

```

*****
2874 Thu Feb 26 16:21:24 2009
new/usr/src/cmd/boot/scripts/boot-archive-update.xml
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
1 <?xml version="1.0"?>
2 <!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
3 <!--
4 Copyright 2009 Sun Microsystems, Inc. All rights reserved.
4 Copyright 2006 Sun Microsystems, Inc. All rights reserved.
5 Use is subject to license terms.

7 CDDL HEADER START

9 The contents of this file are subject to the terms of the
10 Common Development and Distribution License (the "License").
11 You may not use this file except in compliance with the License.

13 You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
14 or http://www.opensolaris.org/os/licensing.
15 See the License for the specific language governing permissions
16 and limitations under the License.

18 When distributing Covered Code, include this CDDL HEADER in each
19 file and include the License file at usr/src/OPENSOLARIS.LICENSE.
20 If applicable, add the following below this CDDL HEADER, with the
21 fields enclosed by brackets "[]" replaced with your own identifying
22 information: Portions Copyright [yyyy] [name of copyright owner]

24 CDDL HEADER END

26 ident "%Z%M% %I% %E% SMI"

26 NOTE: This service manifest is not editable; its contents will
27 be overwritten by package or patch operations, including
28 operating system upgrade. Make customizations in a different
29 file.
30 -->

32 <service_bundle type='manifest' name='SUNWcsr:boot-archive-update'>
34 <service
35 name='system/boot-archive-update'
36 type='service'
37 version='1'>

39 <create_default_instance enabled='true' />

41 <single_instance/>

44 <!--
45 This needs to depend on filesystem/local because the boot-archive
46 or the GRUB menu may live in /stubboot.
47 -->
48 <dependency
49 name='filesystem'
50 grouping='require_all'
51 restart_on='none'
52 type='service'>
53 <service_fmri value='svc:/system/filesystem/local' />
54 </dependency>

56 <!--
57 Since updating the boot-archive can take a while on slow machines or

```

```

58 when there are many glommed kernels to wad up, a longer timeout is
59 needed. However if it doesn't finish eventually something up, and
60 an error message to trigger further investigation is appropriate.

62 Please note that a similar timeout exists in startd, which again
63 runs bootadm at the end of shutdown. These timeouts should be
64 adjusted in sync with each other.
65 -->
66 <exec_method
67 type='method'
68 name='start'
69 exec='/lib/svc/method/boot-archive-update'
70 timeout_seconds='3600' />

72 <exec_method
73 type='method'
74 name='stop'
75 exec=':true'
76 timeout_seconds='3' />

78 <property_group name='startd' type='framework'>
79 <propval name='duration' type='astring' value='transient' />
80 </property_group>

82 <stability value='Unstable' />

84 <template>
85 <common_name>
86 <loctext xml:lang='C'>
87 update boot archive if necessary
88 </loctext>
89 </common_name>
90 <description>
91 <loctext xml:lang='C'>
92 This service updates the boot archive if
93 a non fatal file was out of sync or if this
94 is a reconfiguration boot.
95 </loctext>
96 </description>
97 </template>
98 </service>

100 </service_bundle>

```


new/usr/src/cmd/initpkg/rc0.sh

1

```
*****
2491 Thu Feb 26 16:21:27 2009
new/usr/src/cmd/initpkg/rc0.sh
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
1 #!/sbin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 # Common Development and Distribution License, Version 1.0 only
9 # (the "License"). You may not use this file except in compliance
10 # with the License.
11 #
12 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
13 # or http://www.opensolaris.org/os/licensing.
14 # See the License for the specific language governing permissions
15 # and limitations under the License.
16 #
17 # When distributing Covered Code, include this CDDL HEADER in each
18 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
19 # If applicable, add the following below this CDDL HEADER, with the
20 # fields enclosed by brackets "[]" replaced with your own identifying
21 # information: Portions Copyright [yyyy] [name of copyright owner]
22 #
23 # CDDL HEADER END
24 #
25 # Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T.
26 # All rights reserved.
27 #
28 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
29 # Copyright 2004 Sun Microsystems, Inc. All rights reserved.
30 # Use is subject to license terms.
31 #
32 #ident "%Z%M% %I% %E% SMI"
33 #
34 # "Run Commands" for init states 0, 5 and 6.
35 #
36 PATH=/usr/sbin:/usr/bin
37 #
38 if [ -z "$SMF_RESTARTER" ]; then
39     echo "Cannot be run outside smf(5)"
40     exit 1
41 fi
42 #
43 # Export boot parameters to rc scripts
44 #
45 set -- ` /usr/bin/who -r `
46 #
47 _INIT_RUN_LEVEL="$7" # Current run-level
48 _INIT_RUN_NPREV="$8" # Number of times previously at current run-level
49 _INIT_PREV_LEVEL="$9" # Previous run-level
50 #
51 set -- ` /usr/bin/uname -a `
52 #
53 _INIT_UTS_SYSNAME="$1" # Operating system name (uname -s)
54 _INIT_UTS_NODENAME="$2" # Node name (uname -n)
55 _INIT_UTS_RELEASE="$3" # Operating system release (uname -r)
56 _INIT_UTS_VERSION="$4" # Operating system version (uname -v)
57 _INIT_UTS_MACHINE="$5" # Machine class (uname -m)
58 _INIT_UTS_ISA="$6" # Instruction set architecture (uname -p)
```

new/usr/src/cmd/initpkg/rc0.sh

2

```
56 _INIT_UTS_PLATFORM="$7" # Platform string (uname -i)
57 #
58 export _INIT_RUN_LEVEL _INIT_RUN_NPREV _INIT_PREV_LEVEL \
59 _INIT_UTS_SYSNAME _INIT_UTS_NODENAME _INIT_UTS_RELEASE _INIT_UTS_VERSION \
60 _INIT_UTS_MACHINE _INIT_UTS_ISA _INIT_UTS_PLATFORM
61 #
62 if [ -d /etc/rc0.d ]; then
63     for f in /etc/rc0.d/K*; do
64         if [ -s $f ]; then
65             case $f in
66                 *.sh) /lib/svc/bin/lsvcrun -s $f stop;;
67                 *) /lib/svc/bin/lsvcrun $f stop;;
68             esac
69         fi
70     done
71 #
72 # System cleanup functions ONLY (things that end fast!)
73 #
74 for f in /etc/rc0.d/S*; do
75     if [ -s $f ]; then
76         case $f in
77             *.sh) /lib/svc/bin/lsvcrun -s $f start;;
78             *) /lib/svc/bin/lsvcrun $f start ;;
79         esac
80     fi
81 done
82 fi
83 #
84 [ -f /etc/.dynamic_routing ] && /usr/bin/rm -f /etc/.dynamic_routing
85 #
86 trap "" 15
87 #
88 [ -x /usr/lib/acct/closewtmp ] && /usr/lib/acct/closewtmp
89 #
90 /sbin/sync; /sbin/sync; /sbin/sync
```

new/usr/src/cmd/initpkg/shutdown.sh

1

```
*****
4702 Thu Feb 26 16:21:28 2009
new/usr/src/cmd/initpkg/shutdown.sh
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
1 #!/sbin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 # Common Development and Distribution License, Version 1.0 only
9 # (the "License"). You may not use this file except in compliance
10 # with the License.
11 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
12 # or http://www.opensolaris.org/os/licensing.
13 # See the License for the specific language governing permissions
14 # and limitations under the License.
15 #
16 # When distributing Covered Code, include this CDDL HEADER in each
17 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
18 # If applicable, add the following below this CDDL HEADER, with the
19 # fields enclosed by brackets "[]" replaced with your own identifying
20 # information: Portions Copyright [yyyy] [name of copyright owner]
21 #
22 # CDDL HEADER END
23 #
24 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
25 # Copyright 2005 Sun Microsystems, Inc. All rights reserved.
26 # Use is subject to license terms.
27 #
28 # Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T
29 # All Rights Reserved
30 #
31 #ident "%Z%M% %I% %E% SMI"
32 #
33 # Sequence performed to change the init state of a machine. Only allows
34 # transitions to states 0,1,5,6,s,S (i.e.: down or administrative states).
35 #
36 # This procedure checks to see if you are permitted and allows an
37 # interactive shutdown. The actual change of state, killing of
38 # processes and such are performed by the new init state, say 0,
39 # and its /sbin/rc0.
40 #
41 # usage() {
42 #     echo "Usage: $0 [ -y ] [ -g<grace> ] [ -i<initstate> ] [ message ]"
43 #     exit 1
44 # }
45 #
46 # _____
47 #
48 # nologin=/etc/nologin
49 #
50 # Set the PATH so that to guarantee behavior of shell built in commands
51 # (such as echo).
52 #
53 # PATH=/usr/sbin:/usr/bin:/sbin
54 #
55 # Initial sanity checks:
56 # Make sure /usr is mounted
57 # Check the user id (only root can run shutdown)
```

new/usr/src/cmd/initpkg/shutdown.sh

2

```
70 if [ ! -d /usr/bin ]
71 then
72     echo "$0: /usr is not mounted. Mount /usr or use init to shutdown."
73     exit 1
74 fi
75 #
76 if [ -x /usr/bin/id ]
77 then
78     eval ` /usr/bin/id | /usr/bin/sed 's/[^a-z0-9=].*//`
79     if [ "${uid:=0}" -ne 0 ]
80     then
81         echo "$0: Only root can run $0"
82         exit 2
83     fi
84 else
85     echo "$0: can't check user id."
86     exit 2
87 fi
88 #
89 # Get options (defaults immediately below):
90 #
91 grace=60
92 askconfirmation=yes
93 initstate=s
94 #
95 while getopts g:i:y:c
96 do
97     case $c in
98         g)
99             case $OPTARG in
100                 *[!0-9]*)
101                     echo "$0: -g requires a numeric option"
102                     usage
103                     ;;
104                 [0-9]*)
105                     grace=$OPTARG
106                     ;;
107             esac
108             ;;
109         i)
110             case $OPTARG in
111                 {Ss0156})
112                     initstate=$OPTARG
113                     ;;
114                 [234abcqQ])
115                     echo "$0: Initstate $OPTARG is not for system shutdown"
116                     exit 1
117                     ;;
118                 *)
119                     echo "$0: $OPTARG is not a valid initstate"
120                     usage
121                     ;;
122             esac
123             ;;
124         y)
125             askconfirmation=
126             ;;
127         \?)
128             usage
129             ;;
130     esac
131 done
132 shift `expr $OPTIND - 1`
133 echo '\nShutdown started. \c'
134 /usr/bin/date
135 echo
```

```

137 NODENAME=`uname -n`
142 /sbin/sync
139 cd /
141 trap "rm $nologin >/dev/null 2>&1 ;exit 1" 1 2 15
143 # If other users are on the system (and any grace period is given), warn them.
145 for i in 7200 3600 1800 1200 600 300 120 60 30 10; do
146     if [ ${grace} -gt $i ]
147     then
148         hours=`/usr/bin/expr ${grace} / 3600`
149         minutes=`/usr/bin/expr ${grace} % 3600 / 60`
150         seconds=`/usr/bin/expr ${grace} % 60`
151         time=""
152         if [ ${hours} -gt 1 ]
153         then
154             time="${hours} hours "
155         elif [ ${hours} -eq 1 ]
156         then
157             time="1 hour "
158         fi
159         if [ ${minutes} -gt 1 ]
160         then
161             time="${time}${minutes} minutes "
162         elif [ ${minutes} -eq 1 ]
163         then
164             time="${time}1 minute "
165         fi
166         if [ ${hours} -eq 0 -a ${seconds} -gt 0 ]
167         then
168             if [ ${seconds} -eq 1 ]
169             then
170                 time="${time}${seconds} second"
171             else
172                 time="${time}${seconds} seconds"
173             fi
174         fi
175
176         (notify \
177 "The system ${NODENAME} will be shut down in ${time}
178 $*" ) &
180 pid1=$!
182         rm $nologin >/dev/null 2>&1
183         cat > $nologin <<-!
185         NO LOGINS: System going down in ${time}
186         $*
188         !
190         /usr/bin/sleep ` /usr/bin/expr ${grace} - $i `
191         grace=$i
192     fi
193 done
195 # Confirm that we really want to shutdown.
197 if [ ${askconfirmation} ]
198 then
199     echo "Do you want to continue? (y or n): \c"
200     read b

```

```

201     if [ "$b" != "y" ]
202     then
203         notify "False Alarm: The system ${NODENAME} will not be brought
204         echo 'Shutdown aborted.'
205         rm $nologin >/dev/null 2>&1
206         exit 1
207     fi
208 fi
210 # Final shutdown message, and sleep away the final 10 seconds (or less).
212 (notify \
213 "THE SYSTEM ${NODENAME} IS BEING SHUT DOWN NOW ! ! !
214 Log off now or risk your files being damaged
215 $*" ) &
217 pid2=$!
219 if [ ${grace} -gt 0 ]
220 then
221     /usr/bin/sleep ${grace}
222 fi
224 # Go to the requested initstate.
227 echo "Changing to init state $initstate - please wait"
229 if [ "$pid1" ] || [ "$pid2" ]
230 then
231     /usr/bin/kill $pid1 $pid2 > /dev/null 2>&1
232 fi
234 /sbin/init ${initstate}

```

new/usr/src/cmd/initpkg/umountall.sh

1

```
*****
9350 Thu Feb 26 16:21:29 2009
new/usr/src/cmd/initpkg/umountall.sh
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
1 #!/sbin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
24 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
25 # Use is subject to license terms.
26 #
27 # Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T
28 # All Rights Reserved
29 #
31 usage () {
32     if [ -n "$1" ]; then
33         echo "umountall: $1" 1>&2
34     fi
35     echo "Usage:\n\tumountall [-k] [-s] [-F FSType] [-l|-r] [-Z] [-n]" 1>&2
36     echo "\tumountall [-k] [-s] [-h host] [-Z] [-n]" 1>&2
37     exit 2
38 }
40 MNTTAB=/etc/mnttab
42 # This script is installed as both /sbin/umountall (as used in some
43 # /sbin/rc? and /etc/init.d scripts) _and_ as /usr/sbin/umountall (typically
44 # PATHed from the command line). As such it should not depend on /usr
45 # being mounted (if /usr is a separate filesystem).
46 #
47 # /sbin/sh Bourne shell builtins we use:
48 #     echo
49 #     exit
50 #     getopt
51 #     test, [ ]
52 #     exec
53 #     read
54 #
55 # /sbin commands we use:
56 #     /sbin/uname
57 #     /sbin/umount
59 # The following /usr based commands may be used by this script (depending on
```

new/usr/src/cmd/initpkg/umountall.sh

2

```
60 # command line options). We will set our PATH to find them, but where they
61 # are not present (eg, if /usr is not mounted) we will catch references to
62 # them via shell functions conditionally defined after option processing
63 # (don't use any of these commands before then).
64 #
65 # Command Command line option and use
66 # /usr/bin/sleep -k, to sleep after an fuser -k on the mountpoint
67 # /usr/sbin/fuser -k, to kill processes keeping a mount point busy
68 #
69 # In addition, we use /usr/bin/tail if it is available; if not we use
70 # slower shell constructs to reverse a file.
72 PATH=/sbin:/usr/sbin:/usr/bin
74 DEFERRED_ACTIVATION_PATCH_FLAG="/var/run/.patch_loopback_mode"
75 SVC_STARTD="/lib/svc/bin/svc.startd"
77 # Clear these in case they were already set in our inherited environment.
78 FSType=
79 FFLAG=
80 HOST=
81 HFLAG=
82 RFLAG=
83 LFLAG=
84 SFLAG=
85 KFLAG=
86 ZFLAG=
87 NFLAG=
88 LOCALNAME=
89 UMOUNTFLAG=
92 while getoptrslh:Zn c
93 do
94     case $c in
95         r) RFLAG="r";;
96         l) LFLAG="l";;
97         s) SFLAG="s";;
98         k) KFLAG="k";;
99         h) if [ -n "$HFLAG" ]; then
100             usage "more than one host specified"
101         fi
102         HOST=$OPTARG
103         HFLAG="h"
104         LOCALNAME='uname -n'
105         ;;
106         F) if [ -n "$FFLAG" ]; then
107             usage "more than one FSType specified"
108         fi
109         FSType=$OPTARG
110         FFLAG="F"
111         case $FSType in
112             ???????*)
113                 usage "FSType ${FSType} exceeds 8 characters"
114             ;;
115             Z) ZFLAG="z";;
116             n) NFLAG="n"
117                 # Alias any commands that would perform real actions to
118                 # something that tells what action would have been performed
119                 UMOUNTFLAG="-v"
120                 fuser () {
121                     echo "fuser $" 1>&2
122                 }
123                 sleep () {
124                     : # No need to show where we'd sleep
125                 }
126             ;;
127         *)
128             ;;
129     esac
130 done
```

```

126     }
127     ;;
128     \?) usage ""
129     ;;
130     esac
131 done

133 # Sanity checking:
134 # 1) arguments beyond those supported
135 # 2) can't specify both remote and local
136 # 3) can't specify a host with -r or -l
137 # 4) can't specify a fstype with -h
138 # 5) can't specify this host with -h (checks only uname -n)
139 # 6) can't be fstype nfs and local
140 # 7) only fstype nfs is remote

142 if [ $# -ge $OPTIND ]; then # 1
143     usage "additional arguments not supported"
144 fi

146 if [ -n "$RFLAG" -a -n "$LFLAG" ]; then # 2
147     usage "options -r and -l are incompatible"
148 fi

150 if [ \(\ -n "$RFLAG" -o -n "$LFLAG" \) -a "$HFLAG" = "h" ]; then # 3
151     usage "option -${RFLAG}${LFLAG} incompatible with -h option"
152 fi

154 if [ -n "$FFLAG" -a "$HFLAG" = "h" ]; then # 4
155     usage "Specifying FStype incompatible with -h option"
156 fi

158 if [ -n "$HFLAG" -a "$HOST" = "$LOCALNAME" ]; then # 5
159     usage "Specifying local host illegal for -h option"
160 fi

162 if [ "$FSType" = "nfs" -a "$LFLAG" = "l" ]; then # 6
163     usage "option -l and FStype nfs are incompatible"
164 fi

166 if [ -n "$FFLAG" -a "$FSType" != "nfs" -a -n "$RFLAG" ]; then # 7
167     usage "option -r and FStype ${FSType} are incompatible"
168 fi

170 ZONENAME='zonename'

172 # Check and if needed sync the boot archive before unmounting everything.
172 #
174 if [ -z "${RFLAG}${NFLAG}${HFLAG}${FSType}" -a "$ZONENAME" = "global" \
175     -a -x /sbin/bootadm ] ; then
176     /sbin/bootadm -a update_all
177 fi

180 #
173 # If we are in deferred activation patching, and the caller is
174 # svc.startd, then exit without unmounting any of the remaining
175 # file systems since the call path is from shutdown. Note that
176 # by the time we get here, smf stop methods for nfs, cachefs
177 # etc, will have run.
178 #
179 if [ -f $DEFERRED_ACTIVATION_PATCH_FLAG ] ; then
180     ppid='ps -o ppid= -p $$' # parent of umountall will be sh
181                             # from system()
183     ppid='ps -o ppid= -p $ppid' # parent of sh will be svc.startd

```

```

184     COMM='ps -o comm= -p $ppid'
185     if [ "$COMM" = "$SVC_STARTD" ] ; then
186         exit
187     fi
188 fi

190 #
191 # Take advantage of parallel unmounting at this point if we have no
192 # criteria to match and we are in the global zone
193 #
194 if [ -z "${SFLAG}${LFLAG}${RFLAG}${HFLAG}${KFLAG}${FFLAG}${ZFLAG}" -a \
195     "$ZONENAME" = "global" ]; then
196     umount -a ${UMOUNTFLAG}
197     exit # with return code of the umount -a
198 fi

200 #
201 # Catch uses of /usr commands when /usr is not mounted
202 if [ -n "$KFLAG" -a -z "$NFLAG" ]; then
203     if [ ! -x /usr/sbin/fuser ]; then
204         fuser () {
205             echo "umountall: fuser -k skipped (no /usr)" 1>&2
206             # continue - not fatal
207         }
208         sleep () {
209             : # no point in sleeping if fuser is doing nothing
210         }
211     else
212         if [ ! -x /usr/bin/sleep ]; then
213             sleep () {
214                 echo "umountall: sleep after fuser -k skipped (
215                 # continue - not fatal
216             }
217         fi
218     fi
219 fi

221 #
222 # Shell function to avoid using /usr/bin/cut. Given a dev from a
223 # fstype=nfs line in mnttab (eg, "host:/export) extract the host
224 # component.
225 print_host () {
226     OIFS=$IFS
227     IFS=":"
228     set -- $*
229     echo $1
230     IFS=$OIFS
231 }

```

unchanged_portion_omitted

new/usr/src/cmd/svc/shell/smf_include.sh

1

```
*****
6354 Thu Feb 26 16:21:31 2009
new/usr/src/cmd/svc/shell/smf_include.sh
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
```

```
1 #!/bin/sh
2 #
3 # CDDL HEADER START
4 #
5 # The contents of this file are subject to the terms of the
6 # Common Development and Distribution License (the "License").
7 # You may not use this file except in compliance with the License.
8 #
9 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
10 # or http://www.opensolaris.org/os/licensing.
11 # See the License for the specific language governing permissions
12 # and limitations under the License.
13 #
14 # When distributing Covered Code, include this CDDL HEADER in each
15 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
16 # If applicable, add the following below this CDDL HEADER, with the
17 # fields enclosed by brackets "[]" replaced with your own identifying
18 # information: Portions Copyright [yyyy] [name of copyright owner]
19 #
20 # CDDL HEADER END
21 #
22 #
23 # Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 # Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 # Use is subject to license terms.
25 #
```

```
27 smf_present () {
28     [ -r /etc/svc/volatile/repository_door ] && \
29     [ ! -f /etc/svc/volatile/repository_door ]
30 }
```

