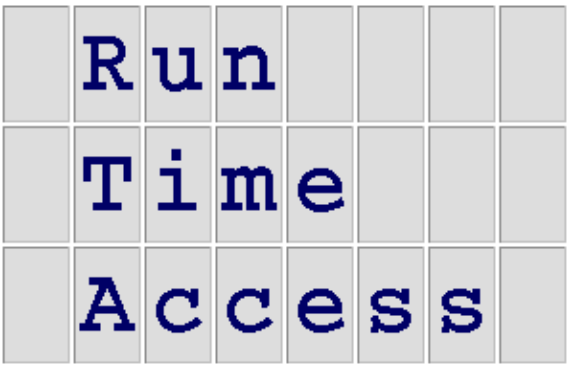
door Bob Smith
<bob/at/linuxtoys.org>

Over de auteur:

Bob is een Linux programmeur en electronica hobbyist. Zijn nieuwste project is te vinden op www.runtimeaccess.com en zijn homepage is te vinden op www.linuxtoys.org.

Vertaald naar het Nederlands door:
Guus Snijders
<ghs(at)linuxfocus.org>

Praten met een lopend proces



Kort:

Run Time Access is een library waarmee je de data structuren in je programma kunt weergeven als tabellen in een PostgreSQL database of als bestanden in een virtueel bestandssysteem (vergelijkbaar met /proc). Met RTA kun je je daemon of service makkelijk van verschillende soorten management interfaces voorzien, zoals web, shell, SNMP of framebuffer.

10 Seconden Overzicht

Stel dat je een programma hebt met data in een array van structuren. De structuur en array zijn gedefinieerd als:

```
struct mydata {
    char   note[20];
    int    count;
}

struct mydata mytable[] = {
  { "Sticky note", 100 },
  { "Music note", 200 },
  { "No note", 300 },
};
```

Als je je programma bouwt met de Run Time Acces library, kun je de interne data van het programma vanuit de shell of een ander programma bekijken en veranderen. Je data verschijnt alsof het zich in een PostgreSQL database bevindt. Het volgende illustreert hoe je Bash en psql, de commando–regel tool van PostgreSQL, kunt gebruiken om de data in je programma kunt lezen en aanpassen.

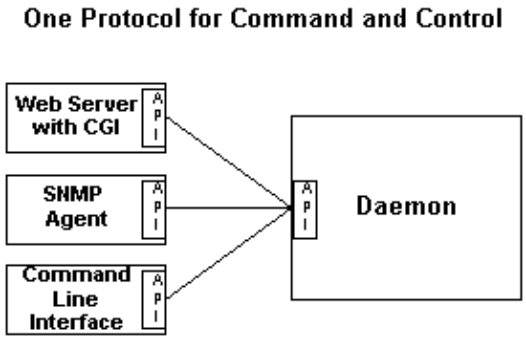
```
# myprogram  &

# psql -c "UPDATE mytable SET note = 'A note' LIMIT 1"
UPDATE 1

# psql -c "SELECT * FROM mytable"
   note      | count
-------------+-------
 A note      | 100
 Music note  | 200
 No note     | 300

#
```

Dit artikel legt uit waarom iets als RTA nodig is, hoe de RTA library gebruikt wordt en welke voordelen je kunt verwachten van RTA.

Vele UIs –– Een service

Traditioneel communiceert UNIX met een service door diens configuratie data in /etc/applicatie.conf te plaatsen en de verzamelde uitvoer in /var/log/applicatie.log. Deze geaccepteerde benadering is waarschijnlijk minder geschikt voor de services van vandaag die op een appliance draaien en geconfigureerd worden door relatief weinig getrainde sysadmins. De traditionele benadering faalt omdat we nu meerdere types simultane gebruikersinterfaces willen, en het liefst dat die interfaces configuratie–, status– en statistische informatie uitwisselen terwijl de service actief is. Daarvoor is runtime toegang nodig.

