```
**********************************************************
    3734 Thu Jun 14 17:15:57 2018
new/usr/src/pkg/manifests/system-test-ostest.mf
9600 LDT still not happy under KPTI
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright (c) 2012, 2016 by Delphix. All rights reserved.
  14 # Copyright 2014, OmniTI Computer Consulting, Inc. All rights reserved.
  15 # Copyright 2018 Joyent, Inc.
  15 # Copyright 2017 Joyent, Inc.
  16 #

  18 set name=pkg.fmri value=pkg:/system/test/ostest@$(PKGVERS)
  19 set name=pkg.description value="Miscellaneous OS Unit Tests"
  20 set name=pkg.summary value="OS Unit Test Suite"
  21 set name=info.classification \
  22     value=org.opensolaris.category.2008:Development/System
  23 set name=variant.arch value=$(ARCH)
  24 dir path=opt/os-tests
  25 dir path=opt/os-tests/bin
  26 dir path=opt/os-tests/runfiles
  27 dir path=opt/os-tests/tests
  28 dir path=opt/os-tests/tests/file-locking
  29 $(i386_ONLY)dir path=opt/os-tests/tests/i386
  30 dir path=opt/os-tests/tests/pf_key
  31 dir path=opt/os-tests/tests/sdevfs
  32 dir path=opt/os-tests/tests/secflags
  33 dir path=opt/os-tests/tests/sigqueue
  34 dir path=opt/os-tests/tests/sockfs
  35 dir path=opt/os-tests/tests/stress
  36 file path=opt/os-tests/README mode=0444
  37 file path=opt/os-tests/bin/ostest mode=0555
  38 file path=opt/os-tests/runfiles/default.run mode=0444
  39 file path=opt/os-tests/tests/epoll_test mode=0555
  40 file path=opt/os-tests/tests/file-locking/acquire-lock.32 mode=0555
  41 file path=opt/os-tests/tests/file-locking/acquire-lock.64 mode=0555
  42 file path=opt/os-tests/tests/file-locking/runtests.32 mode=0555
  43 file path=opt/os-tests/tests/file-locking/runtests.64 mode=0555
  44 $(i386_ONLY)file path=opt/os-tests/tests/i386/ldt mode=0555
  45 file path=opt/os-tests/tests/pf_key/acquire-compare mode=0555
  46 file path=opt/os-tests/tests/pf_key/acquire-spray mode=0555
  47 file path=opt/os-tests/tests/pf_key/eacq-enabler mode=0555
  48 file path=opt/os-tests/tests/pf_key/kmc-update mode=0555
  49 file path=opt/os-tests/tests/pf_key/kmc-updater mode=0555
  50 file path=opt/os-tests/tests/poll_test mode=0555
  51 file path=opt/os-tests/tests/sdevfs/sdevfs_eisdir mode=0555
  52 file path=opt/os-tests/tests/secflags/addrs-32 mode=0555
  53 file path=opt/os-tests/tests/secflags/addrs-64 mode=0555
  54 file path=opt/os-tests/tests/secflags/secflags_aslr mode=0555
  55 file path=opt/os-tests/tests/secflags/secflags_core mode=0555
  56 file path=opt/os-tests/tests/secflags/secflags_dts mode=0555
  57 file path=opt/os-tests/tests/secflags/secflags_elfdump mode=0555
  58 file path=opt/os-tests/tests/secflags/secflags_forbidnullmap mode=0555
  59 file path=opt/os-tests/tests/secflags/secflags_limits mode=0555
  60 file path=opt/os-tests/tests/secflags/secflags_noexecstack mode=0555
```



```
  61 file path=opt/os-tests/tests/secflags/secflags_proc mode=0555
  62 file path=opt/os-tests/tests/secflags/secflags_psecflags mode=0555
  63 file path=opt/os-tests/tests/secflags/secflags_syscall mode=0555
  64 file path=opt/os-tests/tests/secflags/secflags_truss mode=0555
  65 file path=opt/os-tests/tests/secflags/secflags_zonecfg mode=0555
  66 file path=opt/os-tests/tests/secflags/stacky mode=0555
  67 file path=opt/os-tests/tests/sigqueue/sigqueue_queue_size mode=0555
  68 file path=opt/os-tests/tests/sockfs/conn mode=0555
  69 file path=opt/os-tests/tests/sockfs/dgram mode=0555
  70 file path=opt/os-tests/tests/sockfs/drop_priv mode=0555
  71 file path=opt/os-tests/tests/sockfs/nosignal mode=0555
  72 file path=opt/os-tests/tests/sockfs/sockpair mode=0555
  73 file path=opt/os-tests/tests/spoof-ras mode=0555
  74 file path=opt/os-tests/tests/stress/dladm-kstat mode=0555
  75 license cr_Sun license=cr_Sun
  76 license lic_CDDL license=lic_CDDL
  77 depend fmri=pkg:/network/telnet type=require
  78 depend fmri=system/test/testrunner type=require
```

```
**********************************************************
    1595 Thu Jun 14 17:15:57 2018
new/usr/src/test/os-tests/runfiles/default.run
9600 LDT still not happy under KPTI
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2018 Joyent, Inc.
# Copyright 2017 Joyent, Inc.
#

[DEFAULT]
pre =
verbose = False
quiet = False
timeout = 60
post =
outputdir = /var/tmp/test_results

[/opt/os-tests/tests/poll_test]
user = root
tests = ['poll_test', 'epoll_test']

[/opt/os-tests/tests/secflags]
user = root
tests = ['secflags_aslr',
         'secflags_core',
         'secflags_dts',
         'secflags_elfdump',
         'secflags_forbidnullmap',
         'secflags_limits',
         'secflags_noexecstack',
         'secflags_proc',
         'secflags_psecflags',
         'secflags_syscall',
         'secflags_truss',
         'secflags_zonecfg']

[/opt/os-tests/tests/sigqueue]
tests = ['sigqueue_queue_size']

[/opt/os-tests/tests/sdevfs]
user = root
tests = ['sdevfs_eisdir']

[/opt/os-tests/tests/stress]
user = root
tests = ['dladm-kstat']

[/opt/os-tests/tests/file-locking]
tests = ['runtests.32', 'runtests.64']

[/opt/os-tests/tests/sockfs]
user = root
tests = ['conn', 'dgram', 'drop_priv', 'nosignal', 'sockpair']
```



```
[/opt/os-tests/tests/pf_key]
user = root
tests = ['acquire-compare', 'acquire-spray']

[/opt/os-tests/tests/i386]
user = root
arch = i86pc
tests = ['ldt']
```

```
**********************************************************
     659 Thu Jun 14 17:15:58 2018
new/usr/src/test/os-tests/tests/Makefile
9600 LDT still not happy under KPTI
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2012, 2016 by Delphix. All rights reserved.
# Copyright 2018 Joyent, Inc.
# Copyright 2017 Joyent, Inc.
#

SUBDIRS_i386 = i386

SUBDIRS = poll secflags sigqueue spoof-ras sdevfs sockfs stress file-locking \
        pf_key $(SUBDIRS_$(MACH))
        pf_key

include $(SRC)/test/Makefile.com
```

```
**********************************************************
     833 Thu Jun 14 17:15:58 2018
new/usr/src/test/os-tests/tests/i386/Makefile
9600 LDT still not happy under KPTI
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2018 Joyent, Inc.
#

include $(SRC)/cmd/Makefile.cmd
include $(SRC)/test/Makefile.com

PROG += ldt

ROOTOPTPKG = $(ROOT)/opt/os-tests
TESTDIR = $(ROOTOPTPKG)/tests/i386

CSTD = $(CSTD_GNU99)

CMDS = $(PROG:%=$(TESTDIR)/%)
$(CMDS) := FILEMODE = 0555

all: $(PROG)

install: all $(CMDS)

lint:

clobber: clean
        -$(RM) $(PROG)

clean:

$(CMDS): $(TESTDIR) $(PROG)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)
```

```
**********************************************************
    1672 Thu Jun 14 17:15:58 2018
new/usr/src/test/os-tests/tests/i386/ldt.c
9600 LDT still not happy under KPTI
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2018 Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/sysi86.h>
#include <sys/segments.h>
#include <sys/segment.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <pthread.h>
#include <err.h>

char foo[4096];

static void *
donothing(void *nothing)
{
        sleep(5);
        return (NULL);
}

int
main(void)
{
        pthread_t tid;

        /*
         * This first is similar to what sbcl does in some variants.  Note the
         * SDT_MEMRW (not SDT_MEMRWA) so we check that the kernel is forcing the
         * 'accessed' bit too.
         */
        int sel = SEL_LDT(7);

        struct ssd ssd = { sel, (unsigned long)&foo, 4096,
            SDT_MEMRW | (SEL_UPL << 5) | (1 << 7), 0x4 };

        if (sysi86(SI86DSCR, &ssd) < 0)
                err(-1, "failed to setup segment");

        __asm__ __volatile__("mov %0, %%fs" : : "r" (sel));

        ssd.acc1 = 0;

        if (sysi86(SI86DSCR, &ssd) == 0)
                errx(-1, "removed in-use segment?");

        __asm__ __volatile__("mov %0, %%fs" : : "r" (0));
```



```
        if (sysi86(SI86DSCR, &ssd) < 0)
                err(-1, "failed to remove segment");

        for (int i = 0; i < MAXNLDT; i++) {
                ssd.sel = SEL_LDT(i);
                (void) sysi86(SI86DSCR, &ssd);
        }

        for (int i = 0; i < 10; i++)
                pthread_create(&tid, NULL, donothing, NULL);

        if (forkall() == 0) {
                sleep(2);
                _exit(0);
        }

        sleep(6);
        return (0);
}
```

```
**********************************************************
   32392 Thu Jun 14 17:15:58 2018
new/usr/src/test/test-runner/cmd/run
9600 LDT still not happy under KPTI
**********************************************************
```

```
   1 #!@PYTHON@

   3 #
   4 # This file and its contents are supplied under the terms of the
   5 # Common Development and Distribution License ("CDDL"), version 1.0.
   6 # You may only use this file in accordance with the terms of version
   7 # 1.0 of the CDDL.
   8 #
   9 # A full copy of the text of the CDDL should have accompanied this
  10 # source.  A copy of the CDDL is also available via the Internet at
  11 # http://www.illumos.org/license/CDDL.
  12 #

  14 #
  15 # Copyright (c) 2012, 2016 by Delphix. All rights reserved.
  16 # Copyright (c) 2017, Chris Fraire <cfraire@me.com>.
  17 # Copyright 2018 Joyent, Inc.
  18 #

  20 import ConfigParser
  21 import os
  22 import logging
  23 import platform
  24 from logging.handlers import WatchedFileHandler
  25 from datetime import datetime
  26 from optparse import OptionParser
  27 from pwd import getpwnam
  28 from pwd import getpwuid
  29 from select import select
  30 from subprocess import PIPE
  31 from subprocess import Popen
  32 from sys import argv
  33 from sys import maxint
  34 from threading import Timer
  35 from time import time

  37 BASEDIR = '/var/tmp/test_results'
  38 KILL = '/usr/bin/kill'
  39 TRUE = '/usr/bin/true'
  40 SUDO = '/usr/bin/sudo'

  42 # Custom class to reopen the log file in case it is forcibly closed by a test.
  43 class WatchedFileHandlerClosed(WatchedFileHandler):
  44     """Watch files, including closed files.
  45     Similar to (and inherits from) logging.handler.WatchedFileHandler,
  46     except that IOErrors are handled by reopening the stream and retrying.
  47     This will be retried up to a configurable number of times before
  48     giving up, default 5.
  49     """

  51     def __init__(self, filename, mode='a', encoding=None, delay=0, max_tries=5):
  52         self.max_tries = max_tries
  53         self.tries = 0
  54         WatchedFileHandler.__init__(self, filename, mode, encoding, delay)

  56     def emit(self, record):
  57         while True:
  58             try:
  59                 WatchedFileHandler.emit(self, record)
  60                 self.tries = 0
  61                 return
```



```
  62             except IOError as err:
  63                 if self.tries == self.max_tries:
  64                     raise
  65                 self.stream.close()
  66                 self.stream = self._open()
  67                 self.tries += 1

  69 class Result(object):
  70     total = 0
  71     runresults = {'PASS': 0, 'FAIL': 0, 'SKIP': 0, 'KILLED': 0}

  73     def __init__(self):
  74         self.starttime = None
  75         self.returncode = None
  76         self.runtime = ''
  77         self.stdout = []
  78         self.stderr = []
  79         self.result = ''

  81     def done(self, proc, killed):
  82         """
  83         Finalize the results of this Cmd.
  84         """
  85         Result.total += 1
  86         m, s = divmod(time() - self.starttime, 60)
  87         self.runtime = '%02d:%02d' % (m, s)
  88         self.returncode = proc.returncode
  89         if killed:
  90             self.result = 'KILLED'
  91             Result.runresults['KILLED'] += 1
  92         elif self.returncode is 0:
  93             self.result = 'PASS'
  94             Result.runresults['PASS'] += 1
  95         elif self.returncode is not 0:
  96             self.result = 'FAIL'
  97             Result.runresults['FAIL'] += 1


 100 class Output(object):
 101     """
 102     This class is a slightly modified version of the 'Stream' class found
 103     here: http://goo.gl/aSGfv
 104     """
 105     def __init__(self, stream):
 106         self.stream = stream
 107         self._buf = ''
 108         self.lines = []

 110     def fileno(self):
 111         return self.stream.fileno()

 113     def read(self, drain=0):
 114         """
 115         Read from the file descriptor. If 'drain' set, read until EOF.
 116         """
 117         while self._read() is not None:
 118             if not drain:
 119                 break

 121     def _read(self):
 122         """
 123         Read up to 4k of data from this output stream. Collect the output
 124         up to the last newline, and append it to any leftover data from a
 125         previous call. The lines are stored as a (timestamp, data) tuple
 126         for easy sorting/merging later.
 127         """
```

```
        fd = self.fileno()
        buf = os.read(fd, 4096)
        if not buf:
            return None
        if '\n' not in buf:
            self._buf += buf
            return []

        buf = self._buf + buf
        tmp, rest = buf.rsplit('\n', 1)
        self._buf = rest
        now = datetime.now()
        rows = tmp.split('\n')
        self.lines += [(now, r) for r in rows]


class Cmd(object):
    verified_users = []

    def __init__(self, pathname, outputdir=None, timeout=None, user=None):
        self.pathname = pathname
        self.outputdir = outputdir or 'BASEDIR'
        self.timeout = timeout
        self.user = user or ''
        self.killed = False
        self.result = Result()

        if self.timeout is None:
            self.timeout = 60

    def __str__(self):
        return "Pathname: %s\nOutputdir: %s\nTimeout: %s\nUser: %s\n" % \
            (self.pathname, self.outputdir, self.timeout, self.user)

    def kill_cmd(self, proc):
        """
        Kill a running command due to timeout, or ^C from the keyboard. If
        sudo is required, this user was verified previously.
        """
        self.killed = True
        do_sudo = len(self.user) != 0
        signal = '-TERM'

        cmd = [SUDO, KILL, signal, str(proc.pid)]
        if not do_sudo:
            del cmd[0]

        try:
            kp = Popen(cmd)
            kp.wait()
        except:
            pass

    def update_cmd_privs(self, cmd, user):
        """
        If a user has been specified to run this Cmd and we're not already
        running as that user, prepend the appropriate sudo command to run
        as that user.
        """
        me = getpwuid(os.getuid())

        if not user or user is me:
            return cmd

        ret = '%s -E -u %s %s' % (SUDO, user, cmd)
        return ret.split(' ')
```



```
    def collect_output(self, proc):
        """
        Read from stdout/stderr as data becomes available, until the
        process is no longer running. Return the lines from the stdout and
        stderr Output objects.
        """
        out = Output(proc.stdout)
        err = Output(proc.stderr)
        res = []
        while proc.returncode is None:
            proc.poll()
            res = select([out, err], [], [], .1)
            for fd in res[0]:
                fd.read()
        for fd in res[0]:
            fd.read(drain=1)

        return out.lines, err.lines

    def run(self, options):
        """
        This is the main function that runs each individual test.
        Determine whether or not the command requires sudo, and modify it
        if needed. Run the command, and update the result object.
        """
        if options.dryrun is True:
            print self
            return

        privcmd = self.update_cmd_privs(self.pathname, self.user)
        try:
            old = os.umask(0)
            if not os.path.isdir(self.outputdir):
                os.makedirs(self.outputdir, mode=0777)
            os.umask(old)
        except OSError, e:
            fail('%s' % e)

        try:
            self.result.starttime = time()
            proc = Popen(privcmd, stdout=PIPE, stderr=PIPE, stdin=PIPE)
            proc.stdin.close()

            # Allow a special timeout value of 0 to mean infinity
            if int(self.timeout) == 0:
                self.timeout = maxint
            t = Timer(int(self.timeout), self.kill_cmd, [proc])
            t.start()
            self.result.stdout, self.result.stderr = self.collect_output(proc)
        except KeyboardInterrupt:
            self.kill_cmd(proc)
            fail('\nRun terminated at user request.')
        finally:
            t.cancel()

        self.result.done(proc, self.killed)

    def skip(self):
        """
        Initialize enough of the test result that we can log a skipped
        command.
        """
        Result.total += 1
        Result.runresults['SKIP'] += 1
        self.result.stdout = self.result.stderr = []
```

```
260         self.result.starttime = time()
261         m, s = divmod(time() - self.result.starttime, 60)
262         self.result.runtime = '%02d:%02d' % (m, s)
263         self.result.result = 'SKIP'

265     def log(self, logger, options):
266         """
267         This function is responsible for writing all output. This includes
268         the console output, the logfile of all results (with timestamped
269         merged stdout and stderr), and for each test, the unmodified
270         stdout/stderr/merged in it's own file.