unchanged_portion_omitted_

```
145 #
146 # smf_kill_contract CONTRACT SIGNAL WAIT TIMEOUT
147 #
148 # To be called from stop methods of non-transient services.
149 # Sends SIGNAL to the service contract CONTRACT. If the
150 # WAIT argument is non-zero, smf_kill_contract will wait
151 # until the contract is empty before returning, or until
152 # TIMEOUT expires.
153 #
154 # Example, send SIGTERM to contract 200:
155 #
156 #     smf_kill_contract 200 TERM
157 #
158 # Since killing a contract with pkill(1) is not atomic,
159 # smf_kill_contract will continue to send SIGNAL to CONTRACT
160 # every second until the contract is empty. This will catch
161 # races between fork(2) and pkill(1).
162 #
163 # Note that time in this routine is tracked (after being input
164 # via TIMEOUT) in 10ths of a second. This is because we want
165 # to sleep for short periods of time, and expr(1) is too dumb
166 # to do non-integer math.
167 #
168 # Returns 1 if the contract is invalid.
169 # Returns 2 if WAIT is "1", TIMEOUT is > 0, and TIMEOUT expires.
170 # Returns 0 on success.
171 #
```

new/usr/src/cmd/svc/shell/smf_include.sh

2

```
172 smf_kill_contract() {
174     time_waited=0
175     time_to_wait=$4
177     [ -z "$time_to_wait" ] && time_to_wait=0
179     # convert to 10ths.
180     time_to_wait='expr $time_to_wait * 10\'
182     # Verify contract id is valid using pgrep
183     /usr/bin/pgrep -c $1 > /dev/null 2>&1
184     ret=$?
185     if [ $ret -gt 1 ] ; then
186         echo "Error, invalid contract \"$1\" " >&2
187         return 1
188     fi
190     # Return if contract is already empty.
191     [ $ret -eq 1 ] && return 0
193     # Kill contract.
194     /usr/bin/pkill -$2 -c $1
195     if [ $? -gt 1 ] ; then
196         echo "Error, could not kill contract \"$1\" " >&2
197         return 1
198     fi
200     # Return if WAIT is not set or is "0"
201     [ -z "$3" ] && return 0
202     [ "$3" -eq 0 ] && return 0
203
204     # If contract does not empty, keep killing the contract to catch
205     # any child processes missed because they were forking
198     /usr/bin/sleep 5
206     /usr/bin/pgrep -c $1 > /dev/null 2>&1
207     while [ $? -eq 0 ] ; do
208         # Return 2 if TIMEOUT was passed, and it has expired
209
210         time_waited='/usr/bin/expr $time_waited + 5'
212
213         # Return if TIMEOUT was passed, and it has expired
209     [ "$time_to_wait" -gt 0 -a $time_waited -ge $time_to_wait ] && \
210         return 2
212     #
213     # At five second intervals, issue the kill again. Note that
214     # the sleep time constant (in tenths) must be a factor of 50
215     # for the remainder trick to work. i.e. sleeping 2 tenths is
216     # fine, but 27 tenths is not.
217     #
218     remainder='/usr/bin/expr $time_waited % 50\'
219     if [ $time_waited -gt 0 -a $remainder -eq 0 ] ; then
220         /usr/bin/pkill -$2 -c $1
221     fi
223     # Wait two tenths, and go again.
224     /usr/bin/sleep 0.2
225     time_waited='/usr/bin/expr $time_waited + 2\'
208     /usr/bin/sleep 5
226     /usr/bin/pgrep -c $1 > /dev/null 2>&1
227     done
229     return 0
230 }
```

unchanged_portion_omitted_

```

*****
19270 Thu Feb 26 16:21:32 2009
new/usr/src/cmd/svc/started/fork.c
6805730 some simple changes would make 'init 5' much faster
6809492 started shouldn't let hung subprocesses impede shutdown
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24 * Use is subject to license terms.
25 */

26 /*
27 * fork.c - safe forking for svc.started
28 *
29 * fork_configd() and fork_sulogin() are related, special cases that handle the
30 * spawning of specific client processes for svc.started.
31 */

33 #include <sys/contract/process.h>
34 #include <sys/corectl.h>
35 #include <sys/ctfs.h>
36 #include <sys/stat.h>
37 #include <sys/types.h>
38 #include <sys/uio.h>
39 #include <sys/wait.h>
40 #include <assert.h>
41 #include <errno.h>
42 #include <fcntl.h>
43 #include <libcontract.h>
44 #include <libcontract_priv.h>
45 #include <libscf_priv.h>
46 #include <limits.h>
47 #include <poll.h>
48 #include <port.h>
49 #include <signal.h>
50 #include <stdarg.h>
51 #include <stdio.h>
52 #include <stdlib.h>
53 #include <string.h>
54 #include <unistd.h>
55 #include <utmpx.h>
56 #include <spawn.h>

58 #include "configd_exit.h"
59 #include "protocol.h"

```

```

60 #include "started.h"

62 static struct utmpx *utmp; /* pointer for getutxent() */

64 pid_t
65 started_forkl(int *forkerr)
66 {
67     pid_t p;

69     /*
70      * prefork stack
71      */
72     wait_prefork();

74     p = forkl();

76     if (p == -1 && forkerr != NULL)
77         *forkerr = errno;

79     /*
80      * postfork stack
81      */
82     wait_postfork(p);

84     return (p);
85 }
_____ unchanged_portion_omitted

600 void
601 fork_rc_script(char rl, const char *arg, boolean_t wait)
602 {
603     pid_t pid;
604     int tmpl, err, stat;
605     char path[20] = "/sbin/rc.", log[20] = "rc..log", timebuf[20];
606     time_t now;
607     struct tm ltime;
608     size_t sz;
609     char *pathenv;
610     char **nenv;

612     path[8] = rl;

614     tmpl = open64(CTFS_ROOT "/process/template", O_RDWR);
615     if (tmpl >= 0) {
616         err = ct_tmpl_set_critical(tmpl, 0);
617         assert(err == 0);

619         err = ct_tmpl_set_informative(tmpl, 0);
620         assert(err == 0);

622         err = ct_pr_tmpl_set_fatal(tmpl, 0);
623         assert(err == 0);

625         err = ct_tmpl_activate(tmpl);
626         assert(err == 0);

628         err = close(tmpl);
629         assert(err == 0);
630     } else {
631         uu_warn("Could not create contract template for %s.\n", path);
632     }

634     pid = started_forkl(NULL);
635     if (pid < 0) {
636         return;
637     } else if (pid != 0) {

```

```

638     /* parent */
639     if (wait) {
640         do
641             err = waitpid(pid, &stat, 0);
642         while (err != 0 && errno == EINTR)
643             ;
644
645         if (!WIFEXITED(stat)) {
646             log_framework(LOG_INFO,
647                 "%s terminated with waitpid() status %d.\n",
648                 path, stat);
649         } else if (WEXITSTATUS(stat) != 0) {
650             log_framework(LOG_INFO,
651                 "%s failed with status %d.\n", path,
652                 WEXITSTATUS(stat));
653         }
654     }
655
656     return;
657 }
658
659 /* child */
660
661 log[2] = rl;
662
663 setlog(log);
664
665 now = time(NULL);
666 sz = strftime(timebuf, sizeof (timebuf), "%b %e %T",
667     localtime_r(&now, &lttime));
668 assert(sz != 0);
669
670 (void) fprintf(stderr, "%s Executing %s %s\n", timebuf, path, arg);
671
672 if (rl == 'S')
673     pathenv = "PATH=/sbin:/usr/sbin:/usr/bin";
674 else
675     pathenv = "PATH=/usr/sbin:/usr/bin";
676
677 nenv = set_smf_env(NULL, 0, pathenv, NULL, NULL);
678
679 (void) execl(path, path, arg, 0, nenv);
680
681 perror("exec");
682 exit(0);
683 }
684
685 extern char **environ;
686
687 /*
688 * A local variation on system(3c) which accepts a timeout argument. This
689 * allows us to better ensure that the system will actually shut down.
690 *
691 * gracetime specifies an amount of time in seconds which the routine must wait
692 * after the command exits, to allow for asynchronous effects (like sent
693 * signals) to take effect. This can be zero.
694 */
695 void
696 fork_with_timeout(const char *cmd, uint_t gracetime, uint_t timeout)
697 {
698     int err = 0;
699     pid_t pid;
700     char *argv[4];
701     posix_spawnattr_t attr;
702     posix_spawn_file_actions_t factions;

```

```

704     sigset_t mask, savemask;
705     uint_t msec_timeout;
706     uint_t msec_spent = 0;
707     uint_t msec_gracetime;
708     int status;
709
710     msec_timeout = timeout * 1000;
711     msec_gracetime = gracetime * 1000;
712
713     /*
714     * See also system(3c) in libc. This is very similar, except
715     * that we avoid some unneeded complexity.
716     */
717     err = posix_spawnattr_init(&attr);
718     if (err == 0)
719         err = posix_spawnattr_setflags(&attr,
720             POSIX_SPAWN_SETSIGMASK | POSIX_SPAWN_SETSIGDEF |
721             POSIX_SPAWN_NOSIGCHLD_NP | POSIX_SPAWN_WAITPID_NP |
722             POSIX_SPAWN_NOEXECERR_NP);
723
724     /*
725     * We choose to close fd's above 2, a deviation from system.
726     */
727     if (err == 0)
728         err = posix_spawn_file_actions_init(&factions);
729     if (err == 0)
730         err = posix_spawn_file_actions_addclosefrom_np(&factions,
731             STDERR_FILENO + 1);
732
733     (void) sigemptyset(&mask);
734     (void) sigaddset(&mask, SIGCHLD);
735     (void) thr_sigsetmask(SIG_BLOCK, &mask, &savemask);
736
737     argv[0] = "/bin/sh";
738     argv[1] = "-c";
739     argv[2] = (char *)cmd;
740     argv[3] = NULL;
741
742     if (err == 0)
743         err = posix_spawn(&pid, "/bin/sh", &factions, &attr,
744             (char *const *)argv, (char *const *)environ);
745
746     (void) posix_spawnattr_destroy(&attr);
747     (void) posix_spawn_file_actions_destroy(&factions);
748
749     if (err) {
750         uu_warn("Failed to spawn %s: %s\n", cmd, strerror(err));
751     } else {
752         for (;;) {
753             int w;
754             w = waitpid(pid, &status, WNOHANG);
755             if (w == -1 && errno != EINTR)
756                 break;
757             if (w > 0) {
758                 /*
759                 * Command succeeded, so give it gracetime
760                 * seconds for it to have an effect.
761                 */
762                 if (status == 0 && msec_gracetime != 0)
763                     (void) poll(NULL, 0, msec_gracetime);
764                 break;
765             }
766
767             (void) poll(NULL, 0, 100);
768             msec_spent += 100;
769             /*

```

```
770     * If we timed out, kill off the process, then try to
771     * wait for it-- it's possible that we could accumulate
772     * a zombie here since we don't allow waitpid to hang,
773     * but it's better to let that happen and continue to
774     * make progress.
775     */
776     if (msec_spent >= msec_timeout) {
777         uu_warn("%s' timed out after %d "
778             "seconds. Killing.\n", cmd,
779             timeout);
780         (void) kill(pid, SIGTERM);
781         (void) poll(NULL, 0, 100);
782         (void) kill(pid, SIGKILL);
783         (void) poll(NULL, 0, 100);
784         (void) waitpid(pid, &status, WNOHANG);
785         break;
786     }
787 }
788 }
789 (void) thr_sigsetmask(SIG_BLOCK, &savemask, NULL);
790 }
```

_____unchanged_portion_omitted_____

```

*****
161764 Thu Feb 26 16:21:33 2009
new/usr/src/cmd/svc/startd/graph.c
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23  * Use is subject to license terms.
24 */

26 /*
27  * graph.c - master restarters graph engine
28  *
29  * The graph engine keeps a dependency graph of all service instances on the
30  * system, as recorded in the repository. It decides when services should
31  * be brought up or down based on service states and dependencies and sends
32  * commands to restarters to effect any changes. It also executes
33  * administrator commands sent by svcadm via the repository.
34  *
35  * The graph is stored in uu_list_t *dgraph and its vertices are
36  * graph_vertex_t's, each of which has a name and an integer id unique to
37  * its name (see dict.c). A vertex's type attribute designates the type
38  * of object it represents: GVT_INST for service instances, GVT_SVC for
39  * service objects (since service instances may depend on another service,
40  * rather than service instance), GVT_FILE for files (which services may
41  * depend on), and GVT_GROUP for dependencies on multiple objects. GVT_GROUP
42  * vertices are necessary because dependency lists may have particular
43  * grouping types (require any, require all, optional, or exclude) and
44  * event-propagation characteristics.
45  *
46  * The initial graph is built by libscf_populate_graph() invoking
47  * dgraph_add_instance() for each instance in the repository. The function
48  * adds a GVT_SVC vertex for the service if one does not already exist, adds
49  * a GVT_INST vertex named by the FMRI of the instance, and sets up the edges.
50  * The resulting web of vertices & edges associated with an instance's vertex
51  * includes
52  *
53  * - an edge from the GVT_SVC vertex for the instance's service
54  *
55  * - an edge to the GVT_INST vertex of the instance's resarter, if its
56  *   restarters is not svc.startd
57  *
58  * - edges from other GVT_INST vertices if the instance is a restarters
59  *
60  * - for each dependency property group in the instance's "running"

```

```

61  * snapshot, an edge to a GVT_GROUP vertex named by the FMRI of the
62  * instance and the name of the property group
63  *
64  * - for each value of the "entities" property in each dependency property
65  *   group, an edge from the corresponding GVT_GROUP vertex to a
66  *   GVT_INST, GVT_SVC, or GVT_FILE vertex
67  *
68  * - edges from GVT_GROUP vertices for each dependent instance
69  *
70  * After the edges are set up the vertex's GV_CONFIGURED flag is set. If
71  * there are problems, or if a service is mentioned in a dependency but does
72  * not exist in the repository, the GV_CONFIGURED flag will be clear.
73  *
74  * The graph and all of its vertices are protected by the dgraph_lock mutex.
75  * See restarters.c for more information.
76  *
77  * The properties of an instance fall into two classes: immediate and
78  * snapshot. Immediate properties should have an immediate effect when
79  * changed. Snapshot properties should be read from a snapshot, so they
80  * only change when the snapshot changes. The immediate properties used by
81  * the graph engine are general/enabled, general/restarters, and the properties
82  * in the restarters_actions property group. Since they are immediate, they
83  * are not read out of a snapshot. The snapshot properties used by the
84  * graph engine are those in the property groups with type "dependency" and
85  * are read out of the "running" snapshot. The "running" snapshot is created
86  * by the the graph engine as soon as possible, and it is updated, along with
87  * in-core copies of the data (dependency information for the graph engine) on
88  * receipt of the refresh command from svcadm. In addition, the graph engine
89  * updates the "start" snapshot from the "running" snapshot whenever a service
90  * comes online.
91  *
92  * When a DISABLE event is requested by the administrator, svc.startd shutdown
93  * the dependents first before shutting down the requested service.
94  * In graph_enable_by_vertex, we create a subtree that contains the dependent
95  * vertices by marking those vertices with the GV_TOOFFLINE flag. And we mark
96  * the vertex to disable with the GV_TODISABLE flag. Once the tree is created,
97  * we send the _ADMIN_DISABLE event to the leaves. The leaves will then
98  * transition from STATE_ONLINE/STATE_DEGRADED to STATE_OFFLINE/STATE_MAINT.
99  * In gt_enter_offline and gt_enter_maint if the vertex was in a subtree then
100 * we clear the GV_TOOFFLINE flag and walk the dependencies to offline the new
101 * exposed leaves. We do the same until we reach the last leaf (the one with
102 * the GV_TODISABLE flag). If the vertex to disable is also part of a larger
103 * subtree (eg. multiple DISABLE events on vertices in the same subtree) then
104 * once the first vertex is disabled (GV_TODISABLE flag is removed), we
105 * continue to propagate the offline event to the vertex's dependencies.
106 */

108 #include <sys/uadmin.h>
109 #include <sys/wait.h>

111 #include <assert.h>
112 #include <errno.h>
113 #include <fcntl.h>
114 #include <libscf.h>
115 #include <libscf_priv.h>
116 #include <libuutil.h>
117 #include <locale.h>
118 #include <poll.h>
119 #include <pthread.h>
120 #include <signal.h>
121 #include <stddef.h>
122 #include <stdio.h>
123 #include <stdlib.h>
124 #include <string.h>
125 #include <strings.h>
126 #include <sys/statvfs.h>

```

new/usr/src/cmd/svc/started/graph.c

3

```
127 #include <sys/uadmin.h>
128 #include <zone.h>

130 #include "started.h"
131 #include "protocol.h"

134 #define MILESTONE_NONE ((graph_vertex_t *)1)

136 #define CONSOLE_LOGIN_FMRI "svc:/system/console-login:default"
137 #define FS_MINIMAL_FMRI "svc:/system/filesystem/minimal:default"

139 #define VERTEX_REMOVED 0 /* vertex has been freed */
140 #define VERTEX_INUSE 1 /* vertex is still in use */

142 /*
143 * Services in these states are not considered 'down' by the
144 * milestone/shutdown code.
145 */
146 #define up_state(state) ((state) == RESTARTER_STATE_ONLINE || \
147 (state) == RESTARTER_STATE_DEGRADED || \
148 (state) == RESTARTER_STATE_OFFLINE)

150 static uu_list_pool_t *graph_edge_pool, *graph_vertex_pool;
151 static uu_list_t *dgraph;
152 static pthread_mutex_t dgraph_lock;

154 /*
155 * milestone indicates the current subgraph. When NULL, it is the entire
156 * graph. When MILESTONE_NONE, it is the empty graph. Otherwise, it is all
157 * services on which the target vertex depends.
158 */
159 static graph_vertex_t *milestone = NULL;
160 static boolean_t initial_milestone_set = B_FALSE;
161 static pthread_cond_t initial_milestone_cv = PTHREAD_COND_INITIALIZER;

163 /* protected by dgraph_lock */
164 static boolean_t sulogin_thread_running = B_FALSE;
165 static boolean_t sulogin_running = B_FALSE;
166 static boolean_t console_login_ready = B_FALSE;

168 /* Number of services to come down to complete milestone transition. */
169 static uint_t non_subgraph_svcs;

171 /*
172 * These variables indicate what should be done when we reach the milestone
173 * target milestone, i.e., when non_subgraph_svcs == 0. They are acted upon in
174 * dgraph_set_instance_state().
175 */
176 static int halting = -1;
177 static boolean_t go_single_user_mode = B_FALSE;
178 static boolean_t go_to_level1 = B_FALSE;

180 /*
181 * Tracks when we started halting.
182 */
183 static time_t halting_time = 0;

185 /*
186 * This tracks the legacy runlevel to ensure we signal init and manage
187 * utmpx entries correctly.
188 */
189 static char current_runlevel = '\0';

191 /* Number of single user threads currently running */
192 static pthread_mutex_t single_user_thread_lock;
```

new/usr/src/cmd/svc/started/graph.c

4

```
193 static int single_user_thread_count = 0;

195 /* Statistics for dependency cycle-checking */
196 static u_longlong_t dep_inserts = 0;
197 static u_longlong_t dep_cycle_ns = 0;
198 static u_longlong_t dep_insert_ns = 0;

201 static const char * const emsg_invalid_restarter =
202 "Transitioning %s to maintenance, restarter FMRI %s is invalid "
203 "(see 'svcs -xv' for details).\n";
204 static const char * const console_login_fmri = CONSOLE_LOGIN_FMRI;
205 static const char * const single_user_fmri = SCF_MILESTONE_SINGLE_USER;
206 static const char * const multi_user_fmri = SCF_MILESTONE_MULTI_USER;
207 static const char * const multi_user_svr_fmri = SCF_MILESTONE_MULTI_USER_SERVER;

210 /*
211 * These services define the system being "up". If none of them can come
212 * online, then we will run sulogin on the console. Note that the install ones
213 * are for the miniroot and when installing CDs after the first. can_come_up()
214 * does the decision making, and an sulogin_thread() runs sulogin, which can be
215 * started by dgraph_set_instance_state() or single_user_thread().
216 *
217 * NOTE: can_come_up() relies on SCF_MILESTONE_SINGLE_USER being the first
218 * entry, which is only used when booting_to_single_user (boot -s) is set.
219 * This is because when doing a "boot -s", sulogin is started from specials.c
220 * after milestone/single-user comes online, for backwards compatibility.
221 * In this case, SCF_MILESTONE_SINGLE_USER needs to be part of up_svcs
222 * to ensure sulogin will be spawned if milestone/single-user cannot be reached.
223 */
224 static const char * const up_svcs[] = {
225 SCF_MILESTONE_SINGLE_USER,
226 CONSOLE_LOGIN_FMRI,
227 "svc:/system/install-setup:default",
228 "svc:/system/install:default",
229 NULL
230 };
_____unchanged_portion_omitted_____

3450 static void
3451 kill_user_procs(void)
3452 {
3453 (void) fputs("svc.started: Killing user processes.\n", stdout);

3455 /*
3456 * Despite its name, killall's role is to get select user processes--
3457 * basically those representing terminal-based logins-- to die. Victims
3458 * are located by killall in the utmp database. Since these are most
3459 * often shell based logins, and many shells mask SIGTERM (but are
3460 * responsive to SIGHUP) we first HUP and then shortly thereafter
3461 * kill -9.
3462 */
3463 (void) fork_with_timeout("/usr/sbin/killall HUP", 1, 5);
3464 (void) fork_with_timeout("/usr/sbin/killall KILL", 1, 5);

3466 /*
3467 * Note the selection of user id's 0, 1 and 15, subsequently
3468 * inverted by -v. 15 is reserved for dladmd. Yes, this is a
3469 * kludge-- a better policy is needed.
3470 *
3471 * Note that fork_with_timeout will only wait out the 1 second
3472 * "grace time" if pkill actually returns 0. So if there are
3473 * no matches, this will run to completion much more quickly.
3474 */
```

```

3475     (void) fork_with_timeout("/usr/bin/pkill -TERM -v -u 0,1,15", 1, 5);
3476     (void) fork_with_timeout("/usr/bin/pkill -KILL -v -u 0,1,15", 1, 5);
3477 }

3479 static void
3480 do_uadmin(void)
3481 {
3482     int fd;
3483     int fd, left;
3484     struct statvfs vfs;
3485     time_t now;
3486     struct tm nowtm;
3487     char down_buf[256], time_buf[256];

3488     const char * const resetting = "/etc/svc/volatile/resetting";

3490     fd = creat(resetting, 0777);
3491     if (fd >= 0)
3492         startd_close(fd);
3493     else
3494         uu_warn("Could not create \"%s\"", resetting);

3496     /* Kill dhcpgent if we're not using nfs for root */
3497     if ((statvfs("/", &vfs) == 0) &&
3498         (strncmp(vfs.f_basetype, "nfs", sizeof("nfs") - 1) != 0))
3499         fork_with_timeout("/usr/bin/pkill -x -u 0 dhcpgent", 0, 5);
3500     (void) system("/usr/bin/pkill -x -u 0 dhcpgent");

3501     /*
3502     * Call sync(2) now, before we kill off user processes. This takes
3503     * advantage of the several seconds of pause we have before the
3504     * killalls are done. Time we can make good use of to get pages
3505     * moving out to disk.
3506     *
3507     * Inside non-global zones, we don't bother, and it's better not to
3508     * anyway, since sync(2) can have system-wide impact.
3509     */
3510     if (getzoneid() == 0)
3511         sync();
3512     (void) system("/usr/sbin/killall");
3513     left = 5;
3514     while (left > 0)
3515         left = sleep(left);

3516     kill_user_procs();
3517     (void) system("/usr/sbin/killall 9");
3518     left = 10;
3519     while (left > 0)
3520         left = sleep(left);

3521     /*
3522     * Note that this must come after the killing of user procs, since
3523     * killall relies on utmpx, and this command affects the contents of
3524     * said file.
3525     */
3526     if (access("/usr/lib/acct/closewtmp", X_OK) == 0)
3527         fork_with_timeout("/usr/lib/acct/closewtmp", 0, 5);
3528     sync();
3529     sync();
3530     sync();

3531     /*
3532     * For patches which may be installed as the system is shutting
3533     * down, we need to ensure, one more time, that the boot archive
3534     * really is up to date.
3535     */

