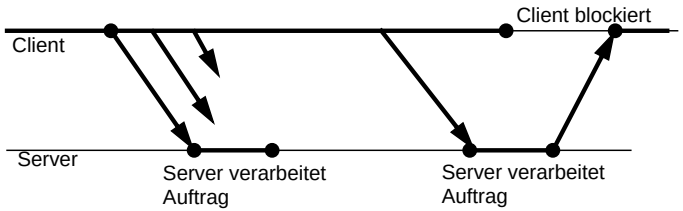
J.1 Überblick

- Asynchroner RPC
  - Ausprägung ohne/mit Rückgabewert
- Asynchroner RPC am Beispiel von RMI und Java-1.5

- Literatur:
  - B. Liskov: "Promises: Linguistic Support for Efficient Asynchronous Procedure Calls in Distributed Systems", SIGPLAN, 1988
  - A.L. Ananda, B.H. Tay, E.K. Koh: "A Survey of Asynchronous Remote Procedure Calls", SIGOPS, 1992

J.2 Asychroner RPC

- Asychroner RPC
  - Fernaufruf kehrt sofort nach Übergabe der Parameter zurück
- Es gibt zwei Ausprägungen
  - ohne Rückgabewert / mit Rückgabewert
- Vorteile
  - Parallelverarbeitung von Client und Server (effizientere Implementierung)
  - Geeignet für hohen Durchsatz wenn kein Rückgabewert gefordert



J.3 Asynchrone RPCs mit Rückgabewert

- Problem: Zuordnung des Rückgabewert
- Verschiedene Implementierungskonzepte
  - Es wird ein spezieller Stellvertreter (handle) bereit gestellt

```
Int_Handle handle = multiply(4,6);
if(handle.poll()){
    y = handle.get();
}
y = handle.get();
```

  - Annahme des Ergebnis über einen speziellen Anweisungsblock (vgl. Interrupt-Routine)
  - Spracheinbettung durch "Future-Variablen"

```
FUTURE int future;
int num = 4711;
future = multiply(4,6);
print(num+future);
```


J.4 Asychrone Methodenaufrufe in Java 1.5

```
import java.rmi.*;
import java.util.concurrent.*;

class Client implements Callable {
    Service s;

    public static void main (String[] p) throws Exception {
       new Client();
    }

    public Client() throws Exception {
       s = (Service)Naming.lookup("something");
       ExecutorService es =
                     Executors.newSingleThreadExecutor();
       Future f = es.submit(this);
       System.out.println("result:"+f.get());
    }

    public Integer call() throws Exception{
       return  s.multiply(5,5);
}   }
```

J.5 Asynchrones multiply

- Erhalt der Schnittstellendefinition

```
interface MultiplyServer {
    int multiply(in int val1, in int val2);
};
```

- Erweiterung der Stub-Schnittstelle durch Methoden für asynchrone Aufrufe

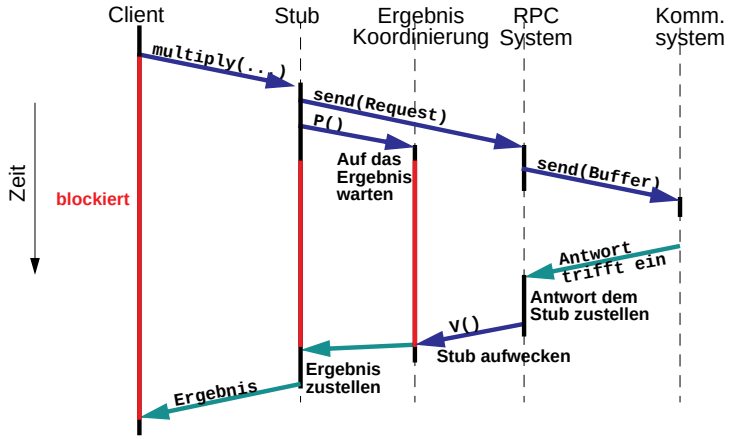
```
struct MultiplyServerStub {
    int16_t multiply(int16_t val1, int16_t val2);
    Future<int16_t>async_multiply(int16_t val1, int16_t val2);
};
```

- Abfrage des Ergebnis mit Hilfe eines Platzhalterobjektes

```
template <class t>
struct Future {
    t get();       /* wait for result (blocking!)*/
    bool poll();   /* test whether the result is available */
};
```

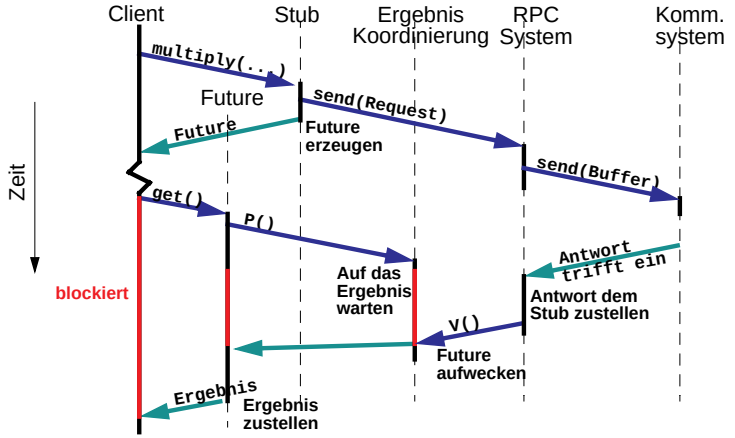

1 Synchroner Aufruf

- Ausgangssituation (Aufgabe 6)



2 Asynchroner Aufruf

- Entkopplung durch Ergebnisplatzhalter (Future-Objekt)



2 Asynchroner Aufruf

- Alternativer Ablauf