271         """
272         if logger is None:
273             return

275         logname = getpwuid(os.getuid()).pw_name
276         user = ' (run as %s)' % (self.user if len(self.user) else logname)
277         msga = 'Test: %s%s ' % (self.pathname, user)
278         msgb = '[%s] [%s]' % (self.result.runtime, self.result.result)
279         pad = ' ' * (80 - (len(msga) + len(msgb)))

281         # If -q is specified, only print a line for tests that didn't pass.
282         # This means passing tests need to be logged as DEBUG, or the one
283         # line summary will only be printed in the logfile for failures.
284         if not options.quiet:
285             logger.info('%s%s%s' % (msga, pad, msgb))
286         elif self.result.result is not 'PASS':
287             logger.info('%s%s%s' % (msga, pad, msgb))
288         else:
289             logger.debug('%s%s%s' % (msga, pad, msgb))

291         lines = sorted(self.result.stdout + self.result.stderr,
292                        cmp=lambda x, y: cmp(x[0], y[0]))

294         for dt, line in lines:
295             logger.debug('%s %s' % (dt.strftime("%H:%M:%S.%f ")[:11], line))

297         if len(self.result.stdout):
298             with open(os.path.join(self.outputdir, 'stdout'), 'w') as out:
299                 for _, line in self.result.stdout:
300                     os.write(out.fileno(), '%s\n' % line)
301         if len(self.result.stderr):
302             with open(os.path.join(self.outputdir, 'stderr'), 'w') as err:
303                 for _, line in self.result.stderr:
304                     os.write(err.fileno(), '%s\n' % line)
305         if len(self.result.stdout) and len(self.result.stderr):
306             with open(os.path.join(self.outputdir, 'merged'), 'w') as merged:
307                 for _, line in lines:
308                     os.write(merged.fileno(), '%s\n' % line)


311 class Test(Cmd):
312     props = ['outputdir', 'timeout', 'user', 'pre', 'pre_user', 'post',
313              'post_user']

315     def __init__(self, pathname, outputdir=None, timeout=None, user=None,
316                  pre=None, pre_user=None, post=None, post_user=None):
317         super(Test, self).__init__(pathname, outputdir, timeout, user)
318         self.pre = pre or ''
319         self.pre_user = pre_user or ''
320         self.post = post or ''
321         self.post_user = post_user or ''

323     def __str__(self):
324         post_user = pre_user = ''
325         if len(self.pre_user):
```



```
326             pre_user = ' (as %s)' % (self.pre_user)
327         if len(self.post_user):
328             post_user = ' (as %s)' % (self.post_user)
329         return "Pathname: %s\nOutputdir: %s\nTimeout: %d\nPre: %s%s\nPost: " \
330                "%s%s\nUser: %s\n" % \
331                (self.pathname, self.outputdir, self.timeout, self.pre,
332                 pre_user, self.post, post_user, self.user)

334     def verify(self, logger):
335         """
336         Check the pre/post scripts, user and Test. Omit the Test from this
337         run if there are any problems.
338         """
339         files = [self.pre, self.pathname, self.post]
340         users = [self.pre_user, self.user, self.post_user]

342         for f in [f for f in files if len(f)]:
343             if not verify_file(f):
344                 logger.info("Warning: Test '%s' not added to this run because"
345                             " it failed verification." % f)
346                 return False

348         for user in [user for user in users if len(user)]:
349             if not verify_user(user, logger):
350                 logger.info("Not adding Test '%s' to this run." %
351                             self.pathname)
352                 return False

354         return True

356     def run(self, logger, options):
357         """
358         Create Cmd instances for the pre/post scripts. If the pre script
359         doesn't pass, skip this Test. Run the post script regardless.
360         """
361         odir = os.path.join(self.outputdir, os.path.basename(self.pre))
362         pretest = Cmd(self.pre, outputdir=odir, timeout=self.timeout,
363                       user=self.pre_user)
364         test = Cmd(self.pathname, outputdir=self.outputdir,
365                    timeout=self.timeout, user=self.user)
366         odir = os.path.join(self.outputdir, os.path.basename(self.post))
367         posttest = Cmd(self.post, outputdir=odir, timeout=self.timeout,
368                        user=self.post_user)

370         cont = True
371         if len(pretest.pathname):
372             pretest.run(options)
373             cont = pretest.result.result is 'PASS'
374             pretest.log(logger, options)

376         if cont:
377             test.run(options)
378         else:
379             test.skip()

381         test.log(logger, options)

383         if len(posttest.pathname):
384             posttest.run(options)
385             posttest.log(logger, options)


388 class TestGroup(Test):
389     props = Test.props + ['tests']

391     def __init__(self, pathname, outputdir=None, timeout=None, user=None,
```

```
392                 pre=None, pre_user=None, post=None, post_user=None,
393                 tests=None):
394         super(TestGroup, self).__init__(pathname, outputdir, timeout, user,
395                                         pre, pre_user, post, post_user)
396         self.tests = tests or []

398     def __str__(self):
399         post_user = pre_user = ''
400         if len(self.pre_user):
401             pre_user = ' (as %s)' % (self.pre_user)
402         if len(self.post_user):
403             post_user = ' (as %s)' % (self.post_user)
404         return "Pathname: %s\nOutputdir: %s\nTests: %s\nTimeout: %d\n" \
405                "Pre: %s%s\nPost: %s%s\nUser: %s\n" % \
406                (self.pathname, self.outputdir, self.tests, self.timeout,
407                 self.pre, pre_user, self.post, post_user, self.user)

409     def verify(self, logger):
410         """
411         Check the pre/post scripts, user and tests in this TestGroup. Omit
412         the TestGroup entirely, or simply delete the relevant tests in the
413         group, if that's all that's required.
414         """
415         # If the pre or post scripts are relative pathnames, convert to
416         # absolute, so they stand a chance of passing verification.
417         if len(self.pre) and not os.path.isabs(self.pre):
418             self.pre = os.path.join(self.pathname, self.pre)
419         if len(self.post) and not os.path.isabs(self.post):
420             self.post = os.path.join(self.pathname, self.post)

422         auxfiles = [self.pre, self.post]
423         users = [self.pre_user, self.user, self.post_user]

425         for f in [f for f in auxfiles if len(f)]:
426             if self.pathname != os.path.dirname(f):
427                 logger.info("Warning: TestGroup '%s' not added to this run. "
428                             "Auxiliary script '%s' exists in a different "
429                             "directory." % (self.pathname, f))
430                 return False

432             if not verify_file(f):
433                 logger.info("Warning: TestGroup '%s' not added to this run. "
434                             "Auxiliary script '%s' failed verification." %
435                             (self.pathname, f))
436                 return False

438         for user in [user for user in users if len(user)]:
439             if not verify_user(user, logger):
440                 logger.info("Not adding TestGroup '%s' to this run." %
441                             self.pathname)
442                 return False

444         # If one of the tests is invalid, delete it, log it, and drive on.
445         self.tests[:] = [f for f in self.tests if
446           verify_file(os.path.join(self.pathname, f))]

448         return len(self.tests) is not 0

450     def run(self, logger, options):
451         """
452         Create Cmd instances for the pre/post scripts. If the pre script
453         doesn't pass, skip all the tests in this TestGroup. Run the post
454         script regardless.
455         """
456         odir = os.path.join(self.outputdir, os.path.basename(self.pre))
457         pretest = Cmd(self.pre, outputdir=odir, timeout=self.timeout,
```



```
458                       user=self.pre_user)
459         odir = os.path.join(self.outputdir, os.path.basename(self.post))
460         posttest = Cmd(self.post, outputdir=odir, timeout=self.timeout,
461                        user=self.post_user)

463         cont = True
464         if len(pretest.pathname):
465             pretest.run(options)
466             cont = pretest.result.result is 'PASS'
467             pretest.log(logger, options)

469         for fname in self.tests:
470             test = Cmd(os.path.join(self.pathname, fname),
471                        outputdir=os.path.join(self.outputdir, fname),
472                        timeout=self.timeout, user=self.user)
473             if cont:
474                 test.run(options)
475             else:
476                 test.skip()

478             test.log(logger, options)

480         if len(posttest.pathname):
481             posttest.run(options)
482             posttest.log(logger, options)


485 class TestRun(object):
486     props = ['quiet', 'outputdir']

488     def __init__(self, options):
489         self.tests = {}
490         self.testgroups = {}
491         self.starttime = time()
492         self.timestamp = datetime.now().strftime('%Y%m%dT%H%M%S')
493         self.outputdir = os.path.join(options.outputdir, self.timestamp)
494         self.logger = self.setup_logging(options)
495         self.defaults = [
496             ('outputdir', BASEDIR),
497             ('quiet', False),
498             ('timeout', 60),
499             ('user', ''),
500             ('pre', ''),
501             ('pre_user', ''),
502             ('post', ''),
503             ('post_user', '')
504         ]

506     def __str__(self):
507         s = 'TestRun:\n    outputdir: %s\n' % self.outputdir
508         s += 'TESTS:\n'
509         for key in sorted(self.tests.keys()):
510             s += '%s%s' % (self.tests[key].__str__(), '\n')
511         s += 'TESTGROUPS:\n'
512         for key in sorted(self.testgroups.keys()):
513             s += '%s%s' % (self.testgroups[key].__str__(), '\n')
514         return s

516     def addtest(self, pathname, options):
517         """
518         Create a new Test, and apply any properties that were passed in
519         from the command line. If it passes verification, add it to the
520         TestRun.
521         """
522         test = Test(pathname)
523         for prop in Test.props:
```

```
524             setattr(test, prop, getattr(options, prop))
525
526         if test.verify(self.logger):
527             self.tests[pathname] = test
528
529     def addtestgroup(self, dirname, filenames, options):
530         """
531         Create a new TestGroup, and apply any properties that were passed
532         in from the command line. If it passes verification, add it to the
533         TestRun.
534         """
535         if dirname not in self.testgroups:
536             testgroup = TestGroup(dirname)
537             for prop in Test.props:
538                 setattr(testgroup, prop, getattr(options, prop))
539
540             # Prevent pre/post scripts from running as regular tests
541             for f in [testgroup.pre, testgroup.post]:
542                 if f in filenames:
543                     del filenames[filenames.index(f)]
544
545             self.testgroups[dirname] = testgroup
546             self.testgroups[dirname].tests = sorted(filenames)
547
548             testgroup.verify(self.logger)
549
550     def read(self, logger, options):
551         """
552         Read in the specified runfile, and apply the TestRun properties
553         listed in the 'DEFAULT' section to our TestRun. Then read each
554         section, and apply the appropriate properties to the Test or
555         TestGroup. Properties from individual sections override those set
556         in the 'DEFAULT' section. If the Test or TestGroup passes
557         verification, add it to the TestRun.
558         """
559         config = ConfigParser.RawConfigParser()
560         if not len(config.read(options.runfile)):
561             fail("Coulnd't read config file %s" % options.runfile)
562
563         for opt in TestRun.props:
564             if config.has_option('DEFAULT', opt):
565                 setattr(self, opt, config.get('DEFAULT', opt))
566         self.outputdir = os.path.join(self.outputdir, self.timestamp)
567
568         for section in config.sections():
569             if ('arch' in config.options(section) and
570                 platform.machine() != config.get(section, 'arch')):
571                 continue
572
573             if 'tests' in config.options(section):
574                 testgroup = TestGroup(section)
575                 for prop in TestGroup.props:
576                     for sect in ['DEFAULT', section]:
577                         if config.has_option(sect, prop):
578                             setattr(testgroup, prop, config.get(sect, prop))
579
580                 # Repopulate tests using eval to convert the string to a list
581                 testgroup.tests = eval(config.get(section, 'tests'))
582
583                 if testgroup.verify(logger):
584                     self.testgroups[section] = testgroup
585
586             elif 'autotests' in config.options(section):
587                 testgroup = TestGroup(section)
588                 for prop in TestGroup.props:
589                     for sect in ['DEFAULT', section]:
```



```
590                         if config.has_option(sect, prop):
591                             setattr(testgroup, prop, config.get(sect, prop))
592
593                 filenames = os.listdir(section)
594                 # only files starting with "tst." are considered tests
595                 filenames = [f for f in filenames if f.startswith("tst.")]
596                 testgroup.tests = sorted(filenames)
597
598                 if testgroup.verify(logger):
599                     self.testgroups[section] = testgroup
600
601             else:
602                 test = Test(section)
603                 for prop in Test.props:
604                     for sect in ['DEFAULT', section]:
605                         if config.has_option(sect, prop):
606                             setattr(test, prop, config.get(sect, prop))
607
608                 if test.verify(logger):
609                     self.tests[section] = test
610
611     def write(self, options):
612         """
613         Create a configuration file for editing and later use. The
614         'DEFAULT' section of the config file is created from the
615         properties that were specified on the command line. Tests are
616         simply added as sections that inherit everything from the
617         'DEFAULT' section. TestGroups are the same, except they get an
618         option including all the tests to run in that directory.
619         """
620
621         defaults = dict([(prop, getattr(options, prop)) for prop, _ in
622                          self.defaults])
623         config = ConfigParser.RawConfigParser(defaults)
624
625         for test in sorted(self.tests.keys()):
626             config.add_section(test)
627
628         for testgroup in sorted(self.testgroups.keys()):
629             config.add_section(testgroup)
630             config.set(testgroup, 'tests', self.testgroups[testgroup].tests)
631
632         try:
633             with open(options.template, 'w') as f:
634                 return config.write(f)
635         except IOError:
636             fail('Could not open \'%s\' for writing.' % options.template)
637
638     def complete_outputdirs(self):
639         """
640         Collect all the pathnames for Tests, and TestGroups. Work
641         backwards one pathname component at a time, to create a unique
642         directory name in which to deposit test output. Tests will be able
643         to write output files directly in the newly modified outputdir.
644         TestGroups will be able to create one subdirectory per test in the
645         outputdir, and are guaranteed uniqueness because a group can only
646         contain files in one directory. Pre and post tests will create a
647         directory rooted at the outputdir of the Test or TestGroup in
648         question for their output.
649         """
650         done = False
651         components = 0
652         tmp_dict = dict(self.tests.items() + self.testgroups.items())
653         total = len(tmp_dict)
654         base = self.outputdir
```

```
656         while not done:
657             l = []
658             components -= 1
659             for testfile in tmp_dict.keys():
660                 uniq = '/'.join(testfile.split('/')[components:]).lstrip('/')
661                 if uniq not in l:
662                     l.append(uniq)
663                     tmp_dict[testfile].outputdir = os.path.join(base, uniq)
664                 else:
665                     break
666             done = total == len(l)

668     def setup_logging(self, options):
669         """
670         Two loggers are set up here. The first is for the logfile which
671         will contain one line summarizing the test, including the test
672         name, result, and running time. This logger will also capture the
673         timestamped combined stdout and stderr of each run. The second
674         logger is optional console output, which will contain only the one
675         line summary. The loggers are initialized at two different levels
676         to facilitate segregating the output.
677         """
678         if options.dryrun is True:
679             return

681         testlogger = logging.getLogger(__name__)
682         testlogger.setLevel(logging.DEBUG)

684         if options.cmd is not 'wrconfig':
685             try:
686                 old = os.umask(0)
687                 os.makedirs(self.outputdir, mode=0777)
688                 os.umask(old)
689             except OSError, e:
690                 fail('%s' % e)
691             filename = os.path.join(self.outputdir, 'log')

693             logfile = WatchedFileHandlerClosed(filename)
694             logfile.setLevel(logging.DEBUG)
695             logfilefmt = logging.Formatter('%(message)s')
696             logfile.setFormatter(logfilefmt)
697             testlogger.addHandler(logfile)

699         cons = logging.StreamHandler()
700         cons.setLevel(logging.INFO)
701         consfmt = logging.Formatter('%(message)s')
702         cons.setFormatter(consfmt)
703         testlogger.addHandler(cons)

705         return testlogger

707     def run(self, options):
708         """
709         Walk through all the Tests and TestGroups, calling run().
710         """
711         if not options.dryrun:
712             try:
713                 os.chdir(self.outputdir)
714             except OSError:
715                 fail('Could not change to directory %s' % self.outputdir)
716         for test in sorted(self.tests.keys()):
717             self.tests[test].run(self.logger, options)
718         for testgroup in sorted(self.testgroups.keys()):
719             self.testgroups[testgroup].run(self.logger, options)

721     def summary(self):
```



```
722         if Result.total is 0:
723             return

725         print '\nResults Summary'
726         for key in Result.runresults.keys():
727             if Result.runresults[key] is not 0:
728                 print '%s\t% 4d' % (key, Result.runresults[key])

730         m, s = divmod(time() - self.starttime, 60)
731         h, m = divmod(m, 60)
732         print '\nRunning Time:\t%02d:%02d:%02d' % (h, m, s)
733         print 'Percent passed:\t%.1f%%' % ((float(Result.runresults['PASS']) /
734                                             float(Result.total)) * 100)
735         print 'Log directory:\t%s' % self.outputdir


738 def verify_file(pathname):
739     """
740     Verify that the supplied pathname is an executable regular file.
741     """
742     if os.path.isdir(pathname) or os.path.islink(pathname):
743         return False

745     if os.path.isfile(pathname) and os.access(pathname, os.X_OK):
746         return True

748     return False


751 def verify_user(user, logger):
752     """
753     Verify that the specified user exists on this system, and can execute
754     sudo without being prompted for a password.
755     """
756     testcmd = [SUDO, '-n', '-u', user, TRUE]

758     if user in Cmd.verified_users:
759         return True

761     try:
762         _ = getpwnam(user)
763     except KeyError:
764         logger.info("Warning: user '%s' does not exist.", user)
765         return False

767     p = Popen(testcmd)
768     p.wait()
769     if p.returncode is not 0:
770         logger.info("Warning: user '%s' cannot use passwordless sudo.", user)
771         return False
772     else:
773         Cmd.verified_users.append(user)

775     return True


778 def find_tests(testrun, options):
779     """
780     For the given list of pathnames, add files as Tests. For directories,
781     if do_groups is True, add the directory as a TestGroup. If False,
782     recursively search for executable files.