```

```

3528     if (getzoneid() == 0 && access("/usr/sbin/bootadm", X_OK) == 0)
3529         fork_with_timeout("/usr/sbin/bootadm -ea update_all", 0, 3600);
3530     (void) system("/sbin/umountall -l");
3531     (void) system("/sbin/umount /tmp >/dev/null 2>&1");
3532     (void) system("/sbin/umount /var/adm >/dev/null 2>&1");
3533     (void) system("/sbin/umount /var/run >/dev/null 2>&1");
3534     (void) system("/sbin/umount /var >/dev/null 2>&1");
3535     (void) system("/sbin/umount /usr >/dev/null 2>&1");

3536     fork_with_timeout("/sbin/umountall -l", 0, 5);
3537     fork_with_timeout("/sbin/umount /tmp /var/adm /var/run /var "
3538         ">/dev/null 2>&1", 0, 5);
3539     uu_warn("The system is down.\n");

3540     /*
3541     * Try to get to consistency for whatever UFS filesystems are left.
3542     * This is pretty expensive, so we save it for the end in the hopes of
3543     * minimizing what it must do. The other option would be to start in
3544     * parallel with the killall's, but lockfs tends to throw out much more
3545     * than is needed, and so subsequent commands (like umountall) take a
3546     * long time to get going again.
3547     *
3548     * Inside of zones, we don't bother, since we're not about to terminate
3549     * the whole OS instance.
3550     *
3551     * On systems using only ZFS, this call to lockfs -fa is a no-op.
3552     */
3553     if (getzoneid() == 0) {
3554         if (access("/usr/sbin/lockfs", X_OK) == 0)
3555             fork_with_timeout("/usr/sbin/lockfs -fa", 0, 30);

3556         sync(); /* once more, with feeling */
3557     }

3558     fork_with_timeout("/sbin/umount /usr >/dev/null 2>&1", 0, 5);

3559     /*
3560     * Construct and emit the last words from userland:
3561     * "<timestamp> The system is down. Shutdown took <N> seconds."
3562     *
3563     * Normally we'd use syslog, but with /var and other things
3564     * potentially gone, try to minimize the external dependencies.
3565     */
3566     now = time(NULL);
3567     (void) localtime_r(&now, &nowtm);

3568     if (strftime(down_buf, sizeof(down_buf),
3569         "%b %e %T The system is down.", &nowtm) == 0) {
3570         (void) strcpy(down_buf, "The system is down.",
3571             sizeof(down_buf));
3572     }

3573     if (halting_time != 0 && halting_time <= now) {
3574         (void) snprintf(time_buf, sizeof(time_buf),
3575             "Shutdown took %lu seconds.", now - halting_time);
3576     } else {
3577         time_buf[0] = '\0';
3578     }
3579     (void) printf("%s\n", down_buf, time_buf);

3580     (void) uadmin(A_SHUTDOWN, halting, NULL);
3581     uu_warn("uadmin() failed");

3582     if (remove(resetting) != 0 && errno != EWOULDBLOCK)
3583         uu_warn("Could not remove \"%s\"", resetting);
3584 }

```

unchanged_portion_omitted

```

3763 /* ARGSUSED */
3764 void *
3765 single_user_thread(void *unused)
3766 {
3767     uint_t left;
3768     scf_handle_t *h;
3769     scf_instance_t *inst;
3770     scf_property_t *prop;
3771     scf_value_t *val;
3772     const char *msg;
3773     char *buf;
3774     int r;

3776     MUTEX_LOCK(&single_user_thread_lock);
3777     single_user_thread_count++;

3779     if (!booting_to_single_user)
3780         kill_user_procs();
3781     if (!booting_to_single_user) {
3782         /*
3783          * From rcS.sh: Look for ttymon, in.telnetd, in.rlogind and
3784          * processes in their process groups so they can be terminated.
3785          */
3786         (void) fputs("svc.startd: Killing user processes: ", stdout);
3787         (void) system("/usr/sbin/killall");
3788         (void) system("/usr/sbin/killall 9");
3789         (void) system("/usr/bin/pkill -TERM -v -u 0,1");
3790     }

3792     left = 5;
3793     while (left > 0)
3794         left = sleep(left);

3796     (void) system("/usr/bin/pkill -KILL -v -u 0,1");
3797     (void) puts("done.");
3798 }

3800 if (go_single_user_mode || booting_to_single_user) {
3801     msg = "SINGLE USER MODE\n";
3802 } else {
3803     assert(go_to_level1);

3805     fork_rc_script('1', "start", B_TRUE);

3807     uu_warn("The system is ready for administration.\n");

3809     msg = "";
3810 }

3812     MUTEX_UNLOCK(&single_user_thread_lock);

3814     for (;;) {
3815         MUTEX_LOCK(&dgraph_lock);
3816         r = run_sulogin(msg);
3817         MUTEX_UNLOCK(&dgraph_lock);
3818         if (r == 0)
3819             break;

3821     assert(r == EALREADY || r == EBUSY);

3823     left = 3;
3824     while (left > 0)
3825         left = sleep(left);
3826 }

3828     MUTEX_LOCK(&single_user_thread_lock);

```

```

3812     /*
3813      * If another single user thread has started, let it finish changing
3814      * the run level.
3815      */
3816     if (single_user_thread_count > 1) {
3817         single_user_thread_count--;
3818         MUTEX_UNLOCK(&single_user_thread_lock);
3819         return (NULL);
3820     }

3822     h = libscf_handle_create_bound_loop();
3823     inst = scf_instance_create(h);
3824     prop = safe_scf_property_create(h);
3825     val = safe_scf_value_create(h);
3826     buf = startd_alloc(max_scf_fmri_size);

3828     lookup:
3829     if (scf_handle_decode_fmri(h, SCF_SERVICE_STARTD, NULL, NULL, inst,
3830         NULL, NULL, SCF_DECODE_FMRI_EXACT) != 0) {
3831         switch (scf_error()) {
3832             case SCF_ERROR_NOT_FOUND:
3833                 r = libscf_create_self(h);
3834                 if (r == 0)
3835                     goto lookup;
3836                 assert(r == ECONNABORTED);
3837                 /* FALLTHROUGH */

3839             case SCF_ERROR_CONNECTION_BROKEN:
3840                 libscf_handle_rebind(h);
3841                 goto lookup;

3843             case SCF_ERROR_INVALID_ARGUMENT:
3844             case SCF_ERROR_CONSTRAINT_VIOLATED:
3845             case SCF_ERROR_NOT_BOUND:
3846             case SCF_ERROR_HANDLE_MISMATCH:
3847             default:
3848                 bad_error("scf_handle_decode_fmri", scf_error());
3849         }
3850     }

3852     MUTEX_LOCK(&dgraph_lock);

3854     r = scf_instance_delete_prop(inst, SCF_PG_OPTIONS_OVR,
3855         SCF_PROPERTY_MILESTONE);
3856     switch (r) {
3857     case 0:
3858     case ECANCELED:
3859         break;

3861     case ECONNABORTED:
3862         MUTEX_UNLOCK(&dgraph_lock);
3863         libscf_handle_rebind(h);
3864         goto lookup;

3866     case EPERM:
3867     case EACCESS:
3868     case EROFS:
3869         log_error(LOG_WARNING, "Could not clear temporary milestone: "
3870             "%s.\n", strerror(r));
3871         break;

3873     default:
3874         bad_error("scf_instance_delete_prop", r);
3875     }

```

```

3877     MUTEX_UNLOCK(&dgraph_lock);

3879     r = libscf_get_milestone(inst, prop, val, buf, max_scf_fmri_size);
3880     switch (r) {
3881     case ECANCELED:
3882     case ENOENT:
3883     case EINVAL:
3884         (void) strcpy(buf, "all");
3885         /* FALLTHROUGH */

3887     case 0:
3888         uu_warn("Returning to milestone %s.\n", buf);
3889         break;

3891     case ECONNABORTED:
3892         libscf_handle_rebind(h);
3893         goto lookup;

3895     default:
3896         bad_error("libscf_get_milestone", r);
3897     }

3899     r = dgraph_set_milestone(buf, h, B_FALSE);
3900     switch (r) {
3901     case 0:
3902     case ECONNRESET:
3903     case EALREADY:
3904     case EINVAL:
3905     case ENOENT:
3906         break;

3908     default:
3909         bad_error("dgraph_set_milestone", r);
3910     }

3912     /*
3913     * See graph_runlevel_changed().
3914     */
3915     MUTEX_LOCK(&dgraph_lock);
3916     utmpx_set_runlevel(target_milestone_as_runlevel(), 'S', B_TRUE);
3917     MUTEX_UNLOCK(&dgraph_lock);

3919     startd_free(buf, max_scf_fmri_size);
3920     scf_value_destroy(val);
3921     scf_property_destroy(prop);
3922     scf_instance_destroy(inst);
3923     scf_handle_destroy(h);

3925     /*
3926     * We'll give ourselves 3 seconds to respond to all of the enablings
3927     * that setting the milestone should have created before checking
3928     * whether to run sulogin.
3929     */
3930     left = 3;
3931     while (left > 0)
3932         left = sleep(left);

3934     MUTEX_LOCK(&dgraph_lock);
3935     /*
3936     * Clearing these variables will allow the sulogin thread to run. We
3937     * check here in case there aren't any more state updates anytime soon.
3938     */
3939     go_to_level1 = go_single_user_mode = booting_to_single_user = B_FALSE;
3940     if (!sulogin_thread_running && !can_come_up()) {
3941         (void) startd_thread_create(sulogin_thread, NULL);
3942         sulogin_thread_running = B_TRUE;