Moderne services vragen veel soorten gebruikers interfaces en de ontwikkelaars zullen niet kunnen voorspellen welke interface het meest gebruikt gaat worden. Wat we moeten doen is de gebruikers–interface scheiden van de service via een algemeen protocol en vervolgens de gebruikers–interface bouwen met dit protocol. Dit maakt het gemakkelijker om interfaces toe te voegen wanneer nodig en de scheiding maakt het testen makkelijker omdat ieder onderdeel onafhankelijk getest kan worden. We willen een architectuur die er ongeveer zo uit ziet:



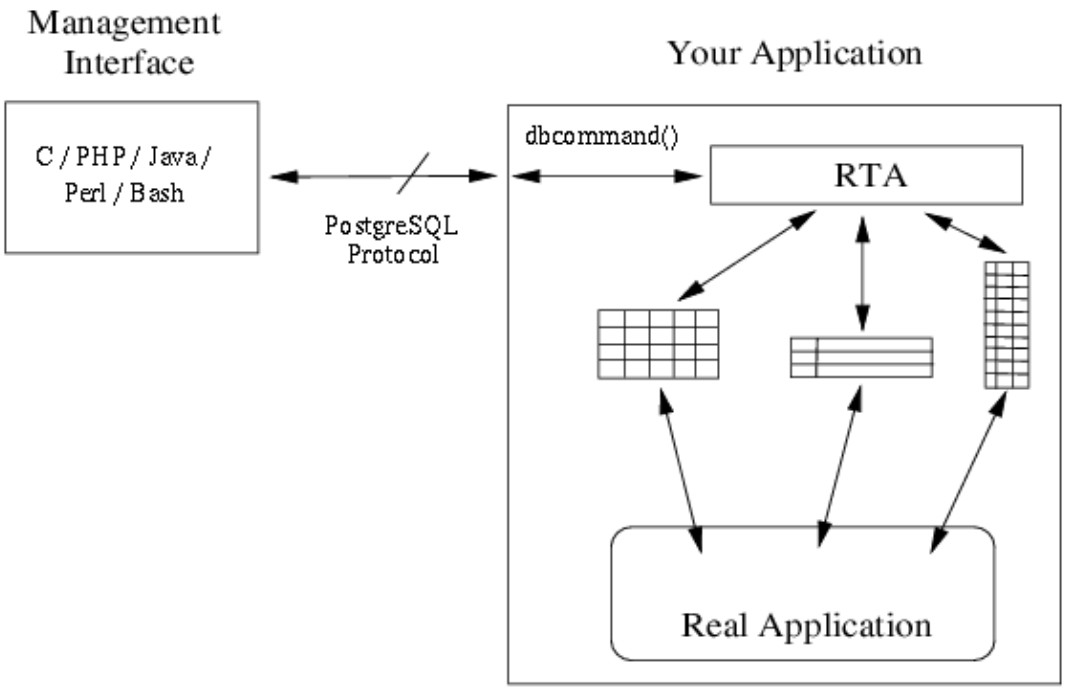
De soorten gebruikersinterface om te beschouwen, omvatten web, opdracht regel, framebuffer, SNMP, keypad en LCD, LDAP, native Windows en andere custom interfaces. Een algemene API en protocol voor alle gebruikers interfaces zou uiteraard een goed idee zijn. Maar welk soort API en protocol?

Een Database Interface

RTA gebruikt een PostgreSQL database als de algemene API en protocol. Configuratie, status en statistieken worden als arrays van structuren geplaatst die op de API verschijnen als tabellen in een PostgreSQL database. De gebruikers−interface−programma's zijn geschreven als clients die een verbinding opzetten met een PostgreSQL database. Deze benadering biedt twee grote voordelen:

- De gebruikers−interface clients gebruiken een bekend, goed gedocumenteerd en goed debugged API. Het gebruik van PostgreSQL reduceert de ontwikkeltijd dramatisch. Verder heeft PostgreSQL bindingen voor C, Java, PHP, Perl en bijna alle andere populaire talen zodat je de UI in de meest geschikte taal voor de taak kunt schrijven.
- Het paradigma van *tabel in een database* komt vrij goed overeen met hoe de meesten van ons programma's schrijven die een service leveren. We gebruiken datastructuren voor wat *rijen* kunnen zijn en arrays of linked−lists voor wat *tabellen* zouden zijn.

De RTA library is de lijm die onze arrays of gelinkte lijsten van de data structuren met de PostgreSQL clients verbindt. De architectuur van een applicatie die RTA gebruikt, ziet er ongeveer uit als...



