Příklad aplikace Klient/Server s Boss/Worker modelem (informativní)

Jan Faigl

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

A0B36PR2 – Programování 2

Příklad aplikace Klient/Server s Boss/Worker modelem

Příklad vícevláknové aplikace klient/server

- K serveru se připojí klient, který obdrží parametry pro generování grafu.
- Klient vrací serveru hash hodnotu řešení nejkratších cest z uzlu 0.
- Během vlastního řešení musí klient komunikovat se serverem definovaným způsobem a udržovat spojení.
- Po odeslání výsledku server posílá potvrzení správného / špatného řešení a ukončuje spojení.
- Není-li spojení správně udržované (periodické), posíláním `alive` zpráv, ukončuje server spojení.

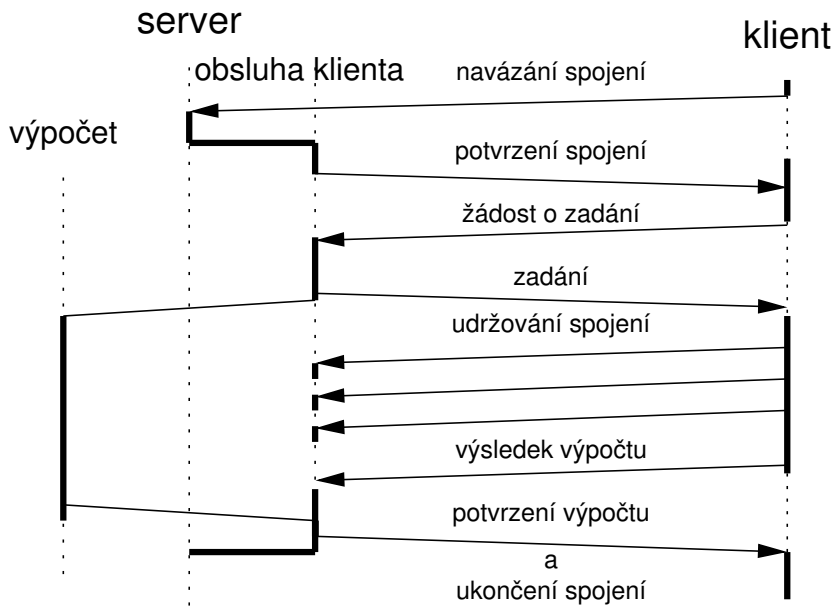
Popis činnosti serveru

Po přijetí klienta jsou provedeny následující operace.

1. Poslání parametrů spojení klientu.
2. Čekání na žádost o parametry grafu.
3. Poslání parametrů grafu.
4. Generování grafu, hledání řešení a výpočet hash.
5. Příjem klientského řešení.
6. Porovnání vlastního a klientského řešení.
7. Poslání výsledku porovnání klientu.
8. Ukončení spojení.

Během bodu 4 a 5 je nutné hlídat, zda klient udržuje spojení.
Klient může poslat řešení dříve než jej nalezne server.

Průběh činnosti



Základní architektura

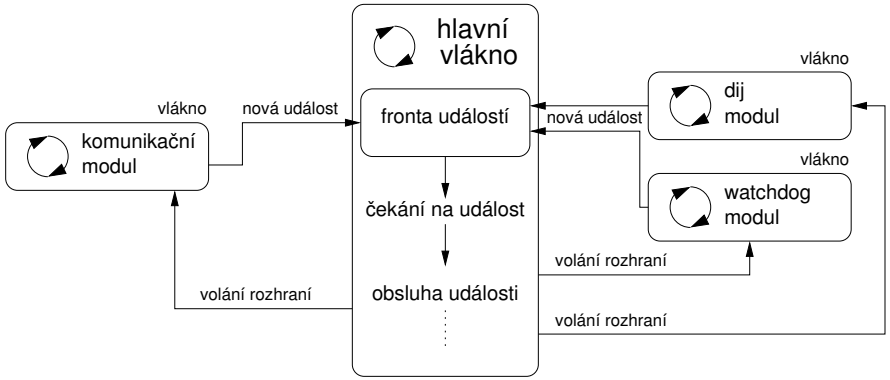
Pro každého připojeného klienta vytvoříme `clientHandler`, který obsahuje

- komunikační modul `comm`,
- watchdog modul `watchdog`, který hlídá udržování komunikace s klientem,
- modul pro řešení zadaní `dij`, který volá program `tdijkstra`.

Synchronizační mechanismy:

- fronta událostí,
- exkluzivní přístup ke sdíleným proměnným.

