Designing MC/ServiceGuard Monitor Scripts

David Totsch

Account Support Engineer

Hewlett-Packard Company

Follow the Process

- ps -f versus ps -e
- ps -eu uname
- ps -p

Following More than One


```
PROCNAME[0]=""
PROCNAME[1]=""
PROCNAME[2]=""
PROCNAME[3]=""

USERNAME="root"

GETPID()
{
ps -eu ${USERNAME} | \
   awk '$NF ~ ^'${1}'$/ {print $1}'
}
```

Following More than One


```
REF=0
while [[ ${REF} < ${#PROC[@]} ]]
do
   PIDS[${REF}]=$(GETPID ${PROC[${REF}]})
   if [[ -z ${PIDS[${REF}]} ]]
       then
       print -n "Unable to determine PID "
       print "for >${PROC[${REF}]}<."
   fi
   (( REF += 1 ))
done
```

Following More than One


```
while true
   for PID in ${PIDS[@]}
   do
        if ps -fp ${PID} > /dev/null 2>&1
        then
                :
        else
                print "PID ${PID} missing!"
                exit 1
        fi
   done
do
```

What if the process can restart?

Check the Database

- A process can be in the process stack, but still…
  - be hung in an endless loop
  - be blocked on a critical resource
  - be unresponsive

There is a Better Way


```
isql  -U ${SYB_USER} \
      -S ${SYB_INST} <<- EOI
           ${SYB_PASS}
           select getdate()
           go
           EOI
```

The Subtleties of SQL

- Selecting an SQL user
  - not a database management id
- Keeping the SQL password
  - You will want to change it occasionally
- SQL code
  - Proof database is up (maybe perf. data?)
- SQL exit status
  - non-zero $? means syntax error

What If SQL Hangs?


```
RC=""
Call_ISQL > ${ES_FILE} &

# Wait up to 15 seconds for isql
# then consider it "hung".
for I in 2 3 5 5
do
  if [[ -s ${ES_FILE} ]]
  then
      RC=$(<$(ES_FILE})
      rm ${ES_FILE}
      break
  fi
  sleep ${I}
done
```

Disable/Enable

- Sooner or later…
  - the database will require maintenance
  - Security will require a password change

Method A

By Signal

```
trap monitor SIGUSR1
trap disabled SIGUSR2

monitor()
{
}

disabled()
{
}

monitor
```

Method B

By File

```
monitor()
{
   if [[ -f ${MONITOR_LOCK} ]]
   then
       print -n "Monitoring disabled at "
       date
       print -n "\tRemove ${MONITOR_LOCK} "
       print "to re-engage."
       disable
   fi
   .
   .
   .
```

Method B

By File

```
disabled()
{
   while [[ -f ${MONITOR_LOCK} ]]
   then
       print -n "Monitoring disabled."
       sleep ${DISABLED_INTERVAL}
   fi
   print -n "Monitoring ENABLED at "
   date
   monitor
}
```

Advantages/Disadvantages

- By Signal
  - only root can signal a process run by root or another user
  - only two user assigned signals available
- By File
  - Whomever controls the file controls the monitor script

Check on Re-enable

- Wouldn't it be nice if you could ask the monitor to check before you enable it?
  - Wrap the checking process into a function.
  - Set a trap to run the function and only report the messages to the log file.

Nice Process

- Disable
- Perform maintenance
- Check
- Enable

WARNING!


```
MAX_DISABLED=120  # in minutes
DISABLED_SLEEP=5  # in minutes

# Inside of disabled function
DTIMR=0
sleep $(( ${DISABLED_SLEEP} * 60 ))
(( DTIMR += ${DISABLED_SLEEP} ))

if (( ${DTIMR} < ${MAX_DISABLED} ))
then
  print "Monitor disabled too long!"
  # send e-mail to pager, etc.
fi
```

Re-configuration

- Wouldn't it be nice to…
  - change the SQL user and password
  - modify the process list
  - use different SQL code
  - change time intervals

Re-configuration with Autoload Functions

- Autoload functions
  - **FPATH=**
  - **typeset -f FNAME**
  - **unset -f FNAME**

Re-configuration With Autoload Functions


```
LOADFUNCTIONS()
{
unset -f MYFN
typeset -f MYFN
}

trap LOADFUNCTIONS SIGUSR1
```