Hier noemen we het een *management interface* daar hij is bedoeld voor status, statistieken en configuratie. Hoewel er maar een interface is weergegeven, zou je er aan moeten denken dat je vele interfaces voor je applicatie kunt hebben die allen de applicatie tegelijkertijd kunnen benaderen.

PostgreSQL gebruikt TCP als transport protocol, dus je applicatie moet kunnen binden aan een TCP poort en verbindingen van de verschillende gebuikers−interfaces accepteren. Alle bytes die worden ontvangen in een geaccepteerde verbinding worden doorgegeven aan de RTA met de dbcommand() subroutine. Alle data om terug te sturen naar de client bevindt zich in een buffer die wordt geretourneerd door dbcommand().

Hoe weet RTA welke tabellen beschikbaar zijn? Je moet het hem vertellen.

Tabellen definiëren

Je kunt RTA over je tabellen met data structuren door de rta_add_table() subroutine aan te roepen. De TBLDEF data structuur beschrijft een tabel en de COLDEF structuur beschrijft een kolom. Hier is een voorbeeld dat illustreerd hoe je een tabel toevoegt aan de RTA interface.

Stel dat je een data structuur met een string, lengte 20, en een integer, en dat je de tabel wilt exporteren met 5 van deze structuren. Je kunt de structuur als volgt definiëren:

```
struct myrow {
    char    note[20];
    int     count;
};

struct myrow mytable[5];
```

Ieder veld in de `myrow` data structuur is een kolom in een database tabel. We vertellen RTA de naam van de kolom, in welke tabel die zich bevind, het data type, de offset vanaf het begin van de rij en of hij wel/niet read–only is. Ook kunnen we *callback* routines definiëren die worden aangeroepen voordat de kolom wordt gelezen en/of na schrijven daarnaar. Voor ons voorbeeld nemen we aan dat `count` read–only is en dat we `do_note()` willen aanroepen als er wordt geschreven naar het `note` veld. We bouwen een array van COLDEF die wordt toegevoegd aan TBLDEF, welke een COLDEF heeft voor ieder strucuur lid.

```
COLDEF mycols[] = {
    {
        "atable",           // tabel naam voor SQL
        "note",             // kolom naam voor SQL
        RTA_STR,            // data type van kolom/veld
        20,                 // kolombreedte in bytes
        0,                  // offset vanaf de start van een rij
        0,                  // bitwise OR van boolean flags
        (void (*)()) 0,     // wordt aangeroepen voor lezen
        do_note(),          // wordt aangeroepen na schrijven
        "Het laatste veld van een kolom-definitie is een "
        "string die de kolom beschrijft. Je wilt "
        "waarschijnlijk uitleggen wat de data in de kolom "
        "betekend en hoe het wordt gebruikt."}
    {
        "atable",           // tabel naam voor SQL
        "count",            // kolom naam voor SQL
        RTA_INT,            // data type van kolom/veld
        sizeof(int),        // kolombreedte in bytes
        offsetof(myrow, count),  // offset vanaf de start van de rij
        RTA_READONLY,       // bitwise OR van boolean flags
        (void (*)()) 0,     // wordt aangeroepen voor lezen
        (void (*)()) 0,     // wordt aangeroepen na schrijven
        "Als je tabellen de interface zijn tussen de "
        "gebruikers-interfaces en de service, dan vormen de "
        "commentaren in kolom en tabel definities de "
        "functionele specificatie voor je project en zijn "
        "wellicht de beste documentatie voor ontwikkelaars."
};
```

Het gebruik van callbacks kan de echte motor van applicatie vormen Je kunt veranderingen aan een tabel andere veranderingen laten veroorzaken of zelfs een herconfiguratie van je applicatie.