783     """

785     for p in sorted(options.pathnames):
786         if os.path.isdir(p):
787             for dirname, _, filenames in os.walk(p):
```

```
                if options.do_groups:
                    testrun.addtestgroup(dirname, filenames, options)
                else:
                    for f in sorted(filenames):
                        testrun.addtest(os.path.join(dirname, f), options)
        else:
            testrun.addtest(p, options)


def fail(retstr, ret=1):
    print '%s: %s' % (argv[0], retstr)
    exit(ret)


def options_cb(option, opt_str, value, parser):
    path_options = ['runfile', 'outputdir', 'template']

    if option.dest is 'runfile' and '-w' in parser.rargs or \
            option.dest is 'template' and '-c' in parser.rargs:
        fail('-c and -w are mutually exclusive.')

    if opt_str in parser.rargs:
        fail('%s may only be specified once.' % opt_str)

    if option.dest is 'runfile':
        parser.values.cmd = 'rdconfig'
    if option.dest is 'template':
        parser.values.cmd = 'wrconfig'

    setattr(parser.values, option.dest, value)
    if option.dest in path_options:
        setattr(parser.values, option.dest, os.path.abspath(value))


def parse_args():
    parser = OptionParser()
    parser.add_option('-c', action='callback', callback=options_cb,
                      type='string', dest='runfile', metavar='runfile',
                      help='Specify tests to run via config file.')
    parser.add_option('-d', action='store_true', default=False, dest='dryrun',
                      help='Dry run. Print tests, but take no other action.')
    parser.add_option('-g', action='store_true', default=False,
                      dest='do_groups', help='Make directories TestGroups.')
    parser.add_option('-o', action='callback', callback=options_cb,
                      default=BASEDIR, dest='outputdir', type='string',
                      metavar='outputdir', help='Specify an output directory.')
    parser.add_option('-p', action='callback', callback=options_cb,
                      default='', dest='pre', metavar='script',
                      type='string', help='Specify a pre script.')
    parser.add_option('-P', action='callback', callback=options_cb,
                      default='', dest='post', metavar='script',
                      type='string', help='Specify a post script.')
    parser.add_option('-q', action='store_true', default=False, dest='quiet',
                      help='Silence on the console during a test run.')
    parser.add_option('-t', action='callback', callback=options_cb, default=60,
                      dest='timeout', metavar='seconds', type='int',
                      help='Timeout (in seconds) for an individual test.')
    parser.add_option('-u', action='callback', callback=options_cb,
                      default='', dest='user', metavar='user', type='string',
                      help='Specify a different user name to run as.')
    parser.add_option('-w', action='callback', callback=options_cb,
                      default=None, dest='template', metavar='template',
                      type='string', help='Create a new config file.')
    parser.add_option('-x', action='callback', callback=options_cb, default='',
                      dest='pre_user', metavar='pre_user', type='string',
                      help='Specify a user to execute the pre script.')
```



```
    parser.add_option('-X', action='callback', callback=options_cb, default='',
                      dest='post_user', metavar='post_user', type='string',
                      help='Specify a user to execute the post script.')
    (options, pathnames) = parser.parse_args()

    if not options.runfile and not options.template:
        options.cmd = 'runtests'

    if options.runfile and len(pathnames):
        fail('Extraneous arguments.')

    options.pathnames = [os.path.abspath(path) for path in pathnames]

    return options


def main():
    options = parse_args()
    testrun = TestRun(options)

    if options.cmd is 'runtests':
        find_tests(testrun, options)
    elif options.cmd is 'rdconfig':
        testrun.read(testrun.logger, options)
    elif options.cmd is 'wrconfig':
        find_tests(testrun, options)
        testrun.write(options)
        exit(0)
    else:
        fail('Unknown command specified')

    testrun.complete_outputdirs()
    testrun.run(options)
    testrun.summary()
    exit(0)


if __name__ == '__main__':
    main()
```

```
**********************************************************
   29753 Thu Jun 14 17:15:58 2018
new/usr/src/uts/common/sys/proc.h
9600 LDT still not happy under KPTI
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1988, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2017 Joyent, Inc.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

#ifndef _SYS_PROC_H
#define _SYS_PROC_H

#include <sys/time.h>
#include <sys/thread.h>
#include <sys/cred.h>
#include <sys/user.h>
#include <sys/watchpoint.h>
#include <sys/timer.h>
#if defined(__x86)
#include <sys/tss.h>
#include <sys/segments.h>
#endif
#include <sys/utrap.h>
#include <sys/model.h>
#include <sys/refstr.h>
#include <sys/avl.h>
#include <sys/rctl.h>
#include <sys/list.h>
#include <sys/avl.h>
#include <sys/door_impl.h>
#include <sys/signalfd.h>
#include <sys/secflags.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Profile arguments.
 */
```



```
struct prof {
        void            *pr_base;       /* buffer base */
        uintptr_t       pr_off;         /* pc offset */
        size_t          pr_size;        /* buffer size */
        uint32_t        pr_scale;       /* pc scaling */
        long            pr_samples;     /* sample count */
};
_____unchanged_portion_omitted_

struct pool;
struct task;
struct zone;
struct brand;
struct corectl_path;
struct corectl_content;

/*
 * One structure allocated per active process.  It contains all
 * data needed about the process while the process may be swapped
 * out.  Other per-process data (user.h) is also inside the proc structure.
 * Lightweight-process data (lwp.h) and the kernel stack may be swapped out.
 */
typedef struct  proc {
        /*
         * Fields requiring no explicit locking
         */
        struct  vnode *p_exec;          /* pointer to a.out vnode */
        struct  as *p_as;               /* process address space pointer */
        struct  plock *p_lockp;         /* ptr to proc struct's mutex lock */
        kmutex_t p_crlock;              /* lock for p_cred */
        struct  cred    *p_cred;        /* process credentials */
        /*
         * Fields protected by pidlock
         */
        int     p_swapcnt;              /* number of swapped out lwps */
        char    p_stat;                 /* status of process */
        char    p_wcode;                /* current wait code */
        ushort_t p_pidflag;             /* flags protected only by pidlock */
        int     p_wdata;                /* current wait return value */
        pid_t   p_ppid;                 /* process id of parent */
        struct  proc    *p_link;        /* forward link */
        struct  proc    *p_parent;      /* ptr to parent process */
        struct  proc    *p_child;       /* ptr to first child process */
        struct  proc    *p_sibling;     /* ptr to next sibling proc on chain */
        struct  proc    *p_psibling;    /* ptr to prev sibling proc on chain */
        struct  proc    *p_sibling_ns;  /* prt to siblings with new state */
        struct  proc    *p_child_ns;    /* prt to children with new state */
        struct  proc    *p_next;        /* active chain link next */
        struct  proc    *p_prev;        /* active chain link prev */
        struct  proc    *p_nextofkin;   /* gets accounting info at exit */
        struct  proc    *p_orphan;
        struct  proc    *p_nextorph;
        struct  proc    *p_pglink;      /* process group hash chain link next */
        struct  proc    *p_ppglink;     /* process group hash chain link prev */
        struct  sess    *p_sessp;       /* session information */
        struct  pid     *p_pidp;        /* process ID info */
        struct  pid     *p_pgidp;       /* process group ID info */
        /*
         * Fields protected by p_lock
         */
        kcondvar_t p_cv;                /* proc struct's condition variable */
        kcondvar_t p_flag_cv;
        kcondvar_t p_lwpexit;           /* waiting for some lwp to exit */
        kcondvar_t p_holdlwps;          /* process is waiting for its lwps */
                                        /* to to be held.  */
        uint_t  p_proc_flag;            /* /proc-related flags */
```

```
184         uint_t  p_flag;                 /* protected while set. */
185                                         /* flags defined below */
186         clock_t p_utime;                /* user time, this process */
187         clock_t p_stime;                /* system time, this process */
188         clock_t p_cutime;               /* sum of children's user time */
189         clock_t p_cstime;               /* sum of children's system time */
190         avl_tree_t *p_segacct;          /* System V shared segment list */
191         avl_tree_t *p_semacct;          /* System V semaphore undo list */
192         caddr_t p_bssbase;              /* base addr of last bss below heap */
193         caddr_t p_brkbase;              /* base addr of heap */
194         size_t  p_brksize;              /* heap size in bytes */
195         uint_t  p_brkpageszc;           /* preferred heap max page size code */
196         /*
197          * Per process signal stuff.
198          */
199         k_sigset_t p_sig;               /* signals pending to this process */
200         k_sigset_t p_extsig;            /* signals sent from another contract */
201         k_sigset_t p_ignore;            /* ignore when generated */
202         k_sigset_t p_siginfo;           /* gets signal info with signal */
203         void *p_sigfd;                  /* signalfd support state */
204         struct sigqueue *p_sigqueue;    /* queued siginfo structures */
205         struct sigqhdr *p_sigqhdr;      /* hdr to sigqueue structure pool */
206         struct sigqhdr *p_signhdr;      /* hdr to signotify structure pool */
207         uchar_t p_stopsig;              /* jobcontrol stop signal */

209         /*
210          * Special per-process flag when set will fix misaligned memory
211          * references.
212          */
213         char    p_fixalignment;

215         /*
216          * Per process lwp and kernel thread stuff
217          */
218         id_t    p_lwpid;                /* most recently allocated lwpid */
219         int     p_lwpcnt;               /* number of lwps in this process */
220         int     p_lwprcnt;              /* number of not stopped lwps */
221         int     p_lwpdaemon;            /* number of TP_DAEMON lwps */
222         int     p_lwpwait;              /* number of lwps in lwp_wait() */
223         int     p_lwpdwait;             /* number of daemons in lwp_wait() */
224         int     p_zombcnt;              /* number of zombie lwps */
225         kthread_t *p_tlist;             /* circular list of threads */
226         lwpdir_t *p_lwpdir;             /* thread (lwp) directory */
227         lwpdir_t *p_lwpfree;            /* p_lwpdir free list */
228         tidhash_t *p_tidhash;           /* tid (lwpid) lookup hash table */
229         uint_t  p_lwpdir_sz;            /* number of p_lwpdir[] entries */
230         uint_t  p_tidhash_sz;           /* number of p_tidhash[] entries */
231         ret_tidhash_t *p_ret_tidhash;   /* retired tidhash hash tables */
232         uint64_t p_lgrpset;             /* unprotected hint of set of lgrps */
233                                         /* on which process has threads */
234         volatile lgrp_id_t  p_t1_lgrpid; /* main's thread lgroup id */
235         volatile lgrp_id_t  p_tr_lgrpid; /* text replica's lgroup id */
236 #if defined(_LP64)
237         uintptr_t  p_lgrpres2;          /* reserved for lgrp migration */
238 #endif
239         /*
240          * /proc (process filesystem) debugger interface stuff.
241          */
242         k_sigset_t p_sigmask;           /* mask of traced signals (/proc) */
243         k_fltset_t p_fltmask;           /* mask of traced faults (/proc) */
244         struct vnode *p_trace;          /* pointer to primary /proc vnode */
245         struct vnode *p_plist;          /* list of /proc vnodes for process */
246         kthread_t *p_agenttp;           /* thread ptr for /proc agent lwp */
247         avl_tree_t p_warea;             /* list of watched areas */
248         avl_tree_t p_wpage;             /* remembered watched pages (vfork) */
249         watched_page_t *p_wprot;        /* pages that need to have prot set */
```



```
250         int     p_mapcnt;               /* number of active pr_mappage()s */
251         kmutex_t p_maplock;             /* lock for pr_mappage() */
252         struct  proc  *p_rlink;         /* linked list for server */
253         kcondvar_t p_srwchan_cv;
254         size_t  p_stksize;              /* process stack size in bytes */
255         uint_t  p_stkpageszc;           /* preferred stack max page size code */

257         /*
258          * Microstate accounting, resource usage, and real-time profiling
259          */
260         hrtime_t p_mstart;              /* hi-res process start time */
261         hrtime_t p_mterm;               /* hi-res process termination time */
262         hrtime_t p_mlreal;              /* elapsed time sum over defunct lwps */
263         hrtime_t p_acct[NMSTATES];      /* microstate sum over defunct lwps */
264         hrtime_t p_cacct[NMSTATES];     /* microstate sum over child procs */
265         struct lrusage p_ru;            /* lrusage sum over defunct lwps */
266         struct lrusage p_cru;           /* lrusage sum over child procs */
267         struct itimerval p_rprof_timer; /* ITIMER_REALPROF interval timer */
268         uintptr_t p_rprof_cyclic;       /* ITIMER_REALPROF cyclic */
269         uint_t  p_defunct;              /* number of defunct lwps */
270         /*
271          * profiling. A lock is used in the event of multiple lwp's
272          * using the same profiling base/size.
273          */
274         kmutex_t p_pflock;              /* protects user profile arguments */
275         struct prof p_prof;             /* profile arguments */

277         /*
278          * Doors.
279          */
280         door_pool_t             p_server_threads; /* common thread pool */
281         struct door_node        *p_door_list;   /* active doors */
282         struct door_node        *p_unref_list;
283         kcondvar_t              p_unref_cv;
284         char                    p_unref_thread; /* unref thread created */

286         /*
287          * Kernel probes
288          */
289         uchar_t                 p_tnf_flags;

291         /*
292          * Solaris Audit
293          */
294         struct p_audit_data     *p_audit_data; /* per process audit structure */

296         pctxop_t        *p_pctx;

298 #if defined(__x86)
299         /*
300          * LDT support.
301          */
302         kmutex_t        p_ldtlock;      /* protects the following fields */
303         user_desc_t     *p_ldt;         /* Pointer to private LDT */
304         uint64_t        p_unused1;      /* no longer used */
305         uint64_t        p_unused2;      /* no longer used */
304         system_desc_t   p_ldt_desc;     /* segment descriptor for private LDT */
306         ushort_t        p_ldtlimit;     /* highest selector used */
307 #endif
308         size_t p_swrss;                 /* resident set size before last swap */
309         struct aio      *p_aio;         /* pointer to async I/O struct */
310         struct itimer   **p_itimer;     /* interval timers */
311         timeout_id_t    p_alarmid;      /* alarm's timeout id */
312         caddr_t         p_usrstack;     /* top of the process stack */
313         uint_t          p_stkprot;      /* stack memory protection */
314         uint_t          p_datprot;      /* data memory protection */
```

```
315         model_t         p_model;        /* data model determined at exec time */
316         struct lwpchan_data *p_lcp;     /* lwpchan cache */
317         kmutex_t        p_lcp_lock;     /* protects assignments to p_lcp */
318         utrap_handler_t *p_utraps;      /* pointer to user trap handlers */
319         struct corectl_path     *p_corefile;    /* pattern for core file */
320         struct task     *p_task;        /* our containing task */
321         struct proc     *p_taskprev;    /* ptr to previous process in task */
322         struct proc     *p_tasknext;    /* ptr to next process in task */
323         kmutex_t        p_sc_lock;      /* protects p_pagep */
324         struct sc_page_ctl *p_pagep;    /* list of process's shared pages */
325         struct rctl_set *p_rctls;       /* resource controls for this process */
326         rlim64_t        p_stk_ctl;      /* currently enforced stack size */
327         rlim64_t        p_fsz_ctl;      /* currently enforced file size */
328         rlim64_t        p_vmem_ctl;     /* currently enforced addr-space size */
329         rlim64_t        p_fno_ctl;      /* currently enforced file-desc limit */
330         pid_t           p_ancpid;       /* ancestor pid, used by exacct */
331         struct itimerval p_realitimer;  /* real interval timer */
332         timeout_id_t    p_itimerid;     /* real interval timer's timeout id */
333         struct corectl_content *p_content;      /* content of core file */

335         avl_tree_t      p_ct_held;      /* held contracts */
336         struct ct_equeue **p_ct_equeue; /* process-type event queues */

338         struct cont_process *p_ct_process; /* process contract */
339         list_node_t     p_ct_member;    /* process contract membership */
340         sigqueue_t      *p_killsqp;     /* sigqueue pointer for SIGKILL */

342         int             p_dtrace_probes; /* are there probes for this proc? */
343         uint64_t        p_dtrace_count; /* number of DTrace tracepoints */
344                                         /* (protected by P_PR_LOCK) */
345         void            *p_dtrace_helpers; /* DTrace helpers, if any */
346         struct pool     *p_pool;        /* pointer to containing pool */
347         kcondvar_t      p_poolcv;       /* synchronization with pools */
348         uint_t          p_poolcnt;      /* # threads inside pool barrier */
349         uint_t          p_poolflag;     /* pool-related flags (see below) */
350         uintptr_t       p_portcnt;      /* event ports counter */
351         struct zone     *p_zone;        /* zone in which process lives */
352         struct vnode    *p_execdir;     /* directory that p_exec came from */
353         struct brand    *p_brand;       /* process's brand  */
354         void            *p_brand_data;  /* per-process brand state */
355         psecflags_t     p_secflags;     /* per-process security flags */

357         /* additional lock to protect p_sessp (but not its contents) */
358         kmutex_t p_splock;
359         rctl_qty_t      p_locked_mem;   /* locked memory charged to proc */
360                                         /* protected by p_lock */
361         rctl_qty_t      p_crypto_mem;   /* /dev/crypto memory charged to proc */
362                                         /* protected by p_lock */
363         clock_t p_ttime;                /* buffered task time */

365         /*
366          * The user structure
367          */
368         struct user p_user;             /* (see sys/user.h) */
369 } proc_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
    8657 Thu Jun 14 17:15:58 2018
new/usr/src/uts/i86pc/ml/offsets.in
9600 LDT still not happy under KPTI
**********************************************************
```

```
\
\ Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
\ Copyright 2012 Garrett D'Amore <garrett@damore.org>.  All rights reserved.
\ Copyright 2018 Joyent, Inc.
\
\ CDDL HEADER START
\
\ The contents of this file are subject to the terms of the
\ Common Development and Distribution License (the "License").
\ You may not use this file except in compliance with the License.
\
\ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
\ or http://www.opensolaris.org/os/licensing.
\ See the License for the specific language governing permissions
\ and limitations under the License.
