Chapter 6: TP Monitors II

Overview

- Calls to TP Monitor and RMs
- Data Structures
- TRPC Guided Tour
- Queues
- Other TP Monitor Functions
Some New Terms

TP Services (TRAPS)

TP Operating System (TPOS)

Basic Operating System (BOS)

Calling a Component

- Single Entry
- Service Entry Plus Callbacks
- [see textbook page 300, Figure 6.3]
**Example:** Startup of a RM

```
rm_Install( RMName, Callbacks, Code, ...)
```

Diagram:
- **TP Monitor**
- **TM**
- **RM**
- **Identify**
- **Activate**
- **Startup**
After a Failure

- TP Monitor has to activate TM, Log Manager, RMs, etc.
- TP Monitor started by BOS
**RM Calls** (provided by RM)

\[
\begin{align*}
\text{rmCall}(\text{RMName}, \text{BoundTo}, \text{Params}, ...) \\
\text{rm_Prepare()} \\
\text{rm_Commit()} \\
\text{rm_Abort()} \\
\text{rm_Savepoint()} \\
\text{rm_Rollback_Savempoint()} \\
\text{rm_StartUp()} \\
\text{rm_ShutDown()} \\
\text{rm_Redо()} \\
\text{rm_Undo()} \\
\text{tm_StartUp()} \Rightarrow \text{note!} \\
\text{rm_CheckPoint()}
\end{align*}
\]
**TP Monitor Calls** (provided by TP Monitor)

rmInstall( RMName, Callbacks, Code, ...)

rmActivate( RMIId)

rmDeactivate()

rmRemove()

rmStartup()
**TM Calls** (provided by TM)

- `rmStartup()`
- `Identify()`
- `Begin_Work()`
- `Commit_Work()`
- `Abort_Work()`
- `Save_Work()`
- `Rollback_Work()`
- `Read_Context()`
- `Chain_Work()`
- `Leave_Transaction()`
- `Resume_Transaction()`
Calls (stateless) vs. Sessions (stateful)

or: who keeps context?

[see textbook page 305, Figure 6.4]

[see textbook page 307, Figure 6.5]

Two Types of Context

• Client Oriented  
  ⇒ bound to client, e.g. value of variable

• Transaction Oriented  
  ⇒ bound to transaction, e.g. lock
How Are Processes Bound

or: how to create a session

(a) on first call, return id for p2
   on next call, ask for p2 "by name"

(b) p2 could tell TP monitor "p1 & p2 are in session"

we want exactly this process
### Example

<table>
<thead>
<tr>
<th>Client</th>
<th>SQL Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ctx_a = null</code></td>
<td></td>
</tr>
<tr>
<td><code>rmCall(SQL, &quot;open&quot;, &amp;ctx_a)</code></td>
<td>parameter BindID is <code>ctx_a</code></td>
</tr>
<tr>
<td></td>
<td>BindID=rmBind(clientRMID)</td>
</tr>
<tr>
<td><code>rmCall(SQL, &quot;next&quot;, &amp;ctx_a)</code></td>
<td>get next record</td>
</tr>
<tr>
<td><code>rmCall(SQL, &quot;close&quot;, &amp;ctx_a)</code></td>
<td>parameter BindID is <code>ctx_a</code></td>
</tr>
<tr>
<td></td>
<td>rmUnBind(BindID)</td>
</tr>
</tbody>
</table>

...
Chained Transactions

Motivation:

\[
\ldots
\]

\[
\begin{align*}
& \text{Begin\_Work()} \\
& \text{rmCall}(\text{SQL, "open"}, \ & p) \\
& \text{for } i=1 \text{ to } n \text{ do} \\
& \quad \text{rmCall}(\text{SQL, "next"}, \ & p) \\
& \quad \text{process record & update} \\
& \text{rmCall}(\text{SQL, "close"}, \ldots) \\
& \text{Commit\_Work()} \\
& \ldots
\end{align*}
\]

Not Good:

\[
\ldots
\]

\[
\begin{align*}
& \text{rmCall}(\text{SQL, "open"}, \ & p) \\
& \text{for } i=1 \text{ to } n \text{ do} \\
& \quad \text{Begin\_Work()} \\
& \quad \text{rmCall}(\text{SQL, "next"}, \ & p) \\
& \quad \text{process record & update} \\
& \quad \text{Commit\_Work()} \\
& \text{rmCall}(\text{SQL, "close"}, \ldots) \\
& \ldots
\end{align*}
\]

\[\Rightarrow\text{loses all information about cursor in every commit; failures!}\]
With Chained Transactions:

Begin_Work()
rmCall(SQL, "open", &p)
for i=1 to n do
    Chain_Work(persistent = TRUE, &p)
    rmCall(SQL, "next", &p)
    process record & update
rmCall(SQL, "close", &p, ...)
Commit_Work()
Data Structures Used by TP Monitor

• Where are they kept?
• What do they represent?