Je vertelt RTA over tabellen door de naam van de tabel te geven, de lengte van iedere rij, een array van COLDEFS om de kolommen te beschrijven, het aantal kolommen, de naam van het bestand om op te slaan −als er velden zijn die wilt behouden−, en een string om de tabel te beschrijven. Als de tabel een statische array van structs is, geef je het start adres en het aantal rijen in de tabel. Als de tabel geïmplementeerd is als een gelinkte lijst, geef je de RTA een routine die *itereert* van de ene rij naar de volgende.

```
TBLDEF   mytableDef = {
      "atable",                 // tabel naam
      mytable,                  // adres van de tabel
      sizeof(myrow),            // lengte van iedere rij
      5,                        // aantal rijen
      (void *) NULL,            // iterator functie
      (void *) NULL,            // iterator callback data
      mycols,                   // Kolom definities
      sizeof(mycols / sizeof(COLDEF), // # kolommen
      "",                       // bestandsnaam
      "Een complete beschrijving van de tabel. "
};
```

Normaal gesproken doe je het zo dat de naam, zoals gezien in SQL gelijk is aan die binnen in het programma. Het voorbeeld veranderde van `mytable` naar `atable` om te laten zien dat ze niet per se gelijk hoeven te zijn.

Met alle bovengegeven code, kun je nu RTA over je tabel vertellen.

```
    rta_add_table(&mytableDef);
```

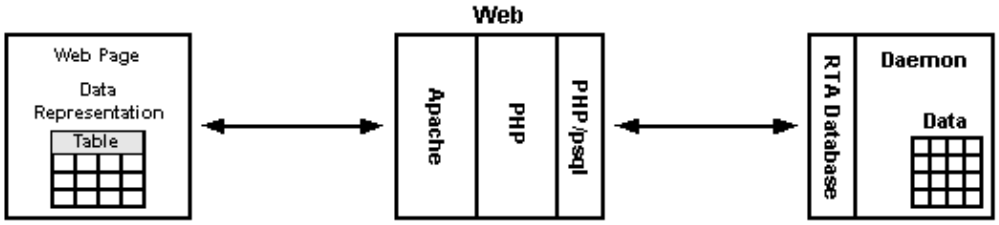
Zo eenvoudig is het. Om RTA te leren gebruiken, hoef je je alleen aan te leren om twee data structuren (COLDEF en TBLDEF) te gebruiken en twee subroutines (dbcommand() en rta_add_table()).

De bovenstaande code is bedoeld als voorproefje van RTA. Het is niet bedoeld als een volledige tutorial of een compleet werkend voorbeeld. Een volledig werkend voorbeeld en een complete beschrijving van de RTA API en datastructuren kun je vinden op de RTA website (www.runtimeaccess.com).

Net zoals je tabellen definieert in je applicatie, kan RTA zijn eigen set interne tabellen definiëren. De twee meest interessante tabellen zijn rta_tables en rta_columns, deze zijn uiteraard tabellen die alle gedefinieerde tabellen en kolommen beschrijven. Dit zijn de zogenaamde *systeemtabellen*. De systeemtabellen doen voor een database wat `ls` doet voor een bestandssysteem en `getnext()` doet voor SNMP.

De Tabel Editor

Een van de utillities die met RTA meekomen is een klein PHP programma dat de systeemtabellen gebruikt om je RTA tabellen in een browser venster weer te geven. De tabelnamen zijn links en klikken op de tabelnaam geeft de eerste 20 rijen van de tabel. Als de tabel bewerkbare velden heeft, kun je op een rij klikken om een edit−venster te openen voor die rij. Dit alles gebeurd met behulp van de systeemtabellen en de kolom−beschrijvingen die in de systeemtabellen staan. De datastroom is in het onderstaande diagram weergegeven.



De top level view van de tabel editor met de weergaven van de voorbeeld RTA applicatie is hieronder weergegeven.

RTA Tabel Editor

Tabel Naam	Beschrijving
rta_tabellen	De tabel van alle tabellen in het systeem. Dit is een pseudo tabel en geen array van structuren zoals de andere tabellen.
rta_kollummen	De lijst van alle kolommen in alle tabellen, samen met hun attributen.
pg_user	De tabel met Postgres gebruikers. We vervalsen deze tabel zodat iedere gebruiker in een WHERE clausule in de tabel verschijnt als een legitieme gebruiker zonder super, createDB, trace of catupd mogelijkheden.
rta_dbg	configuratie van debug logging. Een callback van het 'target' veld sluit en heropend syslog(). Geen van de waarden in deze tabel worden op schijf opgeslagen. Als je niet−standaard waarden wilt, kun je de rta broncode aanpassen of een SQL_string() gebruiken om de waarden in te stellen tijdens het initialiseren van je programma.
rta_stat	

mijntabel	Een voorbeeld applicatie tabel
UIConns	Data over TCP connecties van UI frontend programma's.