\
\ When distributing Covered Code, include this CDDL HEADER in each
\ file and include the License file at usr/src/OPENSOLARIS.LICENSE.
\ If applicable, add the following below this CDDL HEADER, with the
\ fields enclosed by brackets "[]" replaced with your own identifying
\ information: Portions Copyright [yyyy] [name of copyright owner]
\
\ CDDL HEADER END
\



\
\ offsets.in: input file to produce assym.h using the ctfstabs program
\

#ifndef _GENASSYM
#define _GENASSYM
#endif

#define SIZES   1

#include <sys/types.h>
#include <sys/bootsvcs.h>
#include <sys/systm.h>
#include <sys/sysinfo.h>
#include <sys/user.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/cpuvar.h>
#include <sys/tss.h>
#include <sys/privregs.h>
#include <sys/segments.h>
#include <sys/devops.h>
#include <sys/ddi_impldefs.h>
#include <vm/as.h>
#include <sys/avintr.h>
#include <sys/pic.h>
#include <sys/rm_platter.h>
#include <sys/stream.h>
#include <sys/strsubr.h>
#include <sys/sunddi.h>
#include <sys/modctl.h>
#include <sys/traptrace.h>
#include <sys/ontrap.h>
#include <sys/lgrp.h>
#include <sys/dtrace.h>
```



```
#include <sys/brand.h>
#include <sys/fastboot.h>
#include <sys/cpr_wakecode.h>
#include <sys/comm_page.h>

proc            PROCSIZE
        p_link
        p_next
        p_child
        p_sibling
        p_sig
        p_flag
        p_tlist
        p_as
        p_lockp
        p_user
        p_ldt
        p_ldt_desc
        p_model
        p_pctx
        p_agenttp
        p_zone
        p_brand
        p_brand_data

_kthread        THREAD_SIZE
        t_pcb                   T_LABEL
        t_lock
        t_lockstat
        t_lockp
        t_lock_flush
        t_kpri_req
        t_oldspl
        t_pri
        t_pil
        t_lwp
        t_procp
        t_link
        t_state
        t_mstate
        t_preempt_lk
        t_stk                   T_STACK
        t_swap
        t_lwpchan.lc_wchan      T_WCHAN
        t_flag                  T_FLAGS
        t_ctx
        t_lofault
        t_onfault
        t_ontrap
        t_cpu
        t_lpl
        t_bound_cpu
        t_intr
        t_forw
        t_back
        t_sig
        t_tid
        t_pre_sys
        t_preempt
        t_proc_flag
        t_startpc
        t_sysnum
        t_intr_start
        _tu._ts._t_astflag      T_ASTFLAG
        _tu._ts._t_post_sys     T_POST_SYS
        _tu._t_post_sys_ast     T_POST_SYS_AST
```

```
126         t_copyops
127 #ifdef  __amd64
128         t_useracc
129 #endif

131 ctxop
132         save_op                 CTXOP_SAVE

134 as
135         a_hat

137 user    USIZEBYTES
138         u_comm
139         u_signal

141 _label_t
142         val     LABEL_VAL

144 \#define        LABEL_PC        LABEL_VAL
145 \#define        LABEL_SP        _CONST(LABEL_VAL + LABEL_VAL_INCR)
146 \#define        T_PC            _CONST(T_LABEL + LABEL_PC)
147 \#define        T_SP            _CONST(T_LABEL + LABEL_SP)

149 _klwp
150         lwp_thread
151         lwp_procp
152         lwp_brand
153         lwp_eosys
154         lwp_regs
155         lwp_arg
156         lwp_ap
157         lwp_cursig
158         lwp_state
159         lwp_mstate.ms_acct      LWP_MS_ACCT
160         lwp_mstate.ms_prev      LWP_MS_PREV
161         lwp_mstate.ms_start     LWP_MS_START
162         lwp_mstate.ms_state_start LWP_MS_STATE_START
163         lwp_pcb
164         lwp_ru.sysc             LWP_RU_SYSC

166 \#define        LWP_ACCT_USER   _CONST(LWP_MS_ACCT + _MUL(LMS_USER, LWP_MS_ACCT_
167 \#define        LWP_ACCT_SYSTEM _CONST(LWP_MS_ACCT + _MUL(LMS_SYSTEM, LWP_MS_ACC

169 fpu_ctx
170         fpu_regs                FPU_CTX_FPU_REGS
171         fpu_flags               FPU_CTX_FPU_FLAGS
172         fpu_xsave_mask          FPU_CTX_FPU_XSAVE_MASK

174 fxsave_state    FXSAVE_STATE_SIZE
175         fx_fsw                  FXSAVE_STATE_FSW
176         fx_mxcsr_mask           FXSAVE_STATE_MXCSR_MASK


179 autovec         AUTOVECSIZE
180         av_vector
181         av_intarg1
182         av_intarg2
183         av_ticksp
184         av_link
185         av_prilevel
186         av_dip

188 av_head
189         avh_link
190         avh_hi_pri
191         avh_lo_pri
```



```
193 cpu
194         cpu_id
195         cpu_flags
196         cpu_self
197         cpu_thread
198         cpu_thread_lock
199         cpu_kprunrun
200         cpu_lwp
201         cpu_fpowner
202         cpu_idle_thread
203         cpu_intr_thread
204         cpu_intr_actv
205         cpu_base_spl
206         cpu_intr_stack
207         cpu_stats.sys.cpumigrate        CPU_STATS_SYS_CPUMIGRATE
208         cpu_stats.sys.intr              CPU_STATS_SYS_INTR
209         cpu_stats.sys.intrblk           CPU_STATS_SYS_INTRBLK
210         cpu_stats.sys.syscall           CPU_STATS_SYS_SYSCALL
211         cpu_profile_pc
212         cpu_profile_upc
213         cpu_profile_pil
214         cpu_ftrace.ftd_state            CPU_FTRACE_STATE
215         cpu_mstate
216         cpu_intracct

218 \#define        CPU_INTR_ACTV_REF       _CONST(CPU_INTR_ACTV + 2)

220 cpu
221         cpu_m.pil_high_start    CPU_PIL_HIGH_START
222         cpu_m.intrstat          CPU_INTRSTAT
223         cpu_m.mcpu_current_hat  CPU_CURRENT_HAT
224         cpu_m.mcpu_gdt          CPU_GDT
225         cpu_m.mcpu_idt          CPU_IDT
226         cpu_m.mcpu_tss          CPU_TSS
227         cpu_m.mcpu_softinfo     CPU_SOFTINFO
228         cpu_m.mcpu_pri          CPU_PRI
229 #if defined(__xpv)
230         cpu_m.mcpu_vcpu_info    CPU_VCPU_INFO
231 #endif

233 cpu
234         cpu_m.mcpu_kpti.kf_kernel_cr3   CPU_KPTI_KCR3
235         cpu_m.mcpu_kpti.kf_user_cr3     CPU_KPTI_UCR3
236         cpu_m.mcpu_kpti.kf_tr_rsp       CPU_KPTI_TR_RSP
237         cpu_m.mcpu_kpti.kf_tr_cr3       CPU_KPTI_TR_CR3
238         cpu_m.mcpu_kpti.kf_r13          CPU_KPTI_R13
239         cpu_m.mcpu_kpti.kf_r14          CPU_KPTI_R14
240         cpu_m.mcpu_kpti.kf_tr_ret_rsp   CPU_KPTI_RET_RSP

242         cpu_m.mcpu_kpti.kf_ss           CPU_KPTI_SS
243         cpu_m.mcpu_kpti.kf_rsp          CPU_KPTI_RSP
244         cpu_m.mcpu_kpti.kf_rflags       CPU_KPTI_RFLAGS
245         cpu_m.mcpu_kpti.kf_cs           CPU_KPTI_CS
246         cpu_m.mcpu_kpti.kf_rip          CPU_KPTI_RIP
247         cpu_m.mcpu_kpti.kf_err          CPU_KPTI_ERR

249         cpu_m.mcpu_pad2                 CPU_KPTI_START
250         cpu_m.mcpu_pad3                 CPU_KPTI_END

252         cpu_m.mcpu_kpti_dbg             CPU_KPTI_DBG

254 kpti_frame
255         kf_r14          KPTI_R14
256         kf_r13          KPTI_R13
257         kf_err          KPTI_ERR
```

```
        kf_rip          KPTI_RIP
        kf_cs           KPTI_CS
        kf_rflags       KPTI_RFLAGS
        kf_rsp          KPTI_RSP
        kf_ss           KPTI_SS

        kf_tr_rsp       KPTI_TOP

        kf_kernel_cr3   KPTI_KCR3
        kf_user_cr3     KPTI_UCR3
        kf_tr_ret_rsp   KPTI_RET_RSP
        kf_tr_cr3       KPTI_TR_CR3

        kf_tr_flag      KPTI_FLAG

standard_pic
        c_curmask
        c_iplmask

ddi_dma_impl
        dmai_rflags
        dmai_rdip

dev_info
        devi_ops                DEVI_DEV_OPS
        devi_bus_ctl
        devi_bus_dma_ctl
        devi_bus_dma_allochdl
        devi_bus_dma_freehdl
        devi_bus_dma_bindhdl
        devi_bus_dma_unbindhdl
        devi_bus_dma_flush
        devi_bus_dma_win

dev_ops
        devo_bus_ops            DEVI_BUS_OPS

bus_ops
        bus_ctl                 OPS_CTL
        bus_dma_map             OPS_MAP
        bus_dma_ctl             OPS_MCTL
        bus_dma_allochdl        OPS_ALLOCHDL
        bus_dma_freehdl         OPS_FREEHDL
        bus_dma_bindhdl         OPS_BINDHDL
        bus_dma_unbindhdl       OPS_UNBINDHDL
        bus_dma_flush           OPS_FLUSH
        bus_dma_win             OPS_WIN

sysent  SYSENT_SIZE     SYSENT_SIZE_SHIFT
        sy_callc
        sy_flags
        sy_narg

stdata
        sd_lock

queue
        q_flag
        q_next
        q_stream
        q_syncq
        q_qinfo

qinit
        qi_putp
```



```
syncq
        sq_flags
        sq_count
        sq_lock
        sq_wait

rm_platter
        rm_idt_lim              IDTROFF
        rm_gdt_lim              GDTROFF
        rm_pdbr                 CR3OFF
        rm_cpu                  CPUNOFF
        rm_cr4                  CR4OFF
        rm_cpu_halt_code        CPUHALTCODEOFF
        rm_cpu_halted           CPUHALTEDOFF

ddi_acc_impl
        ahi_acc_attr    ACC_ATTR
        ahi_get8        ACC_GETB
        ahi_get16       ACC_GETW
        ahi_get32       ACC_GETL
        ahi_get64       ACC_GETLL
        ahi_put8        ACC_PUTB
        ahi_put16       ACC_PUTW
        ahi_put32       ACC_PUTL
        ahi_put64       ACC_PUTLL
        ahi_rep_get8    ACC_REP_GETB
        ahi_rep_get16   ACC_REP_GETW
        ahi_rep_get32   ACC_REP_GETL
        ahi_rep_get64   ACC_REP_GETLL
        ahi_rep_put8    ACC_REP_PUTB
        ahi_rep_put16   ACC_REP_PUTW
        ahi_rep_put32   ACC_REP_PUTL
        ahi_rep_put64   ACC_REP_PUTLL

on_trap_data
        ot_prot
        ot_trap
        ot_trampoline
        ot_jmpbuf
        ot_prev
        ot_handle
        ot_pad1

trap_trace_ctl_t        __TRAPTR_SIZE TRAPTR_SIZE_SHIFT
        ttc_next        TRAPTR_NEXT
        ttc_first       TRAPTR_FIRST
        ttc_limit       TRAPTR_LIMIT

trap_trace_rec_t        TRAP_ENT_SIZE
        ttr_cr2
        ttr_info.idt_entry.vector       TTR_VECTOR
        ttr_info.idt_entry.ipl          TTR_IPL
        ttr_info.idt_entry.spl          TTR_SPL
        ttr_info.idt_entry.pri          TTR_PRI
        ttr_info.gate_entry.sysnum      TTR_SYSNUM
        ttr_marker
        ttr_stamp
        ttr_curthread
        ttr_sdepth
        ttr_stack

lgrp_ld
        lpl_lgrpid

dtrace_id_t     DTRACE_IDSIZE
```

```
390 cpu_core        CPU_CORE_SIZE   CPU_CORE_SHIFT
391         cpuc_dtrace_flags
392         cpuc_dtrace_illval

394 timespec        TIMESPEC_SIZE

396 gate_desc       GATE_DESC_SIZE

398 desctbr_t       DESCTBR_SIZE
399         dtr_limit
400         dtr_base

402 mod_stub_info   MODS_SIZE
403         mods_func_adr   MODS_INSTFCN
404         mods_errfcn     MODS_RETFCN
405         mods_flag

407 \#define        TRAP_TSIZE              _MUL(TRAP_ENT_SIZE, TRAPTR_NENT)

409 copyops
410         cp_copyin
411         cp_xcopyin
412         cp_copyout
413         cp_xcopyout
414         cp_copyinstr
415         cp_copyoutstr
416         cp_fuword8
417         cp_fuword16
418         cp_fuword32
419         cp_fuword64
420         cp_suword8
421         cp_suword16
422         cp_suword32
423         cp_suword64
424         cp_physio

426 brand
427         b_machops

429 brand_proc_data_t
430         spd_handler

432 fastboot_file_t
433         fb_va
434         fb_pte_list_va
435         fb_pte_list_pa
436         fb_dest_pa
437         fb_size
438         fb_next_pa
439         fb_sections
440         fb_sectcnt

442 fastboot_section_t
443         fb_sec_offset
444         fb_sec_paddr
445         fb_sec_size
446         fb_sec_bss_size

448 fastboot_info_t
449         fi_files
450         fi_has_pae
451         fi_pagetable_va
452         fi_pagetable_pa
453         fi_last_table_pa
454         fi_new_mbi_pa
455         fi_valid
```



```
457 zone
458         zone_brand_data

460 wc_cpu  WC_CPU_SIZE
461         wc_retaddr
462         wc_virtaddr
463         wc_cr0
464         wc_cr3
465         wc_cr4
466         wc_cr8
467         wc_fs
468         wc_fsbase
469         wc_gs
470         wc_gsbase
471         wc_kgsbase
472         wc_r8
473         wc_r9
474         wc_r10
475         wc_r11
476         wc_r12
477         wc_r13
478         wc_r14
479         wc_r15
480         wc_rax
481         wc_rbp
482         wc_rbx
483         wc_rcx
484         wc_rdi
485         wc_rdx
486         wc_rsi
487         wc_rsp
488         wc_gdt_limit    WC_GDT
489         wc_gdt_base
490         wc_idt_limit    WC_IDT
491         wc_idt_base
492         wc_tr
493         wc_ldt
494         wc_eflags
495         wc_ebx
496         wc_edi
497         wc_esi
498         wc_ebp
499         wc_esp
500         wc_esp
501         wc_ss
502         wc_cs
503         wc_ds
504         wc_es
505         wc_cpu_id
506         wc_saved_stack

508 wc_wakecode
509         wc_cpu

511 comm_page_s     COMM_PAGE_S_SIZE
```

```
**********************************************************
   14793 Thu Jun 14 17:15:59 2018
new/usr/src/uts/i86pc/os/mlsetup.c
9600 LDT still not happy under KPTI
**********************************************************
______unchanged_portion_omitted_


  98 /*
  99  * Setup routine called right before main(). Interposing this function
 100  * before main() allows us to call it in a machine-independent fashion.
 101  */
 102 void
 103 mlsetup(struct regs *rp)
 104 {
 105         u_longlong_t prop_value;
 106         extern struct classfuncs sys_classfuncs;
 107         extern disp_t cpu0_disp;
 108         extern char t0stack[];
 109         extern int post_fastreboot;
 110         extern uint64_t plat_dr_options;

 112         ASSERT_STACK_ALIGNED();

 114         /*
 115          * initialize cpu_self
 116          */
 117         cpu[0]->cpu_self = cpu[0];

 119 #if defined(__xpv)
 120         /*
 121          * Point at the hypervisor's virtual cpu structure
 122          */
 123         cpu[0]->cpu_m.mcpu_vcpu_info = &HYPERVISOR_shared_info->vcpu_info[0];
 124 #endif

 126         /*
 127          * check if we've got special bits to clear or set
 128          * when checking cpu features
 129          */

 131         if (bootprop_getval("cpuid_feature_ecx_include", &prop_value) != 0)
 132                 cpuid_feature_ecx_include = 0;
 133         else
 134                 cpuid_feature_ecx_include = (uint32_t)prop_value;

 136         if (bootprop_getval("cpuid_feature_ecx_exclude", &prop_value) != 0)
 137                 cpuid_feature_ecx_exclude = 0;
 138         else
 139                 cpuid_feature_ecx_exclude = (uint32_t)prop_value;

 141         if (bootprop_getval("cpuid_feature_edx_include", &prop_value) != 0)
 142                 cpuid_feature_edx_include = 0;
 143         else
 144                 cpuid_feature_edx_include = (uint32_t)prop_value;

 146         if (bootprop_getval("cpuid_feature_edx_exclude", &prop_value) != 0)
 147                 cpuid_feature_edx_exclude = 0;
 148         else
 149                 cpuid_feature_edx_exclude = (uint32_t)prop_value;

 151 #if !defined(__xpv)
 152         /*
 153          * Check to see if KPTI has been explicitly enabled or disabled.
 154          * We have to check this before init_desctbls().
 155          */
```



```
 156         if (bootprop_getval("kpti", &prop_value) == 0) {
 157                 kpti_enable = (uint64_t)(prop_value == 1);
 158                 prom_printf("unix: forcing kpti to %s due to boot argument\n",
 159                     (kpti_enable == 1) ? "ON" : "OFF");
 160         } else {
 161                 kpti_enable = 1;
 162         }

 164         if (bootprop_getval("pcid", &prop_value) == 0 && prop_value == 0) {
 165                 prom_printf("unix: forcing pcid to OFF due to boot argument\n");
 166                 x86_use_pcid = 0;
 167         } else if (kpti_enable != 1) {
 168                 x86_use_pcid = 0;
 169         }
 170 #endif

 172         /*
 173          * Initialize idt0, gdt0, ldt0_default, ktss0 and dftss.