Propojení modulů



Fronta událostí

```
1  class EventQueue {
2     Queue queue;
3     Object cond;
4
5     EventQueue() {
6        queue = new Queue();
7        cond = new Object();
8     }
9
10    synchronized void addEvent(int event) {
11       queue.push(event);
12       cond.notify();
13    }
14
15    synchronized int getEvent() {
16       if (queue.size() == 0) {
17          cond.wait();
18       }
19       return queue.pop();
20    }
21 }
```

Modul Dij

```
1  class Dij extends Thread {
2     EventQueue queue;
3     int number; int seed; int from;
4     int hash;
5     Dij(EventQueue iQueue) {
6        queue = iQueue;
7     }
8     public void run() {
9        hash = callTdijkstra(number, seed, from);
10       queue.addEvent(SOLUTION_FOUND);
11    }
12    synchronized int getHash() {
13       return hash;
14    }
15    synchronized void solve(int n, int s, int f) {
16       if (!isAlive()) {
17          number = n; seed = s; from = f;
18          start();
19    } }
20    synchronized void shutdown() {
21       join();
22    }
23 }
```


Modul Watchdog 1/2

```
1  public class Watchdog extends Thread {
2     EventQueue queue;
3     int period;
4     int ping;
5     Watchdog(EventQueue iQueue, int iPeriod) {
6        queue = iQueue;
7        period = iPeriod;
8     }
9
10
11    synchronized void alive() {
12       ping++;
13    }
```


Modul Watchdog 2/2

```
1     public void run() {
2        boolean q = false;
3        while (!q) {
4           sleep(period);
5           synchronized(this) {
6              q = quit;
7              if (ping == 0) {
8                 eventqueue.add(WATCHDOG_TIMEOUT);
9              }
10             ping = 0;
11          }
12       }
13    }
14
15    void shutdown() {
16       synchronized(this) {
17          quit = true;
18       }
19       join();
20    }
21 }
```


Modul Comm 1/2

```
1  class Comm extends Thread {
2    NetConnection conn;
3    EventQueue queue;
4    Comm(EventQueue iQueue, NetConnection iConn) {
5      queue = iQueue;
6      conn = iCon;
7    }
8    synchronized void shutdown() {
9      if (!isAlive()) {
10       conn.stop(); //stop network (blocking receiving)
11       join();
12   } }
13   private synchronized int parse(CommMessage msg) {
14     ...
15     return msgType;
16   }
17   int synchronized getHash() {
18     //client hash from network connection
19     return hash;
20   }
```

Modul Comm 2/2

```
1  public void run() {
2     CommMessage msg;
3     boolean quit = false;
4     //blocking receive
5     while(!quit && (msg = conn.receive() != null)) {
6        switch(parse(msg)) {
7           case MSG_GETPARAM:
8              queue.addEvent(CLIENT_GETPARAM);
9           case MSG_ALIVE:
10             queue.addEvent(CLIENT_ALIVE);
11             break;
12          case MSG_SOLUTION:
13             queue.addEvent(CLIENT_SOLUTION);
14          default:
15             //unknown message
16             quit = true;
17             break;
18       }
19    }
20    //inform main thread
21    queue.addEvent(COMM_TERMINATED);
22 }
```


Hlavní vlákno 1/3

ClientHandler je vytvořen po navázání spojení.

```
1  public class ClientHandler extends Thread {
2     EventQueue queue;
3     Watchdog watchdog;
4     Dij dij;
5     Comm com;
6     ClientHandler(NetConnection conn) {
7        queue = new EventQueue();
8        watchdog = new Watchdog(queue, PERIOD);
9        dij = new Dij(queue);
10       comm = new Comm(queue, conn);
11       start();
12    }
13
14    void shutdown() {
15       watchdog.shutdown();
16       comm.shutdown();
17       dij.shutdown();
18       queue.addEvent(STOP);
19       join();
20    }
```

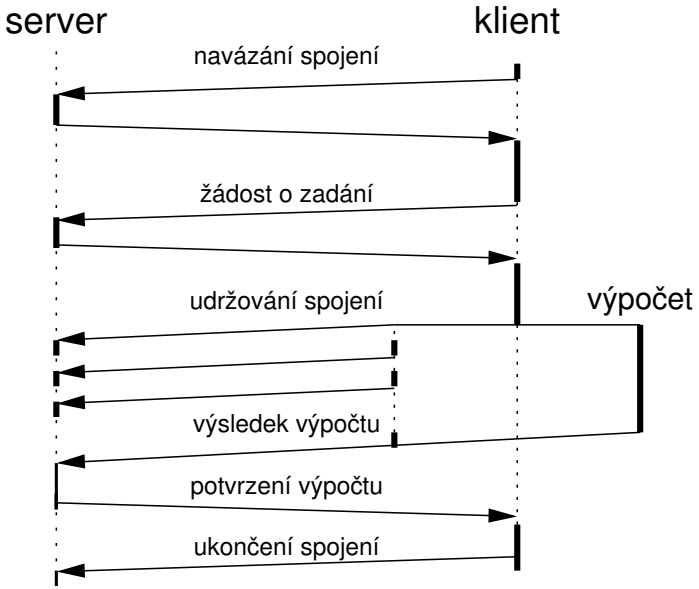

Hlavní vlákno 2/3

```
1  public void run() {
2    int ev;
3    boolean clientSolution; boolean mySolution;
4    Param par = new Param();
5    while (ev = queue.getEvent() != STOP) {
6      switch(ev) {
7        case CLIENT_GETPARAM:
8          par = generateParam();
9          dij.solve(par.number, par.seed, par.from);
10         comm.sendParam(param);
11         watchdog.start();
12         break;
13       case CLIENT_ALIVE:
14         watchdog.alive();
15         break;
16       case CLIENT_SOLUTION:
17         clientSolution = true;
18         queue.addEvent(COMPARE);
19         break;
20       case COMM_TERMINATED:
21         queue.addEvent(STOP);
22         break;
```

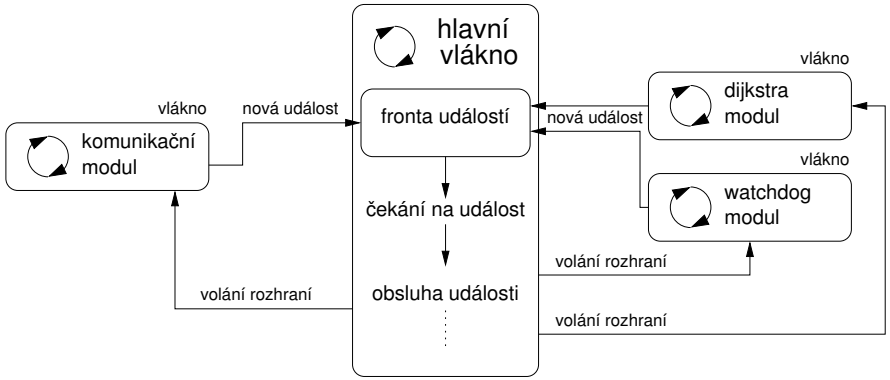

Hlavní vlákno 3/3

```
1        case SOLUTION_FOUND:
2          mySolution = true;
3          queue.addEvent(COMPARE);
4          break;
5        case COMPARE:
6          if (clientSolution && mySolution) {
7            if (dij.getHash() == comm.getHash()) {
8              comm.sendDisconnect("Solution OK");
9            } else {
10             comm.sendDisconnect("Solution WRONG");
11           }
12           queue.addEvent(STOP);
13         }
14         break;
15       case WATCHDOG_TIMEOUT:
16         comm.sendDisconnect("Alive timeout");
17         queue.addEvent(STOP);
18         break;
19       default://unknown event
20         queue.addEvent(STOP);
21         break;
22     }
23 } } }
```

Jak implementovat klientskou část?

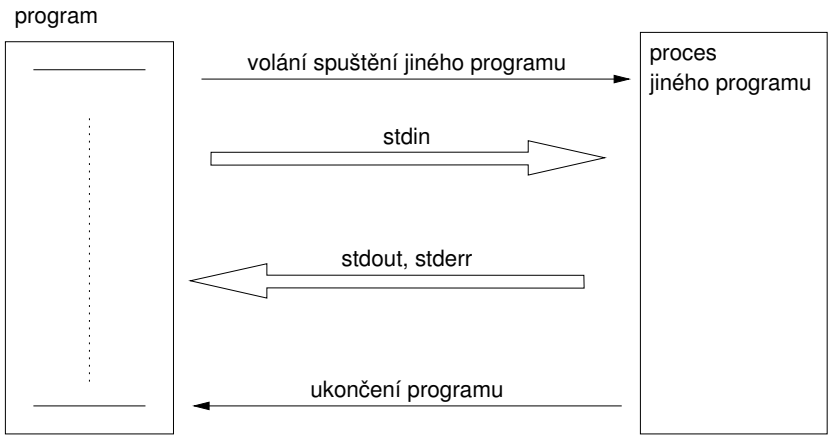


Propojení modulů



Příklad Spuštění jiného programu z procesu

Příklad spuštění jiného programu z Javy



- Identická architektura aplikace (Boss/Worker)
- Hlavní logika programu je v obsluze hlavní smyčky zpráv

Příklad - spuštění jiného programu z procesu

Příklad - spuštění programu `tdijkstra` z Javy

```
1  private boolean callDijkstra() {
2    boolean ret = false;
3    try {
4      String cmd = "./tdijkstra -h -n " + size + " -s " + seed;
5      Process child = Runtime.getRuntime().exec(cmd);
6      child.waitFor();
7      if (child.exitValue() == 0) {
8        BufferedReader out = new BufferedReader(new
       InputStreamReader(child.getInputStream()));
9        hash = Integer.parseInt(out.readLine());
10       ret = true;
11     } else {
12       System.out.println("Error in dijsktra");
13     }
14   } catch (Exception e) {
15     System.out.println("Error: Exception : " + e.getMessage());
16   }
17   return ret;
18 }
```

Co se stane při nedostatečné vyrovnávací paměti pro stdout.