Overigens, als alles goed is gegaan bij het publiceren van dit LinuxFocus artikel, geven de tabelnamen hierboven links naar de voorbeeld applicatie die draait op de RTA webserver in Santa Clare, Californië. Een goede link om te volgen is de `mytable` link.

Twee Commando's

Run Time Access is een library die beheer– (management) of gebruiker–interface programma's, geschreven met de PostgreSQL client library (libpq), verbindt met je applicatie of daemon. RTA is een interface, geen database. Daardoor heeft het slechts twee SQL commando's nodig: SELECT en UPDATE.

De syntax voor het SELECT statement is:

```
SELECT column_list FROM table [where_clause] [limit_clause]
```

De column_list is een comma–gescheiden lijst van kolomnamen. De where_clause is een AND gescheiden lijst van vergelijkingen. De vergelijkings operators zijn =, !=, >=, <=, >, en <. Een limit_clause heeft de vorm `[LIMIT i] [OFFSET j]`, waarbij `i` het maximale aantal te–retourneren–rijen is, en we `j` rijen overslaan alvorens met de uitvoer te beginnen. Enkele voorbeelden kunnen helpen dit te verduidelijken:

```
SELECT * FROM rta_tables

SELECT notes, count FROM atable WHERE count > 0

SELECT count FROM atable WHERE count > 0 AND notes = "Hi Mom!"

SELECT count FROM atable LIMIT 1 OFFSET 3
```

De LIMIT op 1 en het opgeven van een OFFSET is een manier om een specifieke rij te krijgen. Het laatste voorbeeld hierboven is gelijk aan de C code `(mytable[3].count)`.

De syntax van het UPDATE statement is:

```
UPDATE table SET update_list [where_clause] [limit_clause]
```

De where_clause en limit_clause zijn zoals hierboven beschreven. De update_list is een comma–gescheiden lijst van kolom toewijzingen. We zullen enkele voorbeelden gebruiken om dit te verduidelijken.

```
UPDATE atable SET notes = "Not in use" WHERE count = 0

UPDATE rta_dbg SET trace = 1

UPDATE ethers SET mask = "255.255.255.0",
                  addr = "192.168.1.10"
              WHERE name = "eth0"
```

RTA herkent gereserveerde woorden zowel in hoofd– als in kleine letters, al gebruiken de voorbeelden hierboven hoofdletters voor alle SQL–gereserveerde woorden.

Downloaden en Bouwen

RTA kun je downloaden de website op www.runtimeaccess.com (RTA is LGPL gelicenseerd). Let op welke versie van RTA je download. De nieuwste RTA versie gebruikt het nieuwere PostgreSQL protocol, dat werd geïntroduceerd met versie 7.4 van PostgreSQL. De meeste huidige Linux distributies gebruiken versie 7.3.

Hoewel je een oudere versie van RTA kunt gebruiken om bekend te raken met het gebruik ervan, zou je de nieuwste versie moeten gebruiken voor de laatste bug−fixes en uitbreidingen.

Het uitpakken van het pakket zou je de volgende directories moeten opleveren:

```
./doc            # een kopie van de RTA website
./empd           # een prototype daemon met RTA
./src            # bronbestanden voor de RTA library
./table_editor   # PHP code voor de tabel editor
./test           # code voor een voorbeeld applicatie.
./util           # utilities voor het schrijven van RTA
```

Dankzij Graham Philips, ondersteund versie 1.0 van RTA autoconf. Graham portte RTA van Linux naar Mac OS X, Windows en FreeBSD. Met de 1.0 release kun je RTA bouwen met de gebruikelijke

```
./configure
make
make install      # (als root)
```

De installatie plaatst librtadb.so en de bijbehorende bestanden in de /usr/local/lib directory. Om RTA te gebruiken kun je deze directory toevoegen aan /etc/ld.so.conf en het ldconfig commando starten, of je kunt de directory toevoegen aan je loader path met:

```
export LD_LIBRARY_PATH=/usr/local/lib
```

De installatie plaatst het RTA header bestand, rta.h, in /usr/local/include.

De make bouwt een test programma in de `test` directory en je kunt je installatie testen door naar de test directory te gaan en `./app &` uit te voeren. Een `netstat −nat` zou een programma moeten laten zien dat luistert op poort 8888. Nu kun je psql starten en SQL commando's aan je test applicatie voeren.

```
cd test
./app &

psql −h localhost −p 8888
Welcome to psql 7.4.1, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

# select name from rta_tables;
    name
−−−−−−−−−−−−−
 rta_tables
 rta_columns
 rta_dbg
 rta_stat
 mytable
 UIConns
(6 rows)
```

Hoewel het eruit ziet alsof je bent verbonden met een database, ben je dat niet. Vergeet niet: de enige twee commando's die je kunt gebruiken zijn SELECT en UPDATE.

Voordelen van RTA

De voordelen van het splitsen van de gebruikers−programma's van de daemon vallen in de brede categoriën van ontwerp, coden, debug en mogelijkheden.

Vanuit het oogpunt van ontwerp, dwingt de scheiding je om vroeg in het ontwerp te beslissen wat er precies in de UI wordt aangeboden, zonder je zorgen te maken over hoe het wordt weergegeven. Het denkproces dat nodig is om de tabellen te ontwerpen, dwingt je om na te denken over het echte ontwerp van je applicatie. De tabellen kunnen de interne functionele specificatie vormen van je applicatie.

Tijdens het coden kunnen de tabel−definities zijn waar de daemon ontwerpers naartoe bouwen en de UI ontwerpers vanaf bouwen. De scheiding van UI en daemon betekend dat je UI experts en daemon experts afzonderlijk kunt huren en ze onafhankelijk laten werken, wat ertoe kan bijdragen dat je product sneller op de markt komt. Daar er Postgres bindingen zijn voor PHP, Tcl/Tk, Perl en "C", kunnen je ontwikkelaars de juiste tool voor de job gebruiken.

Debug is sneller en makkelijker, omdat zowel de UI als de daemon ontwikkelaars de andere helft eenvoudig kunnen simuleren. Zo kunnen bijvoorbeeld de UI ontwikkelaars hun UI programma's tegen een echte PostgresDB uitvoeren, mits deze dezelfde tabellen als de daemon heeft. Het testen van de daemon kan sneller en completer, doordat test scripts om de UI te simuleren makkelijk kunnen worden gemaakt en het eenvoudig is de interne status en statistieken te bekijken tijdens een test. De mogelijkheid om een interne staat of conditie af te dwingen helpt om extreme gevallen te testen, deze kunnen anders soms lastig zijn in een lab setup.

De mogelijkheden van je product kunnen worden uitgebreid met RTA. Je klanten zullen het zeker kunnen waarderen om gedetailleerde status informatie en statistieken te zien terwijl het programma draait. Het scheiden van de UIs en de daemon betekend ook dat je meer UI programma's kunt hebben: SNMP, opdrachtregel, web, LDAP en de lijst gaat maar door. Deze flexibiliteit is belangrijk als (wanneer!) je klanten om custom UIs vragen.

RTA biedt verscheidene andere features die je in een pakket van een dergelijk type wilt:

- Applicatie model komt overeen met het API data model
- Toegang tot de applicatie op afstand (remote access)
- Gebruik van standaarden en bestaande software bij de applicatie
- Weinig nieuwe protcollen en APIs om te leren
- Ontdekkings (Discovery) mechanismen voor de applicatie
- Weinig remmingen tot de applicatie
- Bron reservering (Resource locking)
- Efficiënt CPU en geheugen gebruik

Samenvatting

Dit artikel geeft een erg korte introductie tot de RTA library en diens mogelijkheden. De RTA website heeft een FAQ, een complete beschrijving van de API en verschillende voorbeeld client−programmas.

Met RTA kun je je datastructuren zichtbaar maken als tabellen in een database en dus kun je ze ook zichtbaar maken als bestanden in een virtueel bestandssysteem (gebruik hiervoor het File System in Userspace (FUSE) pakket van Miklos Szeredi). De website heeft meer informatie over hoe je de bestandssysteem−interface kunt gebruiken.

2005−07−20, generated by lfparser_pdf version 2.51