****
60359 Wed Jan 30 11:17:45 2019
new/usr/src/cmd/zoneadmd/zoneadmd.c
10141 smatch fix for zoneadmd
****
______***unchanged_portion_omitted_***

```
int
main(int argc, char *argv[])
{
        int opt;
        zoneid_t zid;
        priv_set_t *privset;
        zone_state_t zstate;
        char parents_locale[MAXPATHLEN];
        brand_handle_t bh;
        int err;

        pid_t pid;
        sigset_t blockset;
        sigset_t block_cld;

        struct {
                sema_t sem;
                int status;
                zlog_t log;
        } *shstate;
        size_t shstatelen = getpagesize();

        zlog_t errlog;
        zlog_t *zlogp;

        int ctfd;

        progname = get_execbasename(argv[0]);

        /*
         * Make sure stderr is unbuffered
         */
        (void) setbuffer(stderr, NULL, 0);

        /*
         * Get out of the way of mounted filesystems, since we will daemonize
         * soon.
         */
        (void) chdir("/");

        /*
         * Use the default system umask per PSARC 1998/110 rather than
         * anything that may have been set by the caller.
         */
        (void) umask(CMASK);

        /*
         * Initially we want to use our parent's locale.
         */
        (void) setlocale(LC_ALL, "");
        (void) textdomain(TEXT_DOMAIN);
        (void) strlcpy(parents_locale, setlocale(LC_MESSAGES, NULL),
            sizeof (parents_locale));

        /*
         * This zlog_t is used for writing to stderr
         */
        errlog.logfile = stderr;
        errlog.buflen = errlog.loglen = 0;
```

```
        errlog.buf = errlog.log = NULL;
        errlog.locale = parents_locale;

        /*
         * We start off writing to stderr until we're ready to daemonize.
         */
        zlogp = &errlog;

        /*
         * Process options.
         */
        while ((opt = getopt(argc, argv, "R:z:")) != EOF) {
                switch (opt) {
                case 'R':
                        zonecfg_set_root(optarg);
                        break;
                case 'z':
                        zone_name = optarg;
                        break;
                default:
                        usage();
                }
        }

        if (zone_name == NULL)
                usage();

        /*
         * Because usage() prints directly to stderr, it has gettext()
         * wrapping, which depends on the locale.  But since zerror() calls
         * localize() which tweaks the locale, it is not safe to call zerror()
         * until after the last call to usage().  Fortunately, the last call
         * to usage() is just above and the first call to zerror() is just
         * below.  Don't mess this up.
         */
        if (strcmp(zone_name, GLOBAL_ZONENAME) == 0) {
                zerror(zlogp, B_FALSE, "cannot manage the %s zone",
                    GLOBAL_ZONENAME);
                return (1);
        }

        if (zone_get_id(zone_name, &zid) != 0) {
                zerror(zlogp, B_FALSE, "could not manage %s: %s", zone_name,
                    zonecfg_strerror(Z_NO_ZONE));
                return (1);
        }

        if ((err = zone_get_state(zone_name, &zstate)) != Z_OK) {
                zerror(zlogp, B_FALSE, "failed to get zone state: %s",
                    zonecfg_strerror(err));
                return (1);
        }
        if (zstate < ZONE_STATE_INCOMPLETE) {
                zerror(zlogp, B_FALSE,
                    "cannot manage a zone which is in state '%s'",
                    zone_state_str(zstate));
                return (1);
        }

        if (zonecfg_default_brand(default_brand,
            sizeof (default_brand)) != Z_OK) {
                zerror(zlogp, B_FALSE, "unable to determine default brand");
                return (1);
        }

        /* Get a handle to the brand info for this zone */
```

```
	if (zone_get_brand(zone_name, brand_name, sizeof (brand_name))
	    != Z_OK) {
		zerror(zlogp, B_FALSE, "unable to determine zone brand");
		return (1);
	}
	zone_isnative = (strcmp(brand_name, NATIVE_BRAND_NAME) == 0);
	zone_islabeled = (strcmp(brand_name, LABELED_BRAND_NAME) == 0);

	/*
	 * In the alternate root environment, the only supported
	 * operations are mount and unmount.  In this case, just treat
	 * the zone as native if it is cluster.  Cluster zones can be
	 * native for the purpose of LU or upgrade, and the cluster
	 * brand may not exist in the miniroot (such as in net install
	 * upgrade).
	 */
	if (strcmp(brand_name, CLUSTER_BRAND_NAME) == 0) {
		zone_iscluster = B_TRUE;
		if (zonecfg_in_alt_root()) {
			(void) strlcpy(brand_name, default_brand,
			    sizeof (brand_name));
		}
	} else {
		zone_iscluster = B_FALSE;
	}

	if ((bh = brand_open(brand_name)) == NULL) {
		zerror(zlogp, B_FALSE, "unable to open zone brand");
		return (1);
	}

	/* Get state change brand hooks. */
	if (brand_callback_init(bh, zone_name) == -1) {
		zerror(zlogp, B_TRUE,
		    "failed to initialize brand state change hooks");
		brand_close(bh);
		return (1);
	}

	brand_close(bh);

	/*
	 * Check that we have all privileges.  It would be nice to pare
	 * this down, but this is at least a first cut.