```

```

3943     }
3944     MUTEX_UNLOCK(&dgraph_lock);
3945     single_user_thread_count--;
3946     MUTEX_UNLOCK(&single_user_thread_lock);
3947     return (NULL);
3948 }
    unchanged_portion_omitted

4910 /*
4911 * Move to a backwards-compatible runlevel by executing the appropriate
4912 * /etc/rc?.d/K* scripts and/or setting the milestone.
4913 *
4914 * Returns
4915 * 0 - success
4916 * ECONNRESET - success, but handle was reset
4917 * ECONNABORTED - repository connection broken
4918 * ECANCELED - pg was deleted
4919 */
4920 static int
4921 dgraph_set_runlevel(scf_propertygroup_t *pg, scf_property_t *prop)
4922 {
4923     char rl;
4924     scf_handle_t *h;
4925     int r;
4926     const char *ms = NULL; /* what to commit as options/milestone */
4927     boolean_t rebound = B_FALSE;
4928     int mark_rl = 0;

4930     const char * const stop = "stop";

4932     r = libscf_extract_runlevel(prop, &rl);
4933     switch (r) {
4934     case 0:
4935         break;

4937     case ECONNABORTED:
4938     case ECANCELED:
4939         return (r);

4941     case EINVAL:
4942     case ENOENT:
4943         log_error(LOG_WARNING, "runlevel property is misconfigured; "
4944                 "ignoring.\n");
4945         /* delete the bad property */
4946         goto nlock_out;

4948     default:
4949         bad_error("libscf_extract_runlevel", r);
4950     }

4952     switch (rl) {
4953     case 's':
4954         rl = 'S';
4955         /* FALLTHROUGH */

4957     case 'S':
4958     case '2':
4959     case '3':
4960         /*
4961         * These cases cause a milestone change, so
4962         * graph_runlevel_changed() will eventually deal with
4963         * signalling init.
4964         */
4965         break;

4967     case '0':

```

```

4968     case '1':
4969     case '4':
4970     case '5':
4971     case '6':
4972         mark_rl = 1;
4973         break;
4974
4975     default:
4976         log_framework(LOG_NOTICE, "Unknown runlevel '%c'.\n", rl);
4977         ms = NULL;
4978         goto noloack_out;
4979     }
4980
4981     h = scf_pg_handle(pg);
4982
4983     MUTEX_LOCK(&dgraph_lock);
4984
4985     /*
4986     * Since this triggers no milestone changes, force it by hand.
4987     */
4988     if (current_runlevel == '4' && rl == '3')
4989         mark_rl = 1;
4990
4991     /*
4992     * 1. If we are here after an "init X":
4993     *
4994     * init X
4995     *     init/lscf_set_runlevel()
4996     *     process_pg_event()
4997     *     dgraph_set_runlevel()
4998     *
4999     * then we haven't passed through graph_runlevel_changed() yet,
5000     * therefore 'current_runlevel' has not changed for sure but 'rl' has.
5001     * In consequence, if 'rl' is lower than 'current_runlevel', we change
5002     * the system runlevel and execute the appropriate /etc/rc?.d/K* scripts
5003     * past this test.
5004     *
5005     * 2. On the other hand, if we are here after a "svcadm milestone":
5006     *
5007     * svcadm milestone X
5008     *     dgraph_set_milestone()
5009     *     handle_graph_update_event()
5010     *     dgraph_set_instance_state()
5011     *     graph_post_X_[online|offline]()
5012     *     graph_runlevel_changed()
5013     *     signal_init()
5014     *     init/lscf_set_runlevel()
5015     *     process_pg_event()
5016     *     dgraph_set_runlevel()
5017     *
5018     * then we already passed through graph_runlevel_changed() (by the way
5019     * of dgraph_set_milestone()) and 'current_runlevel' may have changed
5020     * and already be equal to 'rl' so we are going to return immediately
5021     * from dgraph_set_runlevel() without changing the system runlevel and
5022     * without executing the /etc/rc?.d/K* scripts.
5023     */
5024     if (rl == current_runlevel) {
5025         ms = NULL;
5026         goto out;
5027     }
5028
5029     log_framework(LOG_DEBUG, "Changing to runlevel '%c'.\n", rl);
5030
5031     /*
5032     * Make sure stop rc scripts see the new settings via who -r.
5033     */

```

```

5034         utmpx_set_runlevel(rl, current_runlevel, B_TRUE);
5035
5036     /*
5037     * Some run levels don't have a direct correspondence to any
5038     * milestones, so we have to signal init directly.
5039     */
5040     if (mark_rl) {
5041         current_runlevel = rl;
5042         signal_init(rl);
5043     }
5044
5045     switch (rl) {
5046     case 'S':
5047         uu_warn("The system is coming down for administration. "
5048             "Please wait.\n");
5049         fork_rc_script(rl, stop, B_FALSE);
5050         ms = single_user_fmri;
5051         go_single_user_mode = B_TRUE;
5052         break;
5053
5054     case '0':
5055         halting_time = time(NULL);
5056         fork_rc_script(rl, stop, B_TRUE);
5057         halting = AD_HALT;
5058         goto uadmin;
5059
5060     case '5':
5061         halting_time = time(NULL);
5062         fork_rc_script(rl, stop, B_TRUE);
5063         halting = AD_POWEROFF;
5064         goto uadmin;
5065
5066     case '6':
5067         halting_time = time(NULL);
5068         fork_rc_script(rl, stop, B_TRUE);
5069         halting = AD_BOOT;
5070         goto uadmin;
5071
5072     uadmin:
5073         uu_warn("The system is coming down. Please wait.\n");
5074         ms = "none";
5075
5076     /*
5077     * We can't wait until all services are offline since this
5078     * thread is responsible for taking them offline. Instead we
5079     * set halting to the second argument for uadmin() and call
5080     * do_uadmin() from dgraph_set_instance_state() when
5081     * appropriate.
5082     */
5083     break;
5084
5085     case '1':
5086         if (current_runlevel != 'S') {
5087             uu_warn("Changing to state 1.\n");
5088             fork_rc_script(rl, stop, B_FALSE);
5089         } else {
5090             uu_warn("The system is coming up for administration. "
5091                 "Please wait.\n");
5092         }
5093         ms = single_user_fmri;
5094         go_to_level1 = B_TRUE;
5095         break;
5096
5097     case '2':
5098         if (current_runlevel == '3' || current_runlevel == '4')
5099             fork_rc_script(rl, stop, B_FALSE);

```

```
5100             ms = multi_user_fmri;
5101             break;

5103             case '3':
5104             case '4':
5105                 ms = "all";
5106                 break;

5108             default:
5109 #ifndef NDEBUG
5110                 (void) fprintf(stderr, "%s:%d: Uncaught case %d ('%c').\n",
5111                     __FILE__, __LINE__, rl, rl);
5112 #endif
5113                 abort();
5114             }

5116 out:
5117             MUTEX_UNLOCK(&dgraph_lock);

5119 noload_out:
5120             switch (r = libscf_clear_runlevel(pg, ms)) {
5121             case 0:
5122                 break;

5124             case ECONNABORTED:
5125                 libscf_handle_rebind(h);
5126                 rebound = B_TRUE;
5127                 goto noload_out;

5129             case ECANCELED:
5130                 break;

5132             case EPERM:
5133             case EACCES:
5134             case EROFS:
5135                 log_error(LOG_NOTICE, "Could not delete \"%s/%s\" property: "
5136                     "%s.\n", SCF_PG_OPTIONS, "runlevel", strerror(r));
5137                 break;

5139             default:
5140                 bad_error("libscf_clear_runlevel", r);
5141             }

5143             return (rebound ? ECONNRESET : 0);
5144 }
unchanged portion omitted
```

```

*****
30856 Thu Feb 26 16:21:35 2009
new/usr/src/cmd/svc/started/method.c
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22  * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23  * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
26 #pragma ident "%Z%%M% %I% %E% SMI"
27
28 /*
29  * method.c - method execution functions
30  *
31  * This file contains the routines needed to run a method: a fork(2)-exec(2)
32  * invocation monitored using either the contract filesystem or waitpid(2).
33  * (Plain fork1(2) support is provided in fork.c.)
34  *
35  * Contract Transfer
36  *
37  * When we restart a service, we want to transfer any contracts that the old
38  * service's contract inherited. This means that (a) we must not abandon the
39  * old contract when the service dies and (b) we must write the id of the old
40  * contract into the terms of the new contract. There should be limits to
41  * (a), though, since we don't want to keep the contract around forever. To
42  * this end we'll say that services in the offline state may have a contract
43  * to be transferred and services in the disabled or maintenance states cannot.
44  * This means that when a service transitions from online (or degraded) to
45  * offline, the contract should be preserved, and when the service transitions
46  * from offline to online (i.e., the start method), we'll transfer inherited
47  * contracts.
48  */
49
50 #include <sys/contract/process.h>
51 #include <sys/ctfs.h>
52 #include <sys/stat.h>
53 #include <sys/time.h>
54 #include <sys/types.h>
55 #include <sys/uio.h>
56 #include <sys/wait.h>
57 #include <alloca.h>
58 #include <assert.h>
59 #include <errno.h>
60 #include <fcntl.h>

```

```

58 #include <libcontract.h>
59 #include <libcontract_priv.h>
60 #include <libgen.h>
61 #include <librestart.h>
62 #include <libscf.h>
63 #include <limits.h>
64 #include <port.h>
65 #include <sac.h>
66 #include <signal.h>
67 #include <stdlib.h>
68 #include <string.h>
69 #include <strings.h>
70 #include <unistd.h>
71 #include <atomic.h>
72 #include <poll.h>
73
74 #include "startd.h"
75
76 #define SBIN_SH "/sbin/sh"
77
78 /*
79  * Used to tell if contracts are in the process of being
80  * stored into the svc.startd internal hash table.
81  */
82 volatile uint16_t storing_contract = 0;
83
84 /*
85  * Mapping from restart_on method-type to contract events. Must correspond to
86  * enum method_restart_t.
87  */
88 static uint_t method_events[] = {
89  /* METHOD_RESTART_ALL */
90  CT_PR_EV_HWERR | CT_PR_EV_SIGNAL | CT_PR_EV_CORE | CT_PR_EV_EMPTY,
91  /* METHOD_RESTART_EXTERNAL_FAULT */
92  CT_PR_EV_HWERR | CT_PR_EV_SIGNAL,
93  /* METHOD_RESTART_ANY_FAULT */
94  CT_PR_EV_HWERR | CT_PR_EV_SIGNAL | CT_PR_EV_CORE
95 };
96
97 unchanged_portion_omitted
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

602     scf_handle_t *h;
603     scf_snapshot_t *snap;
604     const char *mname;
605     const char *errstr;
606     struct method_context *mcp;
607     int result = 0, timeout_fired = 0;
608     int sig, r;
609     boolean_t transient;
610     uint64_t timeout;
611     uint8_t timeout_retry;
612     ctid_t ctid;
613     int ctfid = -1;
614     restarter_inst_t *inst = *instp;
615     int id = inst->ri_id;
616     int forkerr;

618     assert(PTHREAD_MUTEX_HELD(&inst->ri_lock));
619     assert(instance_in_transition(inst));

621     if (inst->ri_mi_deleted)
622         return (ECANCELED);

624     *exit_code = 0;

626     assert(0 <= type && type <= 2);
627     mname = method_names[type];

629     if (type == METHOD_START)
630         inst->ri_pre_online_hook();

632     h = scf_instance_handle(inst->ri_m_inst);

634     snap = scf_snapshot_create(h);
635     if (snap == NULL ||
636         scf_instance_get_snapshot(inst->ri_m_inst, "running", snap) != 0) {
637         log_framework(LOG_DEBUG,
638             "Could not get running snapshot for %s. "
639             "Using editing version to run method %s.\n",
640             inst->ri_i_fmri, mname);
641         scf_snapshot_destroy(snap);
642         snap = NULL;
643     }

645     /*
646     * After this point, we may be logging to the instance log.
647     * Make sure we've noted where that log is as a property of
648     * the instance.
649     */
650     r = libscf_note_method_log(inst->ri_m_inst, st->st_log_prefix,
651         inst->ri_logstem);
652     if (r != 0) {
653         log_framework(LOG_WARNING,
654             "%s: couldn't note log location: %s\n",
655             inst->ri_i_fmri, strerror(r));
656     }

658     if ((method = libscf_get_method(h, type, inst, snap, &restart_on,
659         &cte_mask, &need_session, &timeout, &timeout_retry)) == NULL) {
660         if (errno == LIBSCF_PGROUP_ABSENT) {
661             log_framework(LOG_DEBUG,
662                 "%s: instance has no method property group '%s'.\n",
663                 inst->ri_i_fmri, mname);
664             if (type == METHOD_REFRESH)
665                 log_instance(inst, B_TRUE, "No '%s' method "
666                     "defined. Treating as :true.", mname);
667             else

```

```

668         log_instance(inst, B_TRUE, "Method property "
669             "group '%s' is not present.", mname);
670         scf_snapshot_destroy(snap);
671         return (0);
672     } else if (errno == LIBSCF_PROPERTY_ABSENT) {
673         log_framework(LOG_DEBUG,
674             "%s: instance has no '%s/exec' method property.\n",
675             inst->ri_i_fmri, mname);
676         log_instance(inst, B_TRUE, "Method property '%s/exec "
677             "is not present.", mname);
678         scf_snapshot_destroy(snap);
679         return (0);
680     } else {
681         log_error(LOG_WARNING,
682             "%s: instance libscf_get_method failed\n",
683             inst->ri_i_fmri);
684         scf_snapshot_destroy(snap);
685         return (EINVAL);
686     }
687 }

689 /* open service contract if stopping a non-transient service */
690 if (type == METHOD_STOP && (!instance_is_transient_style(inst))) {
691     if (inst->ri_i_primary_ctid == 0) {
692         /* service is not running, nothing to stop */
693         log_framework(LOG_DEBUG, "%s: instance has no primary "
694             "contract, no service to stop.\n",
695             inst->ri_i_fmri);
696         scf_snapshot_destroy(snap);
697         return (0);
698     }
699     if ((ctfid = contract_open(inst->ri_i_primary_ctid, "process",
700         "events", O_RDONLY)) < 0) {
701         result = EFAULT;
702         log_instance(inst, B_TRUE, "Could not open service "
703             "contract %ld. Stop method not run.",
704             inst->ri_i_primary_ctid);
705         goto out;
706     }
707 }

709     if (restarter_is_null_method(method)) {
710         log_framework(LOG_DEBUG, "%s: null method succeeds\n",
711             inst->ri_i_fmri);

713         log_instance(inst, B_TRUE, "Executing %s method (null).",
714             mname);

716         if (type == METHOD_START)
717             write_status(inst, mname, 0);
718         goto out;
719     }

721     sig = restarter_is_kill_method(method);
722     if (sig >= 0) {

724         if (inst->ri_i_primary_ctid == 0) {
725             log_error(LOG_ERR, "%s: :kill with no contract\n",
726                 inst->ri_i_fmri);
727             log_instance(inst, B_TRUE, "Invalid use of \":kill\" "
728                 "as stop method for transient service.");
729             result = EINVAL;
730             goto out;
731         }

733         log_framework(LOG_DEBUG,

```

```

734         "%s: :killing contract with signal %d\n",
735         inst->ri_i.i_fmri, sig);

737         log_instance(inst, B_TRUE, "Executing %s method (:kill).",
738         mname);

740         if (contract_kill(inst->ri_i.i_primary_ctid, sig,
741         inst->ri_i.i_fmri) != 0) {
742             result = EIO;
743             goto out;
744         } else
745             goto assured_kill;
746     }

748     log_framework(LOG_DEBUG, "%s: forking to run method %s\n",
749     inst->ri_i.i_fmri, method);

751     errstr = restarter_get_method_context(RESTARTER_METHOD_CONTEXT_VERSION,
752     inst->ri_m_inst, snap, mname, method, &mcp);

754     if (errstr != NULL) {
755         log_error(LOG_WARNING, "%s: %s\n", inst->ri_i.i_fmri, errstr);
756         result = EINVAL;
757         goto out;
758     }

760     r = method_ready_contract(inst, type, restart_on, cte_mask);
761     if (r != 0) {
762         assert(r == ECANCELED);
763         assert(inst->ri_mi_deleted);
764         restarter_free_method_context(mcp);
765         result = ECANCELED;
766         goto out;
767     }

769     /*
770     * Validate safety of method contexts, to save children work.
771     */
772     if (!restarter_rm_libs_loadable())
773         log_framework(LOG_DEBUG, "%s: method contexts limited "
774         "to root-accessible libraries\n", inst->ri_i.i_fmri);

776     /*
777     * If the service is restarting too quickly, send it to
778     * maintenance.
779     */
780     if (type == METHOD_START) {
781         method_record_start(inst);
782         if (method_rate_critical(inst)) {
783             log_instance(inst, B_TRUE, "Restarting too quickly, "
784             "changing state to maintenance.");
785             result = ELOOP;
786             restarter_free_method_context(mcp);
787             goto out;
788         }
789     }

791     atomic_add_16(&storing_contract, 1);
792     pid = startd_fork1(&forkerr);
793     if (pid == 0)
794         exec_method(inst, type, method, mcp, need_session);

796     if (pid == -1) {
797         atomic_add_16(&storing_contract, -1);
798         if (forkerr == EAGAIN)
799             result = EAGAIN;

```

```

800         else
801             result = EFAULT;

803         log_error(LOG_WARNING,
804         "%s: Couldn't fork to execute method %s: %s\n",
805         inst->ri_i.i_fmri, method, strerror(forkerr));

807         restarter_free_method_context(mcp);
808         goto out;
809     }

812     /*
813     * Get the contract id, decide whether it is primary or transient, and
814     * stash it in inst & the repository.
815     */
816     method_store_contract(inst, type, &ctid);
817     atomic_add_16(&storing_contract, -1);

819     restarter_free_method_context(mcp);

821     /*
822     * Similarly for the start method PID.
823     */
824     if (type == METHOD_START && !inst->ri_mi_deleted)
825         (void) libscf_write_start_pid(inst->ri_m_inst, pid);

827     if (instance_is_wait_style(inst) && type == METHOD_START) {
828         /* Wait style instances don't get timeouts on start methods. */
829         if (wait_register(pid, inst->ri_i.i_fmri, 1, 0)) {
830             log_error(LOG_WARNING,
831             "%s: couldn't register %ld for wait\n",
832             inst->ri_i.i_fmri, pid);
833             result = EFAULT;
834             goto contract_out;
835         }
836         write_status(inst, mname, 0);

838     } else {
839         int r, err;
840         time_t start_time;
841         time_t end_time;

843         /*
844         * Because on upgrade/live-upgrade we may have no chance
845         * to override faulty timeout values on the way to
846         * manifest import, all services on the path to manifest
847         * import are treated the same as INFINITE timeout services.
848         */

850         start_time = time(NULL);
851         if (timeout != METHOD_TIMEOUT_INFINITE && !is_timeout_ovr(inst))
852             timeout_insert(inst, ctid, timeout);
853         else
854             timeout = METHOD_TIMEOUT_INFINITE;

856         /* Unlock the instance while waiting for the method. */
857         MUTEX_UNLOCK(&inst->ri_lock);

859         do {
860             r = waitpid(pid, &ret_status, NULL);
861         } while (r == -1 && errno == EINTR);
862         if (r == -1)
863             err = errno;

865         /* Re-grab the lock. */

```

```

866     inst = inst_lookup_by_id(id);
867
868     /*
869     * inst can't be removed, as the removal thread waits
870     * for completion of this one.
871     */
872     assert(inst != NULL);
873     *instp = inst;
874
875     if (inst->ri_timeout != NULL && inst->ri_timeout->te_fired)
876         timeout_fired = 1;
877
878     timeout_remove(inst, ctdid);
879
880     log_framework(LOG_DEBUG,
881                 "%s method for %s exited with status %d.\n", mname,
882                 inst->ri_i_fmri, WEXITSTATUS(ret_status));
883
884     if (r == -1) {
885         log_error(LOG_WARNING,
886                 "Couldn't waitpid() for %s method of %s (%s).\n",
887                 mname, inst->ri_i_fmri, strerror(err));
888         result = EFAULT;
889         goto contract_out;
890     }
891
892     if (type == METHOD_START)
893         write_status(inst, mname, ret_status);
894
895     /* return ERANGE if this service doesn't retry on timeout */
896     if (timeout_fired == 1 && timeout_retry == 0) {
897         result = ERANGE;
898         goto contract_out;
899     }
900
901     if (!WIFEXITED(ret_status)) {
902         /*
903         * If method didn't exit itself (it was killed by an
904         * external entity, etc.), consider the entire
905         * method_run as failed.
906         */
907         if (WIFSIGNALED(ret_status)) {
908             char buf[SIG2STR_MAX];
909             (void) sig2str(WTERMSIG(ret_status), buf);
910
911             log_error(LOG_WARNING, "%s: Method \"%s\" "
912                     "failed due to signal %s.\n",
913                     inst->ri_i_fmri, method, buf);
914             log_instance(inst, B_TRUE, "Method \"%s\" "
915                     "failed due to signal %s.", mname, buf);
916         } else {
917             log_error(LOG_WARNING, "%s: Method \"%s\" "
918                     "failed with exit status %d.\n",
919                     inst->ri_i_fmri, method,
920                     WEXITSTATUS(ret_status));
921             log_instance(inst, B_TRUE, "Method \"%s\" "
922                     "failed with exit status %d.", mname,
923                     WEXITSTATUS(ret_status));
924         }
925         result = EAGAIN;
926         goto contract_out;
927     }
928
929     *exit_code = WEXITSTATUS(ret_status);
930     if (*exit_code != 0) {
931         log_error(LOG_WARNING,

```

```

932         "%s: Method \"%s\" failed with exit status %d.\n",
933         inst->ri_i_fmri, method, WEXITSTATUS(ret_status));
934     }
935
936     log_instance(inst, B_TRUE, "Method \"%s\" exited with status "
937                 "%d.", mname, *exit_code);
938
939     if (*exit_code != 0)
940         goto contract_out;
941
942     end_time = time(NULL);
943
944     /* Give service contract remaining seconds to empty */
945     if (timeout != METHOD_TIMEOUT_INFINITE)
946         timeout -= (end_time - start_time);
947     }
948
949 assured_kill:
950     /*
951     * For stop methods, assure that the service contract has emptied
952     * before returning.
953     */
954     if (type == METHOD_STOP && (!instance_is_transient_style(inst)) &&
955         !(contract_is_empty(inst->ri_i_primary_ctid))) {
956         int times = 0;
957
958         if (timeout != METHOD_TIMEOUT_INFINITE)
959             timeout_insert(inst, inst->ri_i_primary_ctid,
960                 timeout);
961
962         for (;;) {
963             /*
964             * Check frequently at first, then back off. This
965             * keeps started from idling while shutting down.
966             */
967             if (times < 20) {
968                 (void) poll(NULL, 0, 5);
969                 times++;
970             } else {
971                 (void) poll(NULL, 0, 100);
972             }
973             if (contract_is_empty(inst->ri_i_primary_ctid))
974                 break;
975         }
976
977         if (timeout != METHOD_TIMEOUT_INFINITE)
978             if (inst->ri_timeout->te_fired)
979                 result = EFAULT;
980
981         timeout_remove(inst, inst->ri_i_primary_ctid);
982     }
983
984 contract_out:
985     /* Abandon contracts for transient methods & methods that fail. */
986     transient = method_is_transient(inst, type);
987     if ((transient || *exit_code != 0 || result != 0) &&
988         (restarter_is_kill_method(method) < 0))
989         method_remove_contract(inst, !transient, B_TRUE);
990
991 out:
992     if (ctfd >= 0)
993         (void) close(ctfd);
994     scf_snapshot_destroy(snap);
995     free(method);
996     return (result);
997 }