[see textbook page 314, Figure 6.6]

[see textbook page 316, Figure 6.7]

[see textbook page 319, Figure 6.8]

Guided Tour of TRPC

• local call processing
• preparing outgoing call
• incoming call processing
• call delivery
• call return
Domain Switches

- same process switches to new domain (new resource manager)

[see textbook page 320, Figure 6.9]

Example: Doing Domain Switch

rmCall(RMName, ...)

1) translate RMName -> DestRMId
2) see if I can domain switch current process toDestRMId
3) if not, need new process
PCBP:  process control block of current process

PRRM-CBP:  pointer to list of RM control blocks
(RMs whose address space I may switch to)

DESTRMID:  ID of RM we are calling
Here is how it works

PCBP = MyProc();
...
PRRM_CBP = (PCBP -> IMayUse);

follow PRRM_CBP chain until we find
PRRM_CBP.RMID == DESTRMID

if found then
    PCBP.ClientID = PCBP.RunsIn
    PCBP.RunsIn = DESTRMID
    DomainSwitch(DESTRMID);
else
    find a process ...
Summary so far

- calls to RM, TP Monitor, TM
- starting up a RM
- context
- TP Monitor data structures
- TRPC execution
Course Summary - so far

Chapter 1  Introduction
Chapter 2  Basic Hardware & Software
Chapter 3  Fault Tolerance
Chapter 4  Transaction Models - mostly skipped
Chapters 5 & 6  TP Monitor
Queues

- many uses in TP
- many types
  - volatile vs. durable
  - visible to client vs. invisible
  - ACID vs. weaker properties
  - uni-directional vs. bi-directional
  - one vs. many enqueuers/dequeuers
  - FIFO, priority, ...
Already Covered

- queue to handle insufficient processes

⇒ volatile, invisible, uni-directional, multiple enqueuers/dequeuers
Durable Queues

transaction 1

client

send
receive
request
answer

transaction 2

server

receive
send
request
answer

transaction 3
Another Example

transaction 1

client → server A → server B

transaction 2

T_1 → T_1 → T_1 → ... saga?

transaction 3
Third Example
Placement of the Queue

Queue in a TP Monitor?

TP system (old fashioned)
eg. IMS/DC
Or in Queue RM?

- DB/SQL RM stores the queue data
- TP monitor handles rmCalls, just as any call
Example: QRM keeps Queues in Table

QuId short id of queue

RQID request id

Request actual request

Answer answer

KeepThat to help dumb client

Delete has been received by client

FromRM RMId of client

#Dequeues explained later

TimeStamp time of request
3 Examples

1) client-server with single queue (in class)

2) client-server with 2 queues (in homework)

3) client-server with 1.5 queues (in textbook)

   a little chaotic
Client Interface

\[(\text{ok, quid, last\_sent, last\_recv}) = \text{Connect}(\text{quname, ownerRM})\]

- translates \text{quname (+ownerRM)} \Rightarrow \text{quid}

- \text{ok=TRUE if call ok}

- if new connect then
  \[
  \text{last\_sent} = \text{last\_rec} = \text{undef}
  \]

- else
  \[
  \text{last\_sent is RQId of last request,}
  \text{last\_recv is RQId of last answer received.}
  \]
Client Interface

ok = Send( quid, rqid, request)

(ok, rqid, answer) = Receive( quid, keepThat)

(ok, rqid, answer, keepThat) = ReReceives(quid)

(ok, rqid, answer) = RecvOrCancel( quid, rqid)

ok = Disconnect(quid)
Server Interface

( OK, quid ) = DefineQueue( quname, ownerRM, access, ... )
ok = StartQueue ( quid )
ok = StopQueue ( quid )
ok = ClearQueue ( quid )
ok = UndefineQueue ( quid )
( OK, rqid, request ) = FindEntry ( quid )
ok = WriteEntry ( quid, rqid, answer )
**Example:** Ticket Sales

req(flight, credit card#)

client → confirmation code ← ticket printer

→ ticketing application
Client Program

```plaintext
...
Begin_Work()
(Ok, quid, ls, lr)=Connect("ticketing", ..)
if !Ok keepTrying?
Commit_Work()
if ls != null then
    DoRecovery()
else
    rqid = 1
while newCustomer is here do
    DoWork()
    rqid = rqid + 1
```
DoWork

see what customer wants...
Begin_Work()
ok = Send(quid, rqid, "get me a ticket")
if !ok then try again? give up? ... 
Commit_Work()
Print("Your request is being processed...")