	 */
	if ((privset = priv_allocset()) == NULL) {
		zerror(zlogp, B_TRUE, "%s failed", "priv_allocset");
		return (1);
	}

	if (getppriv(PRIV_EFFECTIVE, privset) != 0) {
		zerror(zlogp, B_TRUE, "%s failed", "getppriv");
		priv_freeset(privset);
		return (1);
	}

	if (priv_isfullset(privset) == B_FALSE) {
		zerror(zlogp, B_FALSE, "You lack sufficient privilege to "
		    "run this command (all privs required)");
		priv_freeset(privset);
		return (1);
	}
	priv_freeset(privset);

	if (mkzonedir(zlogp) != 0)
		return (1);
```

```
	/*
	 * Pre-fork: setup shared state
	 */
	if ((shstate = (void *)mmap(NULL, shstatelen,
	    PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANON, -1, (off_t)0)) ==
	    MAP_FAILED) {
		zerror(zlogp, B_TRUE, "%s failed", "mmap");
		return (1);
	}
	if (sema_init(&shstate->sem, 0, USYNC_PROCESS, NULL) != 0) {
		zerror(zlogp, B_TRUE, "%s failed", "sema_init()");
		(void) munmap((char *)shstate, shstatelen);
		return (1);
	}
	shstate->log.logfile = NULL;
	shstate->log.buflen = shstatelen - sizeof (*shstate);
	shstate->log.loglen = shstate->log.buflen;
	shstate->log.buf = (char *)shstate + sizeof (*shstate);
	shstate->log.log = shstate->log.buf;
	shstate->log.locale = parents_locale;
	shstate->status = -1;

	/*
	 * We need a SIGCHLD handler so the sema_wait() below will wake
	 * up if the child dies without doing a sema_post().
	 */
	(void) sigset(SIGCHLD, sigchld);
	/*
	 * We must mask SIGCHLD until after we've coped with the fork
	 * sufficiently to deal with it; otherwise we can race and
	 * receive the signal before pid has been initialized
	 * (yes, this really happens).
	 */
	(void) sigemptyset(&block_cld);
	(void) sigaddset(&block_cld, SIGCHLD);
	(void) sigprocmask(SIG_BLOCK, &block_cld, NULL);

	/*
	 * The parent only needs stderr after the fork, so close other fd's
	 * that we inherited from zoneadm so that the parent doesn't have those
	 * open while waiting. The child will close the rest after the fork.
	 */
	closefrom(3);

	if ((ctfd = init_template()) == -1) {
		zerror(zlogp, B_TRUE, "failed to create contract");
		return (1);
	}

	/*
	 * Do not let another thread localize a message while we are forking.
	 */
	(void) mutex_lock(&msglock);
	pid = fork();
	(void) mutex_unlock(&msglock);

	/*
	 * In all cases (parent, child, and in the event of an error) we
	 * don't want to cause creation of contracts on subsequent fork()s.
	 */
	(void) ct_tmpl_clear(ctfd);
	(void) close(ctfd);

	if (pid == -1) {
		zerror(zlogp, B_TRUE, "could not fork");
```

```
                return (1);

        } else if (pid > 0) { /* parent */
                (void) sigprocmask(SIG_UNBLOCK, &block_cld, NULL);
                /*
                 * This marks a window of vulnerability in which we receive
                 * the SIGCLD before falling into sema_wait (normally we would
                 * get woken up from sema_wait with EINTR upon receipt of
                 * SIGCLD).  So we may need to use some other scheme like
                 * sema_posting in the sigcld handler.
                 * blech
                 */
                (void) sema_wait(&shstate->sem);
                (void) sema_destroy(&shstate->sem);
                if (shstate->status != 0)
                        (void) waitpid(pid, NULL, WNOHANG);
                /*
                 * It's ok if we die with SIGPIPE.  It's not like we could have
                 * done anything about it.
                 */
                (void) fprintf(stderr, "%s", shstate->log.buf);
                _exit(shstate->status == 0 ? 0 : 1);
        }

        /*
         * The child charges on.
         */
        (void) sigset(SIGCHLD, SIG_DFL);
        (void) sigprocmask(SIG_UNBLOCK, &block_cld, NULL);

        /*
         * SIGPIPE can be delivered if we write to a socket for which the
         * peer endpoint is gone.  That can lead to too-early termination
         * of zoneadmd, and that's not good eats.
         */
        (void) sigset(SIGPIPE, SIG_IGN);
        /*
         * Stop using stderr
         */
        zlogp = &shstate->log;

        /*
         * We don't need stdout/stderr from now on.
         */
        closefrom(0);

        /*
         * Initialize the syslog zlog_t.  This needs to be done after
         * the call to closefrom().
         */
        logsys.buf = logsys.log = NULL;
        logsys.buflen = logsys.loglen = 0;
        logsys.logfile = NULL;
        logsys.locale = DEFAULT_LOCALE;

        openlog("zoneadmd", LOG_PID, LOG_DAEMON);

        /*
         * The eventstream is used to publish state changes in the zone
         * from the door threads to the console I/O poller.
         */
        if (eventstream_init() == -1) {
                zerror(zlogp, B_TRUE, "unable to create eventstream");
                goto child_out;
        }
```

```
        (void) snprintf(zone_door_path, sizeof (zone_door_path),
            "%s" ZONE_DOOR_PATH, zonecfg_get_root(), zone_name);

        /*
         * See if another zoneadmd is running for this zone.  If not, then we
         * can now modify system state.
         */
        if (make_daemon_exclusive(zlogp) == -1)
                goto child_out;


        /*
         * Create/join a new session; we need to be careful of what we do with
         * the console from now on so we don't end up being the session leader
         * for the terminal we're going to be handing out.
         */
        (void) setsid();

        /*
         * This thread shouldn't be receiving any signals; in particular,
         * SIGCHLD should be received by the thread doing the fork().
         */
        (void) sigfillset(&blockset);
        (void) thr_sigsetmask(SIG_BLOCK, &blockset, NULL);

        /*
         * Setup the console device and get ready to serve the console;
         * once this has completed, we're ready to let console clients
         * make an attempt to connect (they will block until
         * serve_console_sock() below gets called, and any pending
         * connection is accept()ed).
         */
        if (!zonecfg_in_alt_root() && init_console(zlogp) < 0)
                goto child_out;

        /*
         * Take the lock now, so that when the door server gets going, we
         * are guaranteed that it won't take a request until we are sure
         * that everything is completely set up.  See the child_out: label
         * below to see why this matters.
         */
        (void) mutex_lock(&lock);

        /* Init semaphore for scratch zones. */
        if (sema_init(&scratch_sem, 0, USYNC_THREAD, NULL) == -1) {
                zerror(zlogp, B_TRUE,
                    "failed to initialize semaphore for scratch zone");
                goto child_out;
        }

        /* open the dladm handle */
        if (dladm_open(&dld_handle) != DLADM_STATUS_OK) {
                zerror(zlogp, B_FALSE, "failed to open dladm handle");
                goto child_out;
        }

        /*
         * Note: door setup must occur *after* the console is setup.
         * This is so that as zlogin tests the door to see if zoneadmd
         * is ready yet, we know that the console will get serviced
         * once door_info() indicates that the door is "up".
         */
        if (setup_door(zlogp) == -1)
                goto child_out;

        /*
```

```
         * Things seem OK so far; tell the parent process that we’re done
         * with setup tasks.  This will cause the parent to exit, signalling
         * to zoneadm, zlogin, or whatever forked it that we are ready to
         * service requests.
         */
        shstate->status = 0;
        (void) sema_post(&shstate->sem);
        (void) munmap((char *)shstate, shstatelen);
        shstate = NULL;

        (void) mutex_unlock(&lock);

        /*
         * zlogp is now invalid, so reset it to the syslog logger.
         */
        zlogp = &logsys;

        /*
         * Now that we are free of any parents, switch to the default locale.
         */
        (void) setlocale(LC_ALL, DEFAULT_LOCALE);

        /*
         * At this point the setup portion of main() is basically done, so
         * we reuse this thread to manage the zone console.  When
         * serve_console() has returned, we are past the point of no return
         * in the life of this zoneadmd.
         */
        if (zonecfg_in_alt_root()) {
                /*
                 * This is just awful, but mounted scratch zones don’t (and
                 * can’t) have consoles.  We just wait for unmount instead.
                 */
                while (sema_wait(&scratch_sem) == EINTR)
                        ;
        } else {
                serve_console(zlogp);
                assert(in_death_throes);
        }

        /*
         * This is the next-to-last part of the exit interlock.  Upon calling
         * fdetach(), the door will go unreferenced; once any
         * outstanding requests (like the door thread doing Z_HALT) are
         * done, the door will get an UNREF notification; when it handles
         * the UNREF, the door server will cause the exit.  It’s possible
         * that fdetach() can fail because the file is in use, in which
         * case we’ll retry the operation.
         */
        assert(!MUTEX_HELD(&lock));
        for (;;) {
                if ((fdetach(zone_door_path) == 0) || (errno != EBUSY))
                        break;
                yield();
        }

        for (;;)
                (void) pause();

child_out:
        assert(pid == 0);

        if (shstate != NULL) {
        shstate->status = -1;
        (void) sema_post(&shstate->sem);
        (void) munmap((char *)shstate, shstatelen);
```

```
        }

        /*
         * This might trigger an unref notification, but if so,
         * we are still holding the lock, so our call to exit will
         * ultimately win the race and will publish the right exit
         * code.
         */
        if (zone_door != -1) {
                assert(MUTEX_HELD(&lock));
                (void) door_revoke(zone_door);
                (void) fdetach(zone_door_path);
        }

        if (dld_handle != NULL)
                dladm_close(dld_handle);

        return (1); /* return from main() forcibly exits an MT process */
}
```

______unchanged_portion_omitted_