```

unchanged portion omitted

```

*****
22851 Thu Feb 26 16:21:37 2009
new/usr/src/cmd/svc/startd/startd.h
6805730 some simple changes would make 'init 5' much faster
6809492 startd shouldn't let hung subprocesses impede shutdown
*****
_____unchanged_portion_omitted_____

552 extern volatile uint16_t      storing_contract;

554 uu_list_pool_t *contract_list_pool;

556 /* contract.c */
557 ctid_t contract_init(void);
558 void contract_abandon(ctid_t);
559 int contract_kill(ctid_t, int, const char *);
560 int contract_is_empty(ctid_t);
561 void contract_hash_init();
562 void contract_hash_store(ctid_t, int);
563 void contract_hash_remove(ctid_t);
564 int lookup_inst_by_contract(ctid_t);

566 /* dict.c */
567 void dict_init(void);
568 int dict_lookup_byname(const char *);
569 int dict_insert(const char *);

571 /* expand.c */
572 int expand_method_tokens(const char *, scf_instance_t *,
573     scf_snapshot_t *, int, char **);

575 /* env.c */
576 void init_env(void);
577 char **set_smf_env(char **, size_t, const char *,
578     const restarter_inst_t *, const char *);

580 /* file.c */
581 int file_ready(graph_vertex_t *);

583 /* fork.c */
584 int fork_mount(char *, char *);
585 void fork_sulogin(boolean_t, const char *, ...);
586 void fork_rc_script(char, const char *, boolean_t);

588 void *fork_configd_thread(void *);

590 pid_t startd_fork1(int *);
591 void fork_with_timeout(const char *, uint_t, uint_t);

593 /* graph.c */
594 void graph_init(void);
595 void *single_user_thread(void *);
596 void *graph_thread(void *);
597 void *graph_event_thread(void *);
598 void *repository_event_thread(void *);
599 int dgraph_add_instance(const char *, scf_instance_t *, boolean_t);
600 void graph_engine_start(void);
601 void graph_enable_by_vertex(graph_vertex_t *, int, int);
602 int refresh_vertex(graph_vertex_t *, scf_instance_t *);
603 void vertex_send_event(graph_vertex_t *, restarter_event_type_t);
604 void graph_start_if_satisfied(graph_vertex_t *);
605 int vertex_subgraph_dependencies_shutdown(scf_handle_t *, graph_vertex_t *,
606     restarter_instance_state_t);
607 void graph_transition_sulogin(restarter_instance_state_t,
608     restarter_instance_state_t);
609 void graph_transition_propagate(graph_vertex_t *, propagate_event_t,

```

```

610     restarter_error_t);
611 void graph_offline_subtree_leaves(graph_vertex_t *, void *);

613 /* libscf.c - common */
614 char *inst_fmri_to_svc_fmri(const char *);
615 void *libscf_object_create(void (*)(scf_handle_t *), scf_handle_t *);
616 int libscf_instance_get_fmri(scf_instance_t *, char **);
617 int libscf_fmri_get_instance(scf_handle_t *, const char *, scf_instance_t **);
618 int libscf_lookup_instance(const char *, scf_instance_t *);
619 int libscf_set_reconfig(int);
620 scf_snapshot_t *libscf_get_or_make_running_snapshot(scf_instance_t *,
621     const char *, boolean_t);
622 int libscf_inst_set_count_prop(scf_instance_t *, const char *,
623     const char *pgtype, uint32_t, const char *, uint64_t);

625 /* libscf.c - used by graph.c */
626 int libscf_get_deathrow(scf_handle_t *, scf_instance_t *, int *);
627 int libscf_get_basic_instance_data(scf_handle_t *, scf_instance_t *,
628     const char *, int *, int *, char **);
629 int libscf_inst_get_or_add_pg(scf_instance_t *, const char *, const char *,
630     uint32_t, scf_propertygroup_t *);
631 int libscf_read_states(const scf_propertygroup_t *,
632     restarter_instance_state_t *, restarter_instance_state_t *);
633 int depgroup_empty(scf_handle_t *, scf_propertygroup_t *);
634 gv_type_t depgroup_read_scheme(scf_handle_t *, scf_propertygroup_t *);
635 depgroup_type_t depgroup_read_grouping(scf_handle_t *, scf_propertygroup_t *);
636 restarter_error_t depgroup_read_restart(scf_handle_t *, scf_propertygroup_t *);
637 int libscf_set_enable_ovr(scf_instance_t *, int);
638 int libscf_set_deathrow(scf_instance_t *, int);
639 int libscf_delete_enable_ovr(scf_instance_t *);
640 int libscf_get_milestone(scf_instance_t *, scf_property_t *, scf_value_t *,
641     char *, size_t);
642 int libscf_extract_runlevel(scf_property_t *, char *);
643 int libscf_clear_runlevel(scf_propertygroup_t *, const char *milestone);

645 typedef int (*callback_t)(void *, void *);

647 int walk_dependency_pgs(scf_instance_t *, callback_t, void *);
648 int walk_property_astrings(scf_property_t *, callback_t, void *);

650 /* libscf.c - used by restarter.c/method.c/expand.c */
651 char *libscf_get_method(scf_handle_t *, int, restarter_inst_t *,
652     scf_snapshot_t *, method_restart_t *, uint_t *, uint8_t *, uint64_t *,
653     uint8_t *);
654 void libscf_populate_graph(scf_handle_t *h);
655 int update_fault_count(restarter_inst_t *, int);
656 int libscf_unset_action(scf_handle_t *, scf_propertygroup_t *, admin_action_t,
657     int64_t);
658 int libscf_get_startd_properties(scf_instance_t *, scf_snapshot_t *, uint_t *,
659     char **);
660 int libscf_get_template_values(scf_instance_t *, scf_snapshot_t *, char **,
661     char **);

663 int libscf_read_method_ids(scf_handle_t *, scf_instance_t *, const char *,
664     ctid_t *, ctid_t *, pid_t *);
665 int libscf_write_start_pid(scf_instance_t *, pid_t);
666 int libscf_write_method_status(scf_instance_t *, const char *, int);
667 int libscf_note_method_log(scf_instance_t *, const char *, const char *);

669 scf_handle_t *libscf_handle_create_bound(scf_version_t);
670 void libscf_handle_rebind(scf_handle_t *);
671 scf_handle_t *libscf_handle_create_bound_loop(void);

673 scf_snapshot_t *libscf_get_running_snapshot(scf_instance_t *);
674 int libscf_snapshots_poststart(scf_handle_t *, const char *, boolean_t);
675 int libscf_snapshots_refresh(scf_instance_t *, const char *);

```

```

677 int instance_is_transient_style(restarter_inst_t *);
678 int instance_is_wait_style(restarter_inst_t *);

680 int libscf_create_self(scf_handle_t *);

682 void libscf_reget_instance(restarter_inst_t *);

684 /* log.c */
685 void log_init();
686 void log_error(int, const char *, ...);
687 void log_framework(int, const char *, ...);
688 void log_framework2(int, int, const char *, ...);
689 void log_console(int, const char *, ...);
690 void log_preexec(void);
691 void setlog(const char *);
692 void log_transition(const restarter_inst_t *, start_outcome_t);
693 void log_instance(const restarter_inst_t *, boolean_t, const char *, ...);
694 void log_instance_fmri(const char *, const char *, boolean_t,
695     const char *, ...);

697 /* method.c */
698 void *method_thread(void *);
699 void method_remove_contract(restarter_inst_t *, boolean_t, boolean_t);

701 /* misc.c */
702 void started_close(int);
703 void started_fclose(FILE *);
704 int fmri_canonify(const char *, char **, boolean_t);
705 int fs_is_read_only(char *, ulong_t *);
706 int fs_remount(char *);
707 void xstr_sanitise(char *);

709 /* restarter.c */
710 void restarter_init(void);
711 void restarter_start(void);
712 int instance_in_transition(restarter_inst_t *);
713 int restarter_instance_update_states(scf_handle_t *, restarter_inst_t *,
714     restarter_instance_state_t, restarter_instance_state_t, restarter_error_t,
715     char *);
716 int stop_instance_fmri(scf_handle_t *, const char *, uint_t);
717 restarter_inst_t *inst_lookup_by_id(int);
718 void restarter_mark_pending_snapshot(const char *, uint_t);
719 void *restarter_post_fsminimal_thread(void *);
720 void timeout_insert(restarter_inst_t *, ctid_t, uint64_t);
721 void timeout_remove(restarter_inst_t *, ctid_t);
722 void timeout_init(void);
723 int is_timeout_ovr(restarter_inst_t *);

725 /* started.c */
726 void *safe_realloc(void *, size_t);
727 char *safe_strdup(const char *s);
728 void *started_alloc_retry(void (*)(size_t, int), size_t);
729 void started_free(void *, size_t);
730 uu_list_pool_t *started_list_pool_create(const char *, size_t, size_t,
731     uu_compare_fn_t *, uint32_t);
732 uu_list_t *started_list_create(uu_list_pool_t *, void *, uint32_t);
733 pthread_t started_thread_create(void (*)(void *), void *);

735 /* special.c */
736 void special_null_transition(void);
737 void special_online_hooks_get(const char *, instance_hook_t *,
738     instance_hook_t *, instance_hook_t *);

740 /* transition.c */
741 int gt_transition(scf_handle_t *, graph_vertex_t *, restarter_error_t,

```

```

742     restarter_instance_state_t);

744 /* utmpx.c */
745 void utmpx_init(void);
746 void utmpx_clear_old(void);
747 int utmpx_mark_init(pid_t, char *);
748 void utmpx_mark_dead(pid_t, int, boolean_t);
749 char utmpx_get_runlevel(void);
750 void utmpx_set_runlevel(char, char, boolean_t);
751 void utmpx_write_boottime(void);

753 /* wait.c */
754 void wait_init(void);
755 void wait_prefork(void);
756 void wait_postfork(pid_t);
757 int wait_register(pid_t, const char *, int, int);
758 void *wait_thread(void *);
759 void wait_ignore_by_fmri(const char *);

761 /* proc.c */
762 ctid_t proc_get_ctid();

764 /* deathrow.c */
765 extern void deathrow_init();
766 extern void deathrow_fini();
767 extern boolean_t is_fmri_in_deathrow(const char *);

769 #ifdef __cplusplus
770 }

```

unchanged portion omitted