```
**********************************************************
   50363 Wed Jul 11 18:26:05 2012
new/usr/src/cmd/savecore/savecore.c
2445 savecore fails with dump archive but return EXIT_OK
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright (c) 1983, 2010, Oracle and/or its affiliates. All rights reserved.
  23  */
  24 /*
  25  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  26  */

  28 #include <stdio.h>
  29 #include <stdlib.h>
  30 #include <stdarg.h>
  31 #include <unistd.h>
  32 #include <fcntl.h>
  33 #include <errno.h>
  34 #include <string.h>
  35 #include <deflt.h>
  36 #include <time.h>
  37 #include <syslog.h>
  38 #include <stropts.h>
  39 #include <pthread.h>
  40 #include <limits.h>
  41 #include <atomic.h>
  42 #include <libnvpair.h>
  43 #include <libintl.h>
  44 #include <sys/mem.h>
  45 #include <sys/statvfs.h>
  46 #include <sys/dumphdr.h>
  47 #include <sys/dumpadm.h>
  48 #include <sys/compress.h>
  49 #include <sys/panic.h>
  50 #include <sys/sysmacros.h>
  51 #include <sys/stat.h>
  52 #include <sys/resource.h>
  53 #include <bzip2/bzlib.h>
  54 #include <sys/fm/util.h>
  55 #include <fm/libfmevent.h>
  56 #include <sys/int_fmtio.h>


  59 /* fread/fwrite buffer size */
  60 #define FBUFSIZE                (1ULL << 20)
```



```
  62 /* minimum size for output buffering */
  63 #define MINCOREBLKSIZE          (1ULL << 17)

  65 /* create this file if metrics collection is enabled in the kernel */
  66 #define METRICSFILE "METRICS.csv"

  68 static char     progname[9] = "savecore";
  69 static char     *savedir;               /* savecore directory */
  70 static char     *dumpfile;              /* source of raw crash dump */
  71 static long     bounds = -1;            /* numeric suffix */
  72 static long     pagesize;               /* dump pagesize */
  73 static int      dumpfd = -1;            /* dumpfile descriptor */
  74 static dumphdr_t corehdr, dumphdr;      /* initial and terminal dumphdrs */
  75 static boolean_t dump_incomplete;       /* dumphdr indicates incomplete */
  76 static boolean_t fm_panic;              /* dump is the result of fm_panic */
  77 static offset_t endoff;                 /* offset of end-of-dump header */
  78 static int      verbose;                /* chatty mode */
  79 static int      disregard_valid_flag;   /* disregard valid flag */
  80 static int      livedump;               /* dump the current running system */
  81 static int      interactive;            /* user invoked; no syslog */
  82 static int      csave;                  /* save dump compressed */
  83 static int      filemode;               /* processing file, not dump device */
  84 static int      percent_done;           /* progress indicator */
  85 static hrtime_t startