Chapter 5: TP Monitors I

More Details On:

• what is a TP Monitor
• how you interact with it

Overview

• TP Monitor functions
• types of transaction execution
• mapping requests -> processes
• how a TP Monitor works
• TRPC
• servers
• server calls: savepoints, chain work, leavetrans, ...
Computing Styles

- batch
- time sharing
- real time
- client/server
- transaction oriented

"In a contest for the least well defined software term, TP Monitors would be a tough contender."
TP Monitor Functions

- manage heterogeneity
- control communication
- terminal management
- presentation services
- context management (e.g. x windows, cursors, current authenticated user)
- start / restart of the system
- scheduling
- server class management
- authentication & authorization
- resource administration
- system operation
- recovery
- definition of new resource managers
- changing process configuration
- TRPC
Summary

TP Monitor Manages

- processes
- communication
- system resources
Types of Transaction Execution

- direct vs. queued
  - is application program aware of queues?
- long (conversational) vs. short (one message)
- local vs. distributed

Mapping Requests -> Processes

<table>
<thead>
<tr>
<th>Client or Terminal</th>
<th>Service or Application Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>want(...)</td>
<td>?</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
OS Solution

[see textbook page 254, Figure 5.7a]

Old Fashioned TP Solution

[see textbook page 256, Figure 5.7b]

Server Processes

[see textbook page 257, Figure 5.7c]

Generalized Solution

[see textbook page 258, Figure 5.7d]
Issues

- number of processes
- number of control blocks (e.g. DB, file, ...)
- process switches & starts
- cache misses (e.g. of control blocks)
- load balancing & process priorities
- complexity of TP Monitor
- isolation of applications

How a TP Monitor works

[see textbook pages 264-265, Figures 5.9 and 5.10]
**RPC Example**

```
... 
... 
call foo(integer i); 
... 
... 
```

**Main**

```
Proc foo(real x) {
    ... 
    ... 
    return; }
```

**Server Process**
After Stub Compilation/Generation

main

\[
... \\
call \text{foo}(\text{integer } \ i); \\
... \\
\]

\[
\begin{align*}
\text{proc } \text{foo}(\text{integer } \ i); \\
& \quad \text{convert } \ i \ \text{to real } \ z; \\
& \quad \text{put } \ z \ \text{in } \ \text{msg buffer}; \\
& \quad \text{put control data in } \ \text{msg buffer}; \\
& \quad \text{send message to process } \ \alpha \ \text{in machine } \ B; \\
& \quad \text{receive reply message (on timeout...)}; \\
& \quad \text{extract real from message}; \\
& \quad \text{convert real to integer}; \\
& \quad \text{put in } \ i; \\
& \quad \text{return}; 
\end{align*}
\]
process \( \alpha \) (server process)

\[
\begin{align*}
&\text{while TRUE do} \\
&\quad \text{receive message;} \\
&\quad \text{check for validity;} \\
&\quad \text{extract real } x, \text{ process id "client";} \\
&\quad \text{call foo(real } x); \\
&\quad \text{prepare message with } x; \\
&\quad \text{send message to client;}
\end{align*}
\]

\[
\begin{align*}
\text{proc foo(real } x); \\
\quad \text{...} \\
\quad \text{...} \\
\quad \text{...} \\
\quad \text{return;}
\end{align*}
\]
**Terms**

*coerce* convert parameters to what is expected

*marshaling* packing

*reverse marshaling* unpacking
Static vs. Dynamic Bind

Repository
(can be a memory table in TP monitor or in stub)

Main 1  Main 2  ...  Main n

single stub
Another Step before Stub Compiler

select emp# from emp where sal>1000

↓ SQL pre compiler

!sqlselect(“fastsql”, parse_tree, ...)

↓ stub compiler

call_stub(parameters, ...)
Final RPC Note

Some RPC facilities primitive:
- no automatic marshaling
- no automatic coercion
How Servers Interact with the TP Monitor

```
do_work

server

RM

TM
```
Ignorant Server

// example of simplest version of a transactional resource
// manager.

Boolean IgnorantServer( rmParams * InParams, rmParams * OutResults) {
  declares

  work on input parameters
  ComplCode = rmCall("HelpMe", ...);
  // checking of return code omitted

  prepare output parameters
  return(TRUE);
}


Simple Server

Boolean
SimpleServer( mParams * inParams, mParams * OutResults) {
    TRID    newTRID; // TRID of transaction started here
    RETCODE CompCode; // rmCall completion code
    NewTRID = BeginWork(...); // start transaction

    do work here
    CompCode = rmCall(“HelpMe”,...); // server invocation
    if(CompCode==BAD) { // return code ok?
        Abort_Work(); // abort transaction
        return FALSE; // deny service
    }
    do more work here
    if(result_ok) { // test for success
        prepare output parameters
        Commit_Work();
        return TRUE; // transaction completed
    } else {
        Abort_Work();
        return FALSE;
    }
}
What if caller started transaction?

```c
...  
...  
Begin_Work  
...  
...
```

```c
...  
...  
if(MyTRID() == NULLTRID)  
Begin_Work();  
...  
...  
```
Cautious Server

Boolean
CautiousServer(...) {
    x = "one"; y="red";
    put y in trusted_data;
    newTRID = Begin_Work(..., trusted_data, FALSE);
    do first half: code = rmCall("HelpMe", ...);
        x="two"; y="blue";
        ...
    put y in trusted_data;
    st = Save_Work(FALSE, trusted_data);
    do second half: code = rmCall("helpMe", ...);
        x="three"; y="green";
        ...
    if "something went wrong in second half" {
        st_goal = st;
        st = Rollback_Work( st_goal);
        assert(st==st_goal);
        trusted_data = read_context( st_goal);
        restore y from trusted data;
    }
    print(x,y,dbvalue, ...);
}
if "result is ok" then {
    Commit_Work();
    return TRUE;
} else {
    Abort_Work();
    print(x,y,dbvalue,...);
    return FALSE;
}
Calls to TM

newTrid = Begin_Work(...)

Commit_Work(...)

Abort_Work(...)

savepoint = Save_Work(persistent?, context)

savepoint = Rollback_Work(savepoint)

context = Read_Context(savepoint)

Trid = Chain_Work(persistent?, context)
"Collective" Savepoints

do work

server

... y ...

① rmCall

HelpMe

db

② SaveWork

④ return from SaveWork

③ rmSavepoint

TM
Pulling the Rug under you

⇒ only root invocation hierarchy can call `SaveWork` and `RollbackWork`
Leaving & Resuming Transactions

... 
ClientTrid = Leave_Transaction(...);
MyOwnTrid = Begin_Work(...);
...
Commit_Work(...);
Resume_Transaction(ClientTrid);
...

Notes

• example: server may want to charge for service regardless of outcome of initial transaction
• resume may fail if transaction aborted
• how does server vote on 2PC?
Summary

• What is a TP Monitor

• Server Processes
  - how they are called
  - how they work with TM & RMs