 174          */
 175         init_desctbls();

 177         /*
 178          * lgrp_init() and possibly cpuid_pass1() need PCI config
 179          * space access
 180          */
 181 #if defined(__xpv)
 182         if (DOMAIN_IS_INITDOMAIN(xen_info))
 183                 pci_cfgspace_init();
 184 #else
 185         pci_cfgspace_init();
 186         /*
 187          * Initialize the platform type from CPU 0 to ensure that
 188          * determine_platform() is only ever called once.
 189          */
 190         determine_platform();
 191 #endif

 193         /*
 194          * The first lightweight pass (pass0) through the cpuid data
 195          * was done in locore before mlsetup was called.  Do the next
 196          * pass in C code.
 197          *
 198          * The x86_featureset is initialized here based on the capabilities
 199          * of the boot CPU.  Note that if we choose to support CPUs that have
 200          * different feature sets (at which point we would almost certainly
 201          * want to set the feature bits to correspond to the feature
 202          * minimum) this value may be altered.
 203          */
 204         cpuid_pass1(cpu[0], x86_featureset);

 206 #if !defined(__xpv)
 207         if ((get_hwenv() & HW_XEN_HVM) != 0)
 208                 xen_hvm_init();

 210         /*
 211          * Before we do anything with the TSCs, we need to work around
 212          * Intel erratum BT81.  On some CPUs, warm reset does not
 213          * clear the TSC.  If we are on such a CPU, we will clear TSC ourselves
 214          * here.  Other CPUs will clear it when we boot them later, and the
 215          * resulting skew will be handled by tsc_sync_master()/_slave();
 216          * note that such skew already exists and has to be handled anyway.
 217          *
 218          * We do this only on metal.  This same problem can occur with a
 219          * hypervisor that does not happen to virtualise a TSC that starts from
 220          * zero, regardless of CPU type; however, we do not expect hypervisors
 221          * that do not virtualise TSC that way to handle writes to TSC
```

```
222          * correctly, either.
223          */
224         if (get_hwenv() == HW_NATIVE &&
225             cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
226             cpuid_getfamily(CPU) == 6 &&
227             (cpuid_getmodel(CPU) == 0x2d || cpuid_getmodel(CPU) == 0x3e) &&
228             is_x86_feature(x86_featureset, X86FSET_TSC)) {
229                 (void) wrmsr(REG_TSC, 0UL);
230         }

232         /*
233          * Patch the tsc_read routine with appropriate set of instructions,
234          * depending on the processor family and architecure, to read the
235          * time-stamp counter while ensuring no out-of-order execution.
236          * Patch it while the kernel text is still writable.
237          *
238          * Note: tsc_read is not patched for intel processors whose family
239          * is >6 and for amd whose family >f (in case they don't support rdtscp
240          * instruction, unlikely). By default tsc_read will use cpuid for
241          * serialization in such cases. The following code needs to be
242          * revisited if intel processors of family >= f retains the
243          * instruction serialization nature of mfence instruction.
244          * Note: tsc_read is not patched for x86 processors which do
245          * not support "mfence". By default tsc_read will use cpuid for
246          * serialization in such cases.
247          *
248          * The Xen hypervisor does not correctly report whether rdtscp is
249          * supported or not, so we must assume that it is not.
250          */
251         if ((get_hwenv() & HW_XEN_HVM) == 0 &&
252             is_x86_feature(x86_featureset, X86FSET_TSCP))
253                 patch_tsc_read(TSC_TSCP);
254         else if (cpuid_getvendor(CPU) == X86_VENDOR_AMD &&
255             cpuid_getfamily(CPU) <= 0xf &&
256             is_x86_feature(x86_featureset, X86FSET_SSE2))
257                 patch_tsc_read(TSC_RDTSC_MFENCE);
258         else if (cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
259             cpuid_getfamily(CPU) <= 6 &&
260             is_x86_feature(x86_featureset, X86FSET_SSE2))
261                 patch_tsc_read(TSC_RDTSC_LFENCE);

263 #endif  /* !__xpv */

265 #if defined(__i386) && !defined(__xpv)
266         /*
267          * Some i386 processors do not implement the rdtsc instruction,
268          * or at least they do not implement it correctly. Patch them to
269          * return 0.
270          */
271         if (!is_x86_feature(x86_featureset, X86FSET_TSC))
272                 patch_tsc_read(TSC_NONE);
273 #endif  /* __i386 && !__xpv */

275 #if defined(__amd64) && !defined(__xpv)
276         patch_memops(cpuid_getvendor(CPU));
277 #endif  /* __amd64 && !__xpv */

279 #if !defined(__xpv)
280         /* XXPV what, if anything, should be dorked with here under xen? */

282         /*
283          * While we're thinking about the TSC, let's set up %cr4 so that
284          * userland can issue rdtsc, and initialize the TSC_AUX value
285          * (the cpuid) for the rdtscp instruction on appropriately
286          * capable hardware.
287          */
```



```
288         if (is_x86_feature(x86_featureset, X86FSET_TSC))
289                 setcr4(getcr4() & ~CR4_TSD);

291         if (is_x86_feature(x86_featureset, X86FSET_TSCP))
292                 (void) wrmsr(MSR_AMD_TSCAUX, 0);

294         /*
295          * Let's get the other %cr4 stuff while we're here. Note, we defer
296          * enabling CR4_SMAP until startup_end(); however, that's importantly
297          * before we start other CPUs. That ensures that it will be synced out
298          * to other CPUs.
299          */
300         if (is_x86_feature(x86_featureset, X86FSET_DE))
301                 setcr4(getcr4() | CR4_DE);

303         if (is_x86_feature(x86_featureset, X86FSET_SMEP))
304                 setcr4(getcr4() | CR4_SMEP);
305 #endif /* __xpv */

307         /*
308          * initialize t0
309          */
310         t0.t_stk = (caddr_t)rp - MINFRAME;
311         t0.t_stkbase = t0stack;
312         t0.t_pri = maxclsyspri - 3;
313         t0.t_schedflag = TS_LOAD | TS_DONT_SWAP;
314         t0.t_procp = &p0;
315         t0.t_plockp = &p0lock.pl_lock;
316         t0.t_lwp = &lwp0;
317         t0.t_forw = &t0;
318         t0.t_back = &t0;
319         t0.t_next = &t0;
320         t0.t_prev = &t0;
321         t0.t_cpu = cpu[0];
322         t0.t_disp_queue = &cpu0_disp;
323         t0.t_bind_cpu = PBIND_NONE;
324         t0.t_bind_pset = PS_NONE;
325         t0.t_bindflag = (uchar_t)default_binding_mode;
326         t0.t_cpupart = &cp_default;
327         t0.t_clfuncs = &sys_classfuncs.thread;
328         t0.t_copyops = NULL;
329         THREAD_ONPROC(&t0, CPU);

331         lwp0.lwp_thread = &t0;
332         lwp0.lwp_regs = (void *)rp;
333         lwp0.lwp_procp = &p0;
334         t0.t_tid = p0.p_lwpcnt = p0.p_lwprcnt = p0.p_lwpid = 1;

336         p0.p_exec = NULL;
337         p0.p_stat = SRUN;
338         p0.p_flag = SSYS;
339         p0.p_tlist = &t0;
340         p0.p_stksize = 2*PAGESIZE;
341         p0.p_stkpageszc = 0;
342         p0.p_as = &kas;
343         p0.p_lockp = &p0lock;
344         p0.p_brkpageszc = 0;
345         p0.p_t1_lgrpid = LGRP_NONE;
346         p0.p_tr_lgrpid = LGRP_NONE;
347         psecflags_default(&p0.p_secflags);

349         sigorset(&p0.p_ignore, &ignoredefault);

351         CPU->cpu_thread = &t0;
352         bzero(&cpu0_disp, sizeof (disp_t));
353         CPU->cpu_disp = &cpu0_disp;
```

```
354         CPU->cpu_disp->disp_cpu = CPU;
355         CPU->cpu_dispthread = &t0;
356         CPU->cpu_idle_thread = &t0;
357         CPU->cpu_flags = CPU_READY | CPU_RUNNING | CPU_EXISTS | CPU_ENABLE;
358         CPU->cpu_dispatch_pri = t0.t_pri;

360         CPU->cpu_id = 0;

362         CPU->cpu_pri = 12;              /* initial PIL for the boot CPU */

364         /*
365          * The kernel doesn't use LDTs unless a process explicitly requests one.
366          */
367         p0.p_ldt_desc = null_sdesc;

369         /*
365          * Initialize thread/cpu microstate accounting
366          */
367         init_mstate(&t0, LMS_SYSTEM);
368         init_cpu_mstate(CPU, CMS_SYSTEM);

370         /*
371          * Initialize lists of available and active CPUs.
372          */
373         cpu_list_init(CPU);

375         pg_cpu_bootstrap(CPU);

377         /*
378          * Now that we have taken over the GDT, IDT and have initialized
379          * active CPU list it's time to inform kmdb if present.
380          */
381         if (boothowto & RB_DEBUG)
382                 kdi_idt_sync();

384         if (BOP_GETPROPLEN(bootops, "efi-systab") < 0) {
385                 /*
386                  * In BIOS system, explicitly set console to text mode (0x3)
387                  * if this is a boot post Fast Reboot, and the console is set
388                  * to CONS_SCREEN_TEXT.
389                  */
390                 if (post_fastreboot &&
391                     boot_console_type(NULL) == CONS_SCREEN_TEXT) {
392                         set_console_mode(0x3);
393                 }
394         }

396         /*
397          * If requested (boot -d) drop into kmdb.
398          *
399          * This must be done after cpu_list_init() on the 64-bit kernel
400          * since taking a trap requires that we re-compute gsbase based
401          * on the cpu list.
402          */
403         if (boothowto & RB_DEBUGENTER)
404                 kmdb_enter();

406         cpu_vm_data_init(CPU);

408         rp->r_fp = 0;   /* terminate kernel stack traces! */

410         prom_init("kernel", (void *)NULL);

412         /* User-set option overrides firmware value. */
413         if (bootprop_getval(PLAT_DR_OPTIONS_NAME, &prop_value) == 0) {
414                 plat_dr_options = (uint64_t)prop_value;
```



```
415         }
416 #if defined(__xpv)
417         /* No support of DR operations on xpv */
418         plat_dr_options = 0;
419 #else   /* __xpv */
420         /* Flag PLAT_DR_FEATURE_ENABLED should only be set by DR driver. */
421         plat_dr_options &= ~PLAT_DR_FEATURE_ENABLED;
422 #ifndef __amd64
423         /* Only enable CPU/memory DR on 64 bits kernel. */
424         plat_dr_options &= ~PLAT_DR_FEATURE_MEMORY;
425         plat_dr_options &= ~PLAT_DR_FEATURE_CPU;
426 #endif  /* __amd64 */
427 #endif  /* __xpv */

429         /*
430          * Get value of "plat_dr_physmax" boot option.
431          * It overrides values calculated from MSCT or SRAT table.
432          */
433         if (bootprop_getval(PLAT_DR_PHYSMAX_NAME, &prop_value) == 0) {
434                 plat_dr_physmax = ((uint64_t)prop_value) >> PAGESHIFT;
435         }

437         /* Get value of boot_ncpus. */
438         if (bootprop_getval(BOOT_NCPUS_NAME, &prop_value) != 0) {
439                 boot_ncpus = NCPU;
440         } else {
441                 boot_ncpus = (int)prop_value;
442                 if (boot_ncpus <= 0 || boot_ncpus > NCPU)
443                         boot_ncpus = NCPU;
444         }

446         /*
447          * Set max_ncpus and boot_max_ncpus to boot_ncpus if platform doesn't
448          * support CPU DR operations.
449          */
450         if (plat_dr_support_cpu() == 0) {
451                 max_ncpus = boot_max_ncpus = boot_ncpus;
452         } else {
453                 if (bootprop_getval(PLAT_MAX_NCPUS_NAME, &prop_value) != 0) {
454                         max_ncpus = NCPU;
455                 } else {
456                         max_ncpus = (int)prop_value;
457                         if (max_ncpus <= 0 || max_ncpus > NCPU) {
458                                 max_ncpus = NCPU;
459                         }
460                         if (boot_ncpus > max_ncpus) {
461                                 boot_ncpus = max_ncpus;
462                         }
463                 }

465                 if (bootprop_getval(BOOT_MAX_NCPUS_NAME, &prop_value) != 0) {
466                         boot_max_ncpus = boot_ncpus;
467                 } else {
468                         boot_max_ncpus = (int)prop_value;
469                         if (boot_max_ncpus <= 0 || boot_max_ncpus > NCPU) {
470                                 boot_max_ncpus = boot_ncpus;
471                         } else if (boot_max_ncpus > max_ncpus) {
472                                 boot_max_ncpus = max_ncpus;
473                         }
474                 }
475         }

477         /*
478          * Initialize the lgrp framework
479          */
480         lgrp_init(LGRP_INIT_STAGE1);
```

```
482         if (boothowto & RB_HALT) {
483                 prom_printf("unix: kernel halted by -h flag\n");
484                 prom_enter_mon();
485         }

487         ASSERT_STACK_ALIGNED();

489         /*
490          * Fill out cpu_ucode_info.  Update microcode if necessary.
491          */
492         ucode_check(CPU);

494         if (workaround_errata(CPU) != 0)
495                 panic("critical workaround(s) missing for boot cpu");
496 }
```

unchanged_portion_omitted_

```
**********************************************************
   52277 Thu Jun 14 17:15:59 2018
new/usr/src/uts/i86pc/os/mp_startup.c
9600 LDT still not happy under KPTI
**********************************************************
______unchanged_portion_omitted_

2047 /*
2048  * The following two routines are used as context operators on threads belonging
2049  * to processes with a private LDT (see sysi86).  Due to the rarity of such
2050  * processes, these routines are currently written for best code readability and
2051  * organization rather than speed.  We could avoid checking x86_featureset at
2052  * every context switch by installing different context ops, depending on
2053  * x86_featureset, at LDT creation time -- one for each combination of fast
2054  * syscall features.
2055  */

2057 /*ARGSUSED*/
2057 void
2058 cpu_fast_syscall_disable(void)
2059 cpu_fast_syscall_disable(void *arg)
2059 {
2060         if (is_x86_feature(x86_featureset, X86FSET_MSR) &&
2061             is_x86_feature(x86_featureset, X86FSET_SEP))
2062                 cpu_sep_disable();
2063         if (is_x86_feature(x86_featureset, X86FSET_MSR) &&
2064             is_x86_feature(x86_featureset, X86FSET_ASYSC))
2065                 cpu_asysc_disable();
2066 }

2069 /*ARGSUSED*/
2068 void
2069 cpu_fast_syscall_enable(void)
2071 cpu_fast_syscall_enable(void *arg)
2070 {
2071         if (is_x86_feature(x86_featureset, X86FSET_MSR) &&
2072             is_x86_feature(x86_featureset, X86FSET_SEP))
2073                 cpu_sep_enable();
2074         if (is_x86_feature(x86_featureset, X86FSET_MSR) &&
2075             is_x86_feature(x86_featureset, X86FSET_ASYSC))
2076                 cpu_asysc_enable();
2077 }
______unchanged_portion_omitted_
```

```
**********************************************************
   38161 Thu Jun 14 17:15:59 2018
new/usr/src/uts/intel/ia32/os/desctbls.c
9600 LDT still not happy under KPTI
**********************************************************
______unchanged_portion_omitted_

 166 /*
 167  * The brand infrastructure interposes on two handlers, and we use one as a
 168  * NULL signpost.
 169  */
 170 static struct interposing_handler brand_tbl[2];

 172 /*
 173  * software prototypes for default local descriptor table
 174  */

 176 /*
 177  * Routines for loading segment descriptors in format the hardware
 178  * can understand.
 179  */

 181 #if defined(__amd64)

 181 /*
 182  * In long mode we have the new L or long mode attribute bit
 183  * for code segments. Only the conforming bit in type is used along
 184  * with descriptor priority and present bits. Default operand size must
 185  * be zero when in long mode. In 32-bit compatibility mode all fields
 186  * are treated as in legacy mode. For data segments while in long mode
 187  * only the present bit is loaded.
 188  */
 189 void
 190 set_usegd(user_desc_t *dp, uint_t lmode, void *base, size_t size,
 191     uint_t type, uint_t dpl, uint_t gran, uint_t defopsz)
 192 {
 193         ASSERT(lmode == SDP_SHORT || lmode == SDP_LONG);
 194         /* This should never be a "system" segment. */
 195         ASSERT3U(type & SDT_S, !=, 0);

 197         /*
 198          * 64-bit long mode.
 199          */
 200         if (lmode == SDP_LONG)
 201                 dp->usd_def32 = 0;              /* 32-bit operands only */
 202         else
 203                 /*
 204                  * 32-bit compatibility mode.
 205                  */
 206                 dp->usd_def32 = defopsz;        /* 0 = 16, 1 = 32-bit ops */

 208         /*
 209          * We should always set the "accessed" bit (SDT_A), otherwise the CPU
 210          * will write to the GDT whenever we change segment registers around.
 211          * With KPTI on, the GDT is read-only in the user page table, which
 212          * causes crashes if we don't set this.
 213          */
 214         ASSERT3U(type & SDT_A, !=, 0);

 216         dp->usd_long = lmode;   /* 64-bit mode */
 217         dp->usd_type = type;
 218         dp->usd_dpl = dpl;
 219         dp->usd_p = 1;
 220         dp->usd_gran = gran;            /* 0 = bytes, 1 = pages */

 222         dp->usd_lobase = (uintptr_t)base;
```



```
 223         dp->usd_midbase = (uintptr_t)base >> 16;
 224         dp->usd_hibase = (uintptr_t)base >> (16 + 8);
 225         dp->usd_lolimit = size;
 226         dp->usd_hilimit = (uintptr_t)size >> 16;
 227 }

 221 #elif defined(__i386)

 229 /*
 224  * Install user segment descriptor for code and data.