Begin_Work()
(ok,rqid2,answer) = Receive( quid,ticket_counter)
Commit_Work()

if !ok then
    timeout? disconnect? try RecvOrCancel?
if rqid2 != rqid then can’t cope! help!

using answer compose ticket
PrintTicket // atomic advance counter
Recovery

// did connect and ls != NULL
if ls=lr then { // what happened to lr?
  Begin_Work()
  (ok,rqid,answr,keepThat)= ReReceive(quid)
  Commit_Work();
  if not ok then ...
  if ticket_counter = keepThat then {
    composeTicket
    printTicket
  }
  rqid = rqid + 1
} else if ls=ls+1 then {
  Begin_Work()
  (ok,rqid,answr,keepThat)= ReReceive(quid)
  Commit_Work()
  process reply as in Do_Work
  PrintTicket
} else
  cannot cope! help!
// either ls = lr or ls = lr + 1. Thats it!
Server

\[
(\text{ok}, \text{quid}) = \text{DefineQueue}("\text{ticketing}", \ldots) \\
\text{ok} = \text{StartQueue}(\text{quid}) \\
\text{while TRUE do} \{ \\
\quad \text{Begin\_Work()} \\
\quad \text{ok, rqid, request} = \text{FindEntry}(\text{quid}) \\
\quad \text{process request, prepare answer} \\
\quad \text{ok} = \text{WriteEntry( quid, rqid, answer)} \\
\quad \text{Commit\_Work()} \\
\}\n\]
What if Voluntary Abort?

```plaintext
... while TRUE do {
    Begin_Work()
    ... FindEntry ...
    process request
    if bad_credit then  Abort_Work() else {
        prepare answer
        WriteEntry(...)
        Commit_Work()
    }
}
⇒ abort places request back in queue!
```
Solution #1:
⇒ No Aborts

...  
if result not ok then {
  sp=Rollback_Work(1)
  prepare negative answer
} else {
  prepare positive answer
  ok = WriteEntry( quid,rqid,answer)
}

Solution #2:
⇒ Unprotected Queue Update
Solution #3:
⇒ independent transaction

For Server RM: rm_Abort callback:

```
figure out quid, rqid ...
oldTrid = LeaveTransaction(...)  
newTrid = Begin_Work()
mark queue entry as aborted (or increment # Dequeue)
Commit_Work()
ResumeTransaction( oldTrid)
```
Problem: 2 Transaction Access Queue

Scenario:
• $T_1$ running on server RM accesses queue $\Rightarrow$ queue entry locked by $T_1$
• $T_1$ aborts
• TM calls rmAbort callback of server RM
• rmAbort wants to update queue entry (by starting a new transaction $T_2$) to record the fact that transaction aborted

$\Rightarrow$ queue entry still locked by $T_1$ $\Rightarrow$ Problem!

Solution:
• Specify abort/commit order, i.e. order of calling callbacks.
• in example above: call QRMrmAbort first, this will unlock queue entry
Connection Stateful...

Queue RM keeps state ⇒ is responsible for recovery!

...or Stateless?

State kept in Queue DB ⇒ DB RM handles recovery!

KeepThat

Only needed because
• client is dumb and
• server is lazy!

Alternative

• Clients hands it’s state over to QRM; QRM keeps it in queue and hands it back with the answer
Queue Concurrency Control

• FIFO queue

• set queue

⇒ not strict! Need read past locking or less than full isolation. (“find me next unlocked entry”)
How to Implement Queue on SQL DB

Section 6.4.2.3 of textbook:

Questions

in table def: to_rmid? Where in request?
in send(p.343): store request, not keepThat
in receive: need to get answer back?
          need to store keepThat (p.344)?

Lots of bugs in code in textbook pages 342 - 344! Skim code!
Other Tasks of TP Monitor

• load balancing
  - it is hard!
  - Fig. 6.15
  - do not worry about equations and their derivations!

• authentication

• restart processing

• [see textbook page 351, Figure 6.15]
Summary of Chapter 6

• TP Monitor Data Structures

• RM, TM, TP Monitor Calls

• TRPC Guided Tour

• Queues

• Other Tasks