 225  */
 226 void
 227 set_usegd(user_desc_t *dp, void *base, size_t size, uint_t type,
 228     uint_t dpl, uint_t gran, uint_t defopsz)
 229 {
 230         dp->usd_lolimit = size;
 231         dp->usd_hilimit = (uintptr_t)size >> 16;

 233         dp->usd_lobase = (uintptr_t)base;
 234         dp->usd_midbase = (uintptr_t)base >> 16;
 235         dp->usd_hibase = (uintptr_t)base >> (16 + 8);

 237         dp->usd_type = type;
 238         dp->usd_dpl = dpl;
 239         dp->usd_p = 1;
 240         dp->usd_def32 = defopsz;        /* 0 = 16, 1 = 32 bit operands */
 241         dp->usd_gran = gran;            /* 0 = bytes, 1 = pages */
 242 }

 244 #endif  /* __i386 */

 246 /*
 230  * Install system segment descriptor for LDT and TSS segments.
 231  */

 250 #if defined(__amd64)

 233 void
 234 set_syssegd(system_desc_t *dp, void *base, size_t size, uint_t type,
 235     uint_t dpl)
 236 {
 237         dp->ssd_lolimit = size;
 238         dp->ssd_hilimit = (uintptr_t)size >> 16;

 240         dp->ssd_lobase = (uintptr_t)base;
 241         dp->ssd_midbase = (uintptr_t)base >> 16;
 242         dp->ssd_hibase = (uintptr_t)base >> (16 + 8);
 243         dp->ssd_hi64base = (uintptr_t)base >> (16 + 8 + 8);

 245         dp->ssd_type = type;
 246         dp->ssd_zero1 = 0;      /* must be zero */
 247         dp->ssd_zero2 = 0;
 248         dp->ssd_dpl = dpl;
 249         dp->ssd_p = 1;
 250         dp->ssd_gran = 0;       /* force byte units */
 251 }
______unchanged_portion_omitted_

 284 #elif defined(__i386)

 286 void
 287 set_syssegd(system_desc_t *dp, void *base, size_t size, uint_t type,
 288     uint_t dpl)
 289 {
 290         dp->ssd_lolimit = size;
 291         dp->ssd_hilimit = (uintptr_t)size >> 16;
```

```
293         dp->ssd_lobase = (uintptr_t)base;
294         dp->ssd_midbase = (uintptr_t)base >> 16;
295         dp->ssd_hibase = (uintptr_t)base >> (16 + 8);

297         dp->ssd_type = type;
298         dp->ssd_zero = 0;       /* must be zero */
299         dp->ssd_dpl = dpl;
300         dp->ssd_p = 1;
301         dp->ssd_gran = 0;       /* force byte units */
302 }

304 void *
305 get_ssd_base(system_desc_t *dp)
306 {
307         uintptr_t       base;

309         base = (uintptr_t)dp->ssd_lobase |
310             (uintptr_t)dp->ssd_midbase << 16 |
311             (uintptr_t)dp->ssd_hibase << (16 + 8);
312         return ((void *)base);
313 }

315 #endif  /* __i386 */

265 /*
266  * Install gate segment descriptor for interrupt, trap, call and task gates.
267  *
268  * For 64 bit native if we have KPTI enabled, we use the IST stack mechanism on
269  * all interrupts.  We have different ISTs for each class of exceptions that are
270  * most likely to occur while handling an existing exception; while many of
271  * these are just going to panic, it’s nice not to trample on the existing
272  * exception state for debugging purposes.
273  *
274  * Normal interrupts are all redirected unconditionally to the KPTI trampoline
275  * stack space. This unifies the trampoline handling between user and kernel
276  * space (and avoids the need to touch %gs).
277  *
278  * The KDI IDT *all* uses the DBG IST: consider single stepping tr_pftrap, when
279  * we do a read from KMDB that cause another #PF.  Without its own IST, this
280  * would stomp on the kernel’s mcpu_kpti_flt frame.
281  */
282 uint_t
283 idt_vector_to_ist(uint_t vector)
284 {
285 #if defined(__xpv)
286         _NOTE(ARGUNUSED(vector));
287         return (IST_NONE);
288 #else
289         switch (vector) {
290         /* These should always use IST even without KPTI enabled. */
291         case T_DBLFLT:
292                 return (IST_DF);
293         case T_NMIFLT:
294                 return (IST_NMI);
295         case T_MCE:
296                 return (IST_MCE);

298         case T_BPTFLT:
299         case T_SGLSTP:
300                 if (kpti_enable == 1) {
301                         return (IST_DBG);
302                 }
303                 return (IST_NONE);
304         case T_STKFLT:
305         case T_GPFLT:
```



```
306         case T_PGFLT:
307                 if (kpti_enable == 1) {
308                         return (IST_NESTABLE);
309                 }
310                 return (IST_NONE);
311         default:
312                 if (kpti_enable == 1) {
313                         return (IST_DEFAULT);
314                 }
315                 return (IST_NONE);
316         }
317 #endif
318 }
```

_____***unchanged_portion_omitted_***

```
334 /*
335  * Updates a single user descriptor in the the GDT of the current cpu.
336  * Caller is responsible for preventing cpu migration.
337  */

339 void
340 gdt_update_usegd(uint_t sidx, user_desc_t *udp)
341 {
342 #if defined(DEBUG)
343         /* This should never be a "system" segment, but it might be null. */
344         if (udp->usd_p != 0 || udp->usd_type != 0) {
345                 ASSERT3U(udp->usd_type & SDT_S, !=, 0);
346         }
347         /*
348          * We should always set the "accessed" bit (SDT_A), otherwise the CPU
349          * will write to the GDT whenever we change segment registers around.
350          * With KPTI on, the GDT is read-only in the user page table, which
351          * causes crashes if we don’t set this.
352          */
353         if (udp->usd_p != 0 || udp->usd_type != 0) {
354                 ASSERT3U(udp->usd_type & SDT_A, !=, 0);
355         }
356 #endif

358 #if defined(__xpv)

359         uint64_t dpa = CPU->cpu_m.mcpu_gdtpa + sizeof (*udp) * sidx;

361         if (HYPERVISOR_update_descriptor(pa_to_ma(dpa), *(uint64_t *)udp))
362                 panic("gdt_update_usegd: HYPERVISOR_update_descriptor");

364 #else   /* __xpv */

365         CPU->cpu_gdt[sidx] = *udp;

366 #endif  /* __xpv */
367 }

369 /*
370  * Writes single descriptor pointed to by udp into a processes
371  * LDT entry pointed to by ldp.
372  */
373 int
374 ldt_update_segd(user_desc_t *ldp, user_desc_t *udp)
375 {
376 #if defined(DEBUG)
377         /* This should never be a "system" segment, but it might be null. */
378         if (udp->usd_p != 0 || udp->usd_type != 0) {
379                 ASSERT3U(udp->usd_type & SDT_S, !=, 0);
380         }
381         /*
```

```
382          * We should always set the "accessed" bit (SDT_A), otherwise the CPU
383          * will write to the LDT whenever we change segment registers around.
384          * With KPTI on, the LDT is read-only in the user page table, which
385          * causes crashes if we don't set this.
386          */
387         if (udp->usd_p != 0 || udp->usd_type != 0) {
388                 ASSERT3U(udp->usd_type & SDT_A, !=, 0);
389         }
390 #endif

392 #if defined(__xpv)

393         uint64_t dpa;

395         dpa = mmu_ptob(hat_getpfnum(kas.a_hat, (caddr_t)ldp)) |
396             ((uintptr_t)ldp & PAGEOFFSET);

398         /*
399          * The hypervisor is a little more restrictive about what it
400          * supports in the LDT.
401          */
402         if (HYPERVISOR_update_descriptor(pa_to_ma(dpa), *(uint64_t *)udp) != 0)
403                 return (EINVAL);

405 #else   /* __xpv */

406         *ldp = *udp;

408 #endif  /* __xpv */
409         return (0);
410 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   21632 Thu Jun 14 17:15:59 2018
new/usr/src/uts/intel/ia32/os/sysi86.c
9600 LDT still not happy under KPTI
**********************************************************
______unchanged_portion_omitted_

 277 static void
 278 ssd_to_usd(struct ssd *ssd, user_desc_t *usd)
 279 {

 281         ASSERT(bcmp(usd, &null_udesc, sizeof (*usd)) == 0);

 283         USEGD_SETBASE(usd, ssd->bo);
 284         USEGD_SETLIMIT(usd, ssd->ls);

 286         /*
 287          * Set type, dpl and present bits.
 288          *
 289          * Force the "accessed" bit to on so that we don't run afoul of
 290          * KPTI.
 287          * set type, dpl and present bits.
 291          */
 292         usd->usd_type = ssd->acc1 | SDT_A;
 289         usd->usd_type = ssd->acc1;
 293         usd->usd_dpl = ssd->acc1 >> 5;
 294         usd->usd_p = ssd->acc1 >> (5 + 2);

 296         ASSERT(usd->usd_type >= SDT_MEMRO);
 297         ASSERT(usd->usd_dpl == SEL_UPL);

 299         /*
 300          * 64-bit code selectors are never allowed in the LDT.
 301          * Reserved bit is always 0 on 32-bit systems.
 302          */
 303 #if defined(__amd64)
 304         usd->usd_long = 0;
 305 #else
 306         usd->usd_reserved = 0;
 307 #endif

 309         /*
 310          * set avl, DB and granularity bits.
 311          */
 312         usd->usd_avl = ssd->acc2;
 313         usd->usd_def32 = ssd->acc2 >> (1 + 1);
 314         usd->usd_gran = ssd->acc2 >> (1 + 1 + 1);
 315 }
______unchanged_portion_omitted_

 342 #endif  /* __i386 */

 344 /*
 345  * Load LDT register with the current process's LDT.
 346  */
 347 static void
 348 ldt_load(void)
 349 {
 350 #if defined(__xpv)
 351         xen_set_ldt(curproc->p_ldt, curproc->p_ldtlimit + 1);
 348         xen_set_ldt(get_ssd_base(&curproc->p_ldt_desc),
 349             curproc->p_ldtlimit + 1);
 352 #else
 353         size_t len;
 354         system_desc_t desc;
```



```
 356         /*
 357          * Before we can use the LDT on this CPU, we must install the LDT in the
 358          * user mapping table.
 359          */
 360         len = (curproc->p_ldtlimit + 1) * sizeof (user_desc_t);
 361         bcopy(curproc->p_ldt, CPU->cpu_m.mcpu_ldt, len);
 362         CPU->cpu_m.mcpu_ldt_len = len;
 363         set_syssegd(&desc, CPU->cpu_m.mcpu_ldt, len - 1, SDT_SYSLDT, SEL_KPL);
 364         *((system_desc_t *)&CPU->cpu_gdt[GDT_LDT]) = desc;

 366         wr_ldtr(ULDT_SEL);
 367 #endif
 368 }
______unchanged_portion_omitted_

 388 /*ARGSUSED*/
 389 static void
 390 ldt_savectx(proc_t *p)
 391 {
 392         ASSERT(p->p_ldt != NULL);
 393         ASSERT(p == curproc);

 395 #if defined(__amd64)
 396         /*
 397          * The 64-bit kernel must be sure to clear any stale ldt
 398          * selectors when context switching away from a process that
 399          * has a private ldt. Consider the following example:
 400          *
 401          *      Wine creats a ldt descriptor and points a segment register
 402          *      to it.
 403          *
 404          *      We then context switch away from wine lwp to kernel
 405          *      thread and hit breakpoint in kernel with kmdb
 406          *
 407          *      When we continue and resume from kmdb we will #gp
 408          *      fault since kmdb will have saved the stale ldt selector
 409          *      from wine and will try to restore it but we are no longer in
 410          *      the context of the wine process and do not have our
 411          *      ldtr register pointing to the private ldt.
 412          */
 413         reset_sregs();
 414 #endif

 416         ldt_unload();
 417         cpu_fast_syscall_enable();
 415         cpu_fast_syscall_enable(NULL);
 418 }

 420 static void
 421 ldt_restorectx(proc_t *p)
 422 {
 423         ASSERT(p->p_ldt != NULL);
 424         ASSERT(p == curproc);

 426         ldt_load();
 427         cpu_fast_syscall_disable();
 425         cpu_fast_syscall_disable(NULL);
 428 }

 430 /*
 431  * At exec time, we need to clear up our LDT context and re-enable fast syscalls
 432  * for the new process image.
 433  *
 434  * The same is true for the other case, where we have:
 435  *
 436  * proc_exit()
```

```
 *  ->exitpctx()->ldt_savectx()
 *  ->freepctx()->ldt_freectx()
 *
 * Because pre-emption is not prevented between the two callbacks, we could have
 * come off CPU, and brought back LDT context when coming back on CPU via
 * ldt_restorectx().
 * When a process with a private LDT execs, fast syscalls must be enabled for
 * the new process image.
 */
/* ARGSUSED */
static void
ldt_freectx(proc_t *p, int isexec)
{
        ASSERT(p->p_ldt != NULL);
        ASSERT(p == curproc);
        ASSERT(p->p_ldt);

        if (isexec) {
        kpreempt_disable();
        ldt_free(p);
        cpu_fast_syscall_enable();
                cpu_fast_syscall_enable(NULL);
        kpreempt_enable();
        }

        /*
         * ldt_free() will free the memory used by the private LDT, reset the
         * process's descriptor, and re-program the LDTR.
         */
        ldt_free(p);
}
```

unchanged_portion_omitted_

```
int
setdscr(struct ssd *ssd)
{
        ushort_t seli;          /* selector index */
        user_desc_t *ldp;       /* descriptor pointer */
        user_desc_t ndesc;      /* new descriptor */
        proc_t  *pp = curproc;
        proc_t  *pp = ttoproc(curthread);
        int     rc = 0;

        /*
         * LDT segments: executable and data at DPL 3 only.
         */
        if (!SELISLDT(ssd->sel) || !SELISUPL(ssd->sel))
                return (EINVAL);

        /*
         * check the selector index.
         */
        seli = SELTOIDX(ssd->sel);
        if (seli >= MAXNLDT || seli < LDT_UDBASE)
                return (EINVAL);

        ndesc = null_udesc;
        mutex_enter(&pp->p_ldtlock);

        /*
         * If this is the first time for this process then setup a
         * private LDT for it.
         */
        if (pp->p_ldt == NULL) {
                ldt_alloc(pp, seli);
```



```
                /*
                 * Now that this process has a private LDT, the use of
                 * the syscall/sysret and sysenter/sysexit instructions
                 * is forbidden for this processes because they destroy
                 * the contents of %cs and %ss segment registers.
                 *
                 * Explicity disable them here and add a context handler
                 * to the process. Note that disabling
                 * them here means we can't use sysret or sysexit on
                 * the way out of this system call - so we force this
                 * thread to take the slow path (which doesn't make use
                 * of sysenter or sysexit) back out.
                 */
                kpreempt_disable();
                ldt_installctx(pp, NULL);
                cpu_fast_syscall_disable();
                cpu_fast_syscall_disable(NULL);
                ASSERT(curthread->t_post_sys != 0);
                kpreempt_enable();

        } else if (seli > pp->p_ldtlimit) {
                ASSERT(pp->p_pctx != NULL);

                /*
                 * Increase size of ldt to include seli.
                 */
                ldt_grow(pp, seli);
        }

        ASSERT(seli <= pp->p_ldtlimit);
        ldp = &pp->p_ldt[seli];

        /*
         * On the 64-bit kernel, this is where things get more subtle.
         * Recall that in the 64-bit kernel, when we enter the kernel we
         * deliberately -don't- reload the segment selectors we came in on
         * for %ds, %es, %fs or %gs. Messing with selectors is expensive,
         * and the underlying descriptors are essentially ignored by the
         * hardware in long mode - except for the base that we override with
         * the gsbase MSRs.
         *
         * However, there's one unfortunate issue with this rosy picture --
         * a descriptor that's not marked as 'present' will still generate
         * an #np when loading a segment register.
         *
         * Consider this case.  An lwp creates a harmless LDT entry, points
         * one of it's segment registers at it, then tells the kernel (here)
         * to delete it.  In the 32-bit kernel, the #np will happen on the
         * way back to userland where we reload the segment registers, and be
         * handled in kern_gpfault().  In the 64-bit kernel, the same thing
         * will happen in the normal case too.  However, if we're trying to
         * use a debugger that wants to save and restore the segment registers,
         * and the debugger things that we have valid segment registers, we
         * have the problem that the debugger will try and restore the
         * segment register that points at the now 'not present' descriptor
         * and will take a #np right there.
         *
         * We should obviously fix the debugger to be paranoid about
         * -not- restoring segment registers that point to bad descriptors;
         * however we can prevent the problem here if we check to see if any
         * of the segment registers are still pointing at the thing we're
         * destroying; if they are, return an error instead. (That also seems
         * a lot better failure mode than SIGKILL and a core file
         * from kern_gpfault() too.)
         */
        if (SI86SSD_PRES(ssd) == 0) {
```

```
600                 kthread_t *t;
601                 int bad = 0;
603                 /*
604                  * Look carefully at the segment registers of every lwp
605                  * in the process (they're all stopped by our caller).
606                  * If we're about to invalidate a descriptor that's still
607                  * being referenced by *any* of them, return an error,
608                  * rather than having them #gp on their way out of the kernel.
609                  */
610                 ASSERT(pp->p_lwprcnt == 1);
612                 mutex_enter(&pp->p_lock);
613                 t = pp->p_tlist;
614                 do {
615                         klwp_t *lwp = ttolwp(t);
616                         struct regs *rp = lwp->lwp_regs;
617 #if defined(__amd64)
618                         pcb_t *pcb = &lwp->lwp_pcb;
619 #endif
621                         if (ssd->sel == rp->r_cs || ssd->sel == rp->r_ss) {
622                                 bad = 1;
623                                 break;
624                         }
626 #if defined(__amd64)
627                         if (pcb->pcb_rupdate == 1) {
628                                 if (ssd->sel == pcb->pcb_ds ||
629                                     ssd->sel == pcb->pcb_es ||
630                                     ssd->sel == pcb->pcb_fs ||
631                                     ssd->sel == pcb->pcb_gs) {
632                                         bad = 1;
633                                         break;
634                                 }
635                         } else
636 #endif
637                         {
638                                 if (ssd->sel == rp->r_ds ||
639                                     ssd->sel == rp->r_es ||
640                                     ssd->sel == rp->r_fs ||
641                                     ssd->sel == rp->r_gs) {
642                                         bad = 1;
643                                         break;
644                                 }
645                         }
647                 } while ((t = t->t_forw) != pp->p_tlist);
648                 mutex_exit(&pp->p_lock);
650                 if (bad) {
651                         mutex_exit(&pp->p_ldtlock);
652                         return (EBUSY);
653                 }
654         }
656         /*
657          * If acc1 is zero, clear the descriptor (including the 'present' bit).
658          * Make sure we update the CPU-private copy of the LDT.
650          * If acc1 is zero, clear the descriptor (including the 'present' bit)
659          */
660         if (ssd->acc1 == 0) {
661                 rc  = ldt_update_segd(ldp, &null_udesc);
662                 kpreempt_disable();
663                 ldt_load();
664                 kpreempt_enable();
```



```
665                 mutex_exit(&pp->p_ldtlock);
666                 return (rc);
667         }
669         /*
670          * Check segment type, allow segment not present and
671          * only user DPL (3).
672          */
673         if (SI86SSD_DPL(ssd) != SEL_UPL) {
674                 mutex_exit(&pp->p_ldtlock);
675                 return (EINVAL);
676         }
667 #if defined(__amd64)
678         /*
679          * Do not allow 32-bit applications to create 64-bit mode code
680          * segments.
681          */
682         if (SI86SSD_ISUSEG(ssd) && ((SI86SSD_TYPE(ssd) >> 3) & 1) == 1 &&
683             SI86SSD_ISLONG(ssd)) {
684                 mutex_exit(&pp->p_ldtlock);
685                 return (EINVAL);
686         }
677 #endif /* __amd64 */
688         /*
689          * Set up a code or data user segment descriptor, making sure to update
690          * the CPU-private copy of the LDT.
680          * Set up a code or data user segment descriptor.
691          */
692         if (SI86SSD_ISUSEG(ssd)) {
693                 ssd_to_usd(ssd, &ndesc);
694                 rc = ldt_update_segd(ldp, &ndesc);
695                 kpreempt_disable();
696                 ldt_load();
697                 kpreempt_enable();
698                 mutex_exit(&pp->p_ldtlock);
699                 return (rc);
700         }
689 #if defined(__i386)
690         /*
691          * Allow a call gate only if the destination is in the LDT
692          * and the system is running in 32-bit legacy mode.
693          *
694          * In long mode 32-bit call gates are redefined as 64-bit call
695          * gates and the hw enforces that the target code selector
696          * of the call gate must be 64-bit selector. A #gp fault is
697          * generated if otherwise. Since we do not allow 32-bit processes
698          * to switch themselves to 64-bits we never allow call gates
699          * on 64-bit system system.
700          */
701         if (SI86SSD_TYPE(ssd) == SDT_SYSCGT && SELISLDT(ssd->ls)) {
704                 ssd_to_sgd(ssd, (gate_desc_t *)&ndesc);
705                 rc = ldt_update_segd(ldp, &ndesc);
702         mutex_exit(&pp->p_ldtlock);
707                 return (rc);
708         }
709 #endif  /* __i386 */
711         mutex_exit(&pp->p_ldtlock);
703         return (EINVAL);
704 }
```

```
706 /*
707  * Allocate new LDT for process just large enough to contain seli.  Note we
708  * allocate and grow LDT in PAGESIZE chunks. We do this to simplify the
709  * implementation and because on the hypervisor it's required, since the LDT
710  * must live on pages that have PROT_WRITE removed and which are given to the
711  * hypervisor.
712  *
713  * Note that we don't actually load the LDT into the current CPU here: it's done
714  * later by our caller.
716  * Allocate new LDT for process just large enough to contain seli.
717  * Note we allocate and grow LDT in PAGESIZE chunks. We do this
718  * to simplify the implementation and because on the hypervisor it's
719  * required, since the LDT must live on pages that have PROT_WRITE
720  * removed and which are given to the hypervisor.
715  */
716 static void
717 ldt_alloc(proc_t *pp, uint_t seli)
718 {
719         user_desc_t     *ldt;
720         size_t          ldtsz;
721         uint_t          nsels;

723         ASSERT(MUTEX_HELD(&pp->p_ldtlock));
724         ASSERT(pp->p_ldt == NULL);
725         ASSERT(pp->p_ldtlimit == 0);

727         /*
728          * Allocate new LDT just large enough to contain seli. The LDT must
729          * always be allocated in units of pages for KPTI.
730          */
731         ldtsz = P2ROUNDUP((seli + 1) * sizeof (user_desc_t), PAGESIZE);
732         nsels = ldtsz / sizeof (user_desc_t);
733         ASSERT(nsels >= MINNLDT && nsels <= MAXNLDT);

735         ldt = kmem_zalloc(ldtsz, KM_SLEEP);
736         ASSERT(IS_P2ALIGNED(ldt, PAGESIZE));

738 #if defined(__xpv)
739         if (xen_ldt_setprot(ldt, ldtsz, PROT_READ))
740                 panic("ldt_alloc:xen_ldt_setprot(PROT_READ) failed");
741 #endif

743         pp->p_ldt = ldt;
744         pp->p_ldtlimit = nsels - 1;
751         set_syssegd(&pp->p_ldt_desc, ldt, ldtsz - 1, SDT_SYSLDT, SEL_KPL);

753         if (pp == curproc) {
754                 kpreempt_disable();
755                 ldt_load();
756                 kpreempt_enable();
757         }
745 }

747 static void
748 ldt_free(proc_t *pp)
749 {
750         user_desc_t     *ldt;
751         size_t          ldtsz;

753         ASSERT(pp->p_ldt != NULL);

755         mutex_enter(&pp->p_ldtlock);
756         ldt = pp->p_ldt;
757         ldtsz = (pp->p_ldtlimit + 1) * sizeof (user_desc_t);

759         ASSERT(IS_P2ALIGNED(ldtsz, PAGESIZE));
```



```
761         pp->p_ldt = NULL;
762         pp->p_ldtlimit = 0;
776         pp->p_ldt_desc = null_sdesc;
763         mutex_exit(&pp->p_ldtlock);

765         if (pp == curproc) {
766                 kpreempt_disable();
767                 ldt_unload();
768                 kpreempt_enable();
769         }

771 #if defined(__xpv)
772         /*
773          * We are not allowed to make the ldt writable until after
774          * we tell the hypervisor to unload it.
775          */
776         if (xen_ldt_setprot(ldt, ldtsz, PROT_READ | PROT_WRITE))
777                 panic("ldt_free:xen_ldt_setprot(PROT_READ|PROT_WRITE) failed");
778 #endif

780         kmem_free(ldt, ldtsz);
781 }
```

unchanged_portion_omitted_

```
827 /*
828  * Note that we don't actually load the LDT into the current CPU here: it's done
829  * later by our caller - unless we take an error.  This works out because
830  * ldt_load() does a copy of ->p_ldt instead of directly loading it into the GDT
831  * (and therefore can't be using the freed old LDT), and by definition if the
832  * new entry didn't pass validation, then the proc shouldn't be referencing an
833  * entry in the extended region.
834  */
835 static void
836 ldt_grow(proc_t *pp, uint_t seli)
837 {
838         user_desc_t     *oldt, *nldt;
839         uint_t          nsels;
840         size_t          oldtsz, nldtsz;

842         ASSERT(MUTEX_HELD(&pp->p_ldtlock));
843         ASSERT(pp->p_ldt != NULL);
844         ASSERT(pp->p_ldtlimit != 0);

846         /*
847          * Allocate larger LDT just large enough to contain seli. The LDT must
848          * always be allocated in units of pages for KPTI.
849          */
850         nldtsz = P2ROUNDUP((seli + 1) * sizeof (user_desc_t), PAGESIZE);
851         nsels = nldtsz / sizeof (user_desc_t);
852         ASSERT(nsels >= MINNLDT && nsels <= MAXNLDT);
853         ASSERT(nsels > pp->p_ldtlimit);

855         oldt = pp->p_ldt;
856         oldtsz = (pp->p_ldtlimit + 1) * sizeof (user_desc_t);

858         nldt = kmem_zalloc(nldtsz, KM_SLEEP);
859         ASSERT(IS_P2ALIGNED(nldt, PAGESIZE));

861         bcopy(oldt, nldt, oldtsz);

863         /*
864          * unload old ldt.
865          */
866         kpreempt_disable();
867         ldt_unload();
```

```
 868         kpreempt_enable();

 870 #if defined(__xpv)

 872         /*
 873          * Make old ldt writable and new ldt read only.
 874          */
 875         if (xen_ldt_setprot(oldt, oldtsz, PROT_READ | PROT_WRITE))
 876                 panic("ldt_grow:xen_ldt_setprot(PROT_READ|PROT_WRITE) failed");

 878         if (xen_ldt_setprot(nldt, nldtsz, PROT_READ))
 879                 panic("ldt_grow:xen_ldt_setprot(PROT_READ) failed");
 880 #endif

 882         pp->p_ldt = nldt;
 883         pp->p_ldtlimit = nsels - 1;

 891         /*
 892          * write new ldt segment descriptor.
 893          */
 894         set_syssegd(&pp->p_ldt_desc, nldt, nldtsz - 1, SDT_SYSLDT, SEL_KPL);

 896         /*
 897          * load the new ldt.
 898          */
 899         kpreempt_disable();
 900         ldt_load();
 901         kpreempt_enable();

 885         kmem_free(oldt, oldtsz);
 886 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   25075 Thu Jun 14 17:16:00 2018
new/usr/src/uts/intel/sys/segments.h
9600 LDT still not happy under KPTI
**********************************************************
______unchanged_portion_omitted_

 384 #define GATESEG_GETOFFSET(sgd)  ((uintptr_t)((sgd)->sgd_looffset |      \
 385                                 (sgd)->sgd_hioffset << 16 |             \
 386                                 (uint64_t)((sgd)->sgd_hi64offset) << 32))

 388 #endif  /* __amd64 */

 390 /*
 391  * functions for initializing and updating segment descriptors.
 392  */
 393 #if defined(__amd64)

 395 extern void set_usegd(user_desc_t *, uint_t, void *, size_t, uint_t, uint_t,
 396     uint_t, uint_t);

 398 #elif defined(__i386)

 400 extern void set_usegd(user_desc_t *, void *, size_t, uint_t, uint_t,
 401     uint_t, uint_t);

 403 #endif  /* __i386 */

 405 extern uint_t idt_vector_to_ist(uint_t);

 407 extern void set_gatesegd(gate_desc_t *, void (*)(void), selector_t,
 408     uint_t, uint_t, uint_t);

 410 extern void set_syssegd(system_desc_t *, void *, size_t, uint_t, uint_t);

 412 extern void *get_ssd_base(system_desc_t *);

 414 extern void gdt_update_usegd(uint_t, user_desc_t *);

 416 extern int ldt_update_segd(user_desc_t *, user_desc_t *);

 418 #if defined(__xpv)

 420 extern int xen_idt_to_trap_info(uint_t, gate_desc_t *, void *);
 421 extern void xen_idt_write(gate_desc_t *, uint_t);

 423 #endif  /* __xen */

 425 void init_boot_gdt(user_desc_t *);

 427 #endif  /* _ASM */

 429 /*
 430  * Common segment parameter defintions for granularity, default
 431  * operand size and operaton mode.
 432  */
 433 #define SDP_BYTES       0       /* segment limit scaled to bytes */
 434 #define SDP_PAGES       1       /* segment limit scaled to pages */
 435 #define SDP_OP32        1       /* code and data default operand = 32 bits */
 436 #define SDP_LONG        1       /* long mode code segment (64 bits) */
 437 #define SDP_SHORT       0       /* compat/legacy code segment (32 bits) */
 438 /*
 439  * System segments and gate types.
 440  *
 441  * In long mode i386 32-bit ldt, tss, call, interrupt and trap gate
 442  * types are redefined into 64-bit equivalents.
```



```
 443  */
 444 #define SDT_SYSNULL      0      /* system null */
 445 #define SDT_SYS286TSS    1      /* system 286 TSS available */
 446 #define SDT_SYSLDT       2      /* system local descriptor table */
 447 #define SDT_SYS286BSY    3      /* system 286 TSS busy */
 448 #define SDT_SYS286CGT    4      /* system 286 call gate */
 449 #define SDT_SYSTASKGT    5      /* system task gate */
 450 #define SDT_SYS286IGT    6      /* system 286 interrupt gate */
 451 #define SDT_SYS286TGT    7      /* system 286 trap gate */
 452 #define SDT_SYSNULL2     8      /* system null again */
 453 #define SDT_SYSTSS       9      /* system TSS available */
 454 #define SDT_SYSNULL3    10      /* system null again */
 455 #define SDT_SYSTSSBSY   11      /* system TSS busy */
 456 #define SDT_SYSCGT      12      /* system call gate */
 457 #define SDT_SYSNULL4    13      /* system null again */
 458 #define SDT_SYSIGT      14      /* system interrupt gate */
 459 #define SDT_SYSTGT      15      /* system trap gate */

 461 /*
 462  * Memory segment types.
 463  *
 464  * While in long mode expand-down, writable and accessed type field
 465  * attributes are ignored. Only the conforming bit is loaded by hardware
 466  * for long mode code segment descriptors.
 467  */
 468 #define SDT_MEMRO       16      /* read only */
 469 #define SDT_MEMROA      17      /* read only accessed */
 470 #define SDT_MEMRW       18      /* read write */
 471 #define SDT_MEMRWA      19      /* read write accessed */
 472 #define SDT_MEMROD      20      /* read only expand dwn limit */
 473 #define SDT_MEMRODA     21      /* read only expand dwn limit accessed */
 474 #define SDT_MEMRWD      22      /* read write expand dwn limit */
 475 #define SDT_MEMRWDA     23      /* read write expand dwn limit accessed */
 476 #define SDT_MEME        24      /* execute only */
 477 #define SDT_MEMEA       25      /* execute only accessed */
 478 #define SDT_MEMER       26      /* execute read */
 479 #define SDT_MEMERA      27      /* execute read accessed */
 480 #define SDT_MEMEC       28      /* execute only conforming */
 481 #define SDT_MEMEAC      29      /* execute only accessed conforming */
 482 #define SDT_MEMERC      30      /* execute read conforming */
 483 #define SDT_MEMERAC     31      /* execute read accessed conforming */

 485 /* These bits are within the "type" field, like the values above. */
 486 #define SDT_A           0x01    /* accessed bit */
 487 #define SDT_S           0x10    /* S-bit at the top of "type" for usegs */

 489 /*
 490  * Entries in the Interrupt Descriptor Table (IDT)
 491  */
 492 #define IDT_DE          0       /* #DE: Divide Error */
 493 #define IDT_DB          1       /* #DB: Debug */
 494 #define IDT_NMI         2       /* Nonmaskable External Interrupt */
 495 #define IDT_BP          3       /* #BP: Breakpoint */
 496 #define IDT_OF          4       /* #OF: Overflow */
 497 #define IDT_BR          5       /* #BR: Bound Range Exceeded */
 498 #define IDT_UD          6       /* #UD: Undefined/Invalid Opcode */
 499 #define IDT_NM          7       /* #NM: No Math Coprocessor */
 500 #define IDT_DF          8       /* #DF: Double Fault */
 501 #define IDT_FPUGP       9       /* Coprocessor Segment Overrun */
 502 #define IDT_TS          10      /* #TS: Invalid TSS */
 503 #define IDT_NP          11      /* #NP: Segment Not Present */
 504 #define IDT_SS          12      /* #SS: Stack Segment Fault */
 505 #define IDT_GP          13      /* #GP: General Protection Fault */
 506 #define IDT_PF          14      /* #PF: Page Fault */
 507 #define IDT_MF          16      /* #MF: FPU Floating-Point Error */
 508 #define IDT_AC          17      /* #AC: Alignment Check */
```

```
509 #define IDT_MC          18      /* #MC: Machine Check */
510 #define IDT_XF          19      /* #XF: SIMD Floating-Point Exception */
511 #define NIDT            256     /* size in entries of IDT */

513 /*
514  * Entries in the Global Descriptor Table (GDT)
515  *
516  * We make sure to space the system descriptors (LDT's, TSS')
517  * such that they are double gdt slot aligned. This is because
518  * in long mode system segment decriptors expand to 128 bits.
519  *
520  * GDT_LWPFS and GDT_LWPGS must be the same for both 32 and 64-bit
521  * kernels. See setup_context in libc. 64-bit processes must set
522  * %fs or %gs to null selector to use 64-bit fsbase or gsbase
523  * respectively.
524  */
525 #define GDT_NULL        0       /* null */
526 #define GDT_B32DATA     1       /* dboot 32 bit data descriptor */
527 #define GDT_B32CODE     2       /* dboot 32 bit code descriptor */
528 #define GDT_B16CODE     3       /* bios call 16 bit code descriptor */
529 #define GDT_B16DATA     4       /* bios call 16 bit data descriptor */
530 #define GDT_B64CODE     5       /* dboot 64 bit code descriptor */
531 #define GDT_BGSTMP      7       /* kmdb descriptor only used early in boot */
532 #define GDT_CPUID       16      /* store numeric id of current CPU */

534 #if defined(__amd64)

536 #define GDT_KCODE       6       /* kernel code seg %cs */
537 #define GDT_KDATA       7       /* kernel data seg %ds */
538 #define GDT_U32CODE     8       /* 32-bit process on 64-bit kernel %cs */
539 #define GDT_UDATA       9       /* user data seg %ds (32 and 64 bit) */
540 #define GDT_UCODE       10      /* native user code  seg %cs */
541 #define GDT_LDT         12      /* (12-13) LDT for current process */
542 #define GDT_KTSS        14      /* (14-15) kernel tss */
543 #define GDT_FS          GDT_NULL /* kernel %fs segment selector */
544 #define GDT_GS          GDT_NULL /* kernel %gs segment selector */
545 #define GDT_LWPFS       55      /* lwp private %fs segment selector (32-bit) */
546 #define GDT_LWPGS       56      /* lwp private %gs segment selector (32-bit) */
547 #define GDT_BRANDMIN    57      /* first entry in GDT for brand usage */
548 #define GDT_BRANDMAX    61      /* last entry in GDT for brand usage */
549 #define NGDT            62      /* number of entries in GDT */

551 /*
552  * This selector is only used in the temporary GDT used to bring additional
553  * CPUs from 16-bit real mode into long mode in real_mode_start().
554  */
555 #define TEMPGDT_KCODE64 1       /* 64-bit code selector */

557 #elif defined(__i386)

559 #define GDT_LDT         40      /* LDT for current process */
560 #define GDT_KTSS        42      /* kernel tss */
561 #define GDT_KCODE       43      /* kernel code seg %cs */
562 #define GDT_KDATA       44      /* kernel data seg %ds */
563 #define GDT_UCODE       45      /* native user code  seg %cs */
564 #define GDT_UDATA       46      /* user data seg %ds (32 and 64 bit) */
565 #define GDT_DBFLT       47      /* double fault #DF selector */
566 #define GDT_FS          53      /* kernel %fs segment selector */
567 #define GDT_GS          54      /* kernel %gs segment selector */
568 #define GDT_LWPFS       55      /* lwp private %fs segment selector */
569 #define GDT_LWPGS       56      /* lwp private %gs segment selector */
570 #define GDT_BRANDMIN    57      /* first entry in GDT for brand usage */
571 #define GDT_BRANDMAX    61      /* last entry in GDT for brand usage */
572 #if !defined(__xpv)
573 #define NGDT            90      /* number of entries in GDT */
574 #else
```



```
575 #define NGDT            512     /* single 4K page for the hypervisor */
576 #endif

578 #endif  /* __i386 */

580 /*
581  * Convenient selector definitions.
582  */

584 /*
585  * XXPV 64 bit Xen only allows the guest %cs/%ss be the private ones it
586  * provides, not the ones we create for ourselves.  See FLAT_RING3_CS64 in
587  * public/arch-x86_64.h
588  *
589  * 64-bit Xen runs paravirtual guests in ring 3 but emulates them running in
590  * ring 0 by clearing CPL in %cs value pushed on guest exception stacks.
591  * Therefore we will have KCS_SEL value indicate ring 0 and use that everywhere
592  * in the kernel. But in the few files where we initialize segment registers or
593  * create and update descriptors we will explicity OR in SEL_KPL (ring 3) for
594  * kernel %cs. See desctbls.c for an example.
595  */

597 #if defined(__xpv) && defined(__amd64)
598 #define KCS_SEL         0xe030          /* FLAT_RING3_CS64 & 0xFFF0 */
599 #define KDS_SEL         0xe02b          /* FLAT_RING3_SS64 */
600 #else
601 #define KCS_SEL         SEL_GDT(GDT_KCODE, SEL_KPL)
602 #define KDS_SEL         SEL_GDT(GDT_KDATA, SEL_KPL)
603 #endif

605 #define UCS_SEL         SEL_GDT(GDT_UCODE, SEL_UPL)
606 #if defined(__amd64)
607 #define TEMP_CS64_SEL   SEL_GDT(TEMPGDT_KCODE64, SEL_KPL)
608 #define U32CS_SEL       SEL_GDT(GDT_U32CODE, SEL_UPL)
609 #endif

611 #define UDS_SEL         SEL_GDT(GDT_UDATA, SEL_UPL)
612 #define ULDT_SEL        SEL_GDT(GDT_LDT, SEL_KPL)
613 #define KTSS_SEL        SEL_GDT(GDT_KTSS, SEL_KPL)
614 #define DFTSS_SEL       SEL_GDT(GDT_DBFLT, SEL_KPL)
615 #define KFS_SEL         0
616 #define KGS_SEL         SEL_GDT(GDT_GS, SEL_KPL)
617 #define LWPFS_SEL       SEL_GDT(GDT_LWPFS, SEL_UPL)
618 #define LWPGS_SEL       SEL_GDT(GDT_LWPGS, SEL_UPL)
619 #define BRANDMIN_SEL    SEL_GDT(GDT_BRANDMIN, SEL_UPL)
620 #define BRANDMAX_SEL    SEL_GDT(GDT_BRANDMAX, SEL_UPL)

622 #define B64CODE_SEL     SEL_GDT(GDT_B64CODE, SEL_KPL)
623 #define B32CODE_SEL     SEL_GDT(GDT_B32CODE, SEL_KPL)
624 #define B32DATA_SEL     SEL_GDT(GDT_B32DATA, SEL_KPL)
625 #define B16CODE_SEL     SEL_GDT(GDT_B16CODE, SEL_KPL)
626 #define B16DATA_SEL     SEL_GDT(GDT_B16DATA, SEL_KPL)

628 /*
629  * Temporary %gs descriptor used by kmdb with -d option. Only lives
630  * in boot's GDT and is not copied into kernel's GDT from boot.
631  */
632 #define KMDBGS_SEL      SEL_GDT(GDT_BGSTMP, SEL_KPL)

634 /*
635  * Selector used for kdi_idt when kmdb has taken over the IDT.
636  */
637 #if defined(__amd64)
638 #define KMDBCODE_SEL    B64CODE_SEL
639 #else
640 #define KMDBCODE_SEL    B32CODE_SEL
```

```
641 #endif

643 /*
644  * Entries in default Local Descriptor Table (LDT) for every process.
645  */
646 #define LDT_SYSCALL     0       /* call gate for libc.a (obsolete) */
647 #define LDT_SIGCALL     1       /* EOL me, call gate for static sigreturn */
648 #define LDT_RESVD1      2       /* old user %cs */
649 #define LDT_RESVD2      3       /* old user %ds */
650 #define LDT_ALTSYSCALL  4       /* alternate call gate for system calls */
651 #define LDT_ALTSIGCALL  5       /* EOL me, alternate call gate for sigreturn */
652 #define LDT_UDBASE      6       /* user descriptor base index */
653 #define MINNLDT         512     /* Current min solaris ldt size (1 4K page) */
654 #define MAXNLDT         8192    /* max solaris ldt size (16 4K pages) */

656 #ifdef _KERNEL
657 #define LDT_CPU_SIZE    (16 * 4096)     /* Size of kernel per-CPU allocation */
658 #endif

660 #ifndef _ASM

662 extern  gate_desc_t     *idt0;
663 extern  desctbr_t       idt0_default_reg;
664 extern  user_desc_t     *gdt0;

666 extern user_desc_t      zero_udesc;
667 extern user_desc_t      null_udesc;
668 extern system_desc_t    null_sdesc;

670 #if defined(__amd64)
671 extern user_desc_t      zero_u32desc;
672 #endif
673 #if defined(__amd64)
674 extern user_desc_t      ucs_on;
675 extern user_desc_t      ucs_off;
676 extern user_desc_t      ucs32_on;
677 extern user_desc_t      ucs32_off;
678 #endif  /* __amd64 */

680 extern tss_t *ktss0;

682 #if defined(__i386)
683 extern tss_t *dftss0;
684 #endif  /* __i386 */

686 extern void div0trap(), dbgtrap(), nmiint(), brktrap(), ovflotrap();
687 extern void boundstrap(), invoptrap(), ndptrap();
688 #if !defined(__xpv)
689 extern void syserrtrap();
690 #endif
691 extern void invaltrap(), invtsstrap(), segnptrap(), stktrap();
692 extern void gptrap(), pftrap(), ndperr();
693 extern void overrun(), resvtrap();
694 extern void _start(), cmnint();
695 extern void achktrap(), mcetrap();
696 extern void xmtrap();
697 extern void fasttrap();
698 extern void dtrace_ret();

700 /* KPTI trampolines */
701 extern void tr_invaltrap();
702 extern void tr_div0trap(), tr_dbgtrap(), tr_nmiint(), tr_brktrap();
703 extern void tr_ovflotrap(), tr_boundstrap(), tr_invoptrap(), tr_ndptrap();
704 #if !defined(__xpv)
705 extern void tr_syserrtrap();
706 #endif
```



```
707 extern void tr_invaltrap(), tr_invtsstrap(), tr_segnptrap(), tr_stktrap();
708 extern void tr_gptrap(), tr_pftrap(), tr_ndperr();
709 extern void tr_overrun(), tr_resvtrap();
710 extern void tr_achktrap(), tr_mcetrap();
711 extern void tr_xmtrap();
712 extern void tr_fasttrap();
713 extern void tr_dtrace_ret();

715 #if !defined(__amd64)
716 extern void pentium_pftrap();
717 #endif

719 extern uint64_t kpti_enable;

721 #endif /* _ASM */

723 #ifdef  __cplusplus
724 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   32555 Thu Jun 14 17:16:00 2018
new/usr/src/uts/intel/sys/x86_archext.h
9600 LDT still not happy under KPTI
**********************************************************
______unchanged_portion_omitted_

 731 extern int x86_use_pcid;
 732 extern int x86_use_invpcid;

 734 /*
 735  * Utility functions to get/set extended control registers (XCR)
 736  * Initial use is to get/set the contents of the XFEATURE_ENABLED_MASK.
 737  */
 738 extern uint64_t get_xcr(uint_t);
 739 extern void set_xcr(uint_t, uint64_t);

 741 extern uint64_t rdmsr(uint_t);
 742 extern void wrmsr(uint_t, const uint64_t);
 743 extern uint64_t xrdmsr(uint_t);
 744 extern void xwrmsr(uint_t, const uint64_t);
 745 extern int checked_rdmsr(uint_t, uint64_t *);
 746 extern int checked_wrmsr(uint_t, uint64_t);

 748 extern void invalidate_cache(void);
 749 extern ulong_t getcr4(void);
 750 extern void setcr4(ulong_t);

 752 extern void mtrr_sync(void);

 754 extern void cpu_fast_syscall_enable(void);
 755 extern void cpu_fast_syscall_disable(void);
 754 extern void cpu_fast_syscall_enable(void *);
 755 extern void cpu_fast_syscall_disable(void *);

 757 struct cpu;

 759 extern int cpuid_checkpass(struct cpu *, int);
 760 extern uint32_t cpuid_insn(struct cpu *, struct cpuid_regs *);
 761 extern uint32_t __cpuid_insn(struct cpuid_regs *);
 762 extern int cpuid_getbrandstr(struct cpu *, char *, size_t);
 763 extern int cpuid_getidstr(struct cpu *, char *, size_t);
 764 extern const char *cpuid_getvendorstr(struct cpu *);
 765 extern uint_t cpuid_getvendor(struct cpu *);
 766 extern uint_t cpuid_getfamily(struct cpu *);
 767 extern uint_t cpuid_getmodel(struct cpu *);
 768 extern uint_t cpuid_getstep(struct cpu *);
 769 extern uint_t cpuid_getsig(struct cpu *);
 770 extern uint_t cpuid_get_ncpu_per_chip(struct cpu *);
 771 extern uint_t cpuid_get_ncore_per_chip(struct cpu *);
 772 extern uint_t cpuid_get_ncpu_sharing_last_cache(struct cpu *);
 773 extern id_t cpuid_get_last_lvl_cacheid(struct cpu *);
 774 extern int cpuid_get_chipid(struct cpu *);
 775 extern id_t cpuid_get_coreid(struct cpu *);
 776 extern int cpuid_get_pkgcoreid(struct cpu *);
 777 extern int cpuid_get_clogid(struct cpu *);
 778 extern int cpuid_get_cacheid(struct cpu *);
 779 extern uint32_t cpuid_get_apicid(struct cpu *);
 780 extern uint_t cpuid_get_procnodeid(struct cpu *cpu);
 781 extern uint_t cpuid_get_procnodes_per_pkg(struct cpu *cpu);
 782 extern uint_t cpuid_get_compunitid(struct cpu *cpu);
 783 extern uint_t cpuid_get_cores_per_compunit(struct cpu *cpu);
 784 extern size_t cpuid_get_xsave_size();
 785 extern boolean_t cpuid_need_fp_excp_handling();
 786 extern int cpuid_is_cmt(struct cpu *);
 787 extern int cpuid_syscall32_insn(struct cpu *);
```



```
 788 extern int getl2cacheinfo(struct cpu *, int *, int *, int *);

 790 extern uint32_t cpuid_getchiprev(struct cpu *);
 791 extern const char *cpuid_getchiprevstr(struct cpu *);
 792 extern uint32_t cpuid_getsockettype(struct cpu *);
 793 extern const char *cpuid_getsocketstr(struct cpu *);

 795 extern int cpuid_have_cr8access(struct cpu *);

 797 extern int cpuid_opteron_erratum(struct cpu *, uint_t);

 799 struct cpuid_info;

 801 extern void setx86isalist(void);
 802 extern void cpuid_alloc_space(struct cpu *);
 803 extern void cpuid_free_space(struct cpu *);
 804 extern void cpuid_pass1(struct cpu *, uchar_t *);
 805 extern void cpuid_pass2(struct cpu *);
 806 extern void cpuid_pass3(struct cpu *);
 807 extern void cpuid_pass4(struct cpu *, uint_t *);
 808 extern void cpuid_set_cpu_properties(void *, processorid_t,
 809     struct cpuid_info *);

 811 extern void cpuid_get_addrsize(struct cpu *, uint_t *, uint_t *);
 812 extern uint_t cpuid_get_dtlb_nent(struct cpu *, size_t);

 814 #if !defined(__xpv)
 815 extern uint32_t *cpuid_mwait_alloc(struct cpu *);
 816 extern void cpuid_mwait_free(struct cpu *);
 817 extern int cpuid_deep_cstates_supported(void);
 818 extern int cpuid_arat_supported(void);
 819 extern int cpuid_iepb_supported(struct cpu *);
 820 extern int cpuid_deadline_tsc_supported(void);
 821 extern void vmware_port(int, uint32_t *);
 822 #endif

 824 struct cpu_ucode_info;

 826 extern void ucode_alloc_space(struct cpu *);
 827 extern void ucode_free_space(struct cpu *);
 828 extern void ucode_check(struct cpu *);
 829 extern void ucode_cleanup();

 831 #if !defined(__xpv)
 832 extern  char _tsc_mfence_start;
 833 extern  char _tsc_mfence_end;
 834 extern  char _tscp_start;
 835 extern  char _tscp_end;
 836 extern  char _no_rdtsc_start;
 837 extern  char _no_rdtsc_end;
 838 extern  char _tsc_lfence_start;
 839 extern  char _tsc_lfence_end;
 840 #endif

 842 #if !defined(__xpv)
 843 extern  char bcopy_patch_start;
 844 extern  char bcopy_patch_end;
 845 extern  char bcopy_ck_size;
 846 #endif

 848 extern void post_startup_cpu_fixups(void);

 850 extern uint_t workaround_errata(struct cpu *);

 852 #if defined(OPTERON_ERRATUM_93)
 853 extern int opteron_erratum_93;
```

```
 854 #endif

 856 #if defined(OPTERON_ERRATUM_91)
 857 extern int opteron_erratum_91;
 858 #endif

 860 #if defined(OPTERON_ERRATUM_100)
 861 extern int opteron_erratum_100;
 862 #endif

 864 #if defined(OPTERON_ERRATUM_121)
 865 extern int opteron_erratum_121;
 866 #endif

 868 #if defined(OPTERON_WORKAROUND_6323525)
 869 extern int opteron_workaround_6323525;
 870 extern void patch_workaround_6323525(void);
 871 #endif

 873 #if !defined(__xpv)
 874 extern void determine_platform(void);
 875 #endif
 876 extern int get_hwenv(void);
 877 extern int is_controldom(void);

 879 extern void enable_pcid(void);

 881 extern void xsave_setup_msr(struct cpu *);

 883 /*
 884  * Hypervisor signatures
 885  */
 886 #define HVSIG_XEN_HVM   "XenVMMXenVMM"
 887 #define HVSIG_VMWARE    "VMwareVMware"
 888 #define HVSIG_KVM       "KVMKVMKVM"
 889 #define HVSIG_MICROSOFT "Microsoft Hv"

 891 /*
 892  * Defined hardware environments
 893  */
 894 #define HW_NATIVE       (1 << 0)        /* Running on bare metal */
 895 #define HW_XEN_PV       (1 << 1)        /* Running on Xen PVM */

 897 #define HW_XEN_HVM      (1 << 2)        /* Running on Xen HVM */
 898 #define HW_VMWARE       (1 << 3)        /* Running on VMware hypervisor */
 899 #define HW_KVM          (1 << 4)        /* Running on KVM hypervisor */
 900 #define HW_MICROSOFT    (1 << 5)        /* Running on Microsoft hypervisor */

 902 #define HW_VIRTUAL      (HW_XEN_HVM | HW_VMWARE | HW_KVM | HW_MICROSOFT)

 904 #endif  /* _KERNEL */

 906 #endif  /* !_ASM */

 908 /*
 909  * VMware hypervisor related defines
 910  */
 911 #define VMWARE_HVMAGIC          0x564d5868
 912 #define VMWARE_HVPORT           0x5658
 913 #define VMWARE_HVCMD_GETVERSION 0x0a
 914 #define VMWARE_HVCMD_GETTSCFREQ 0x2d

 916 #ifdef  __cplusplus
 917 }
```

_____***unchanged_portion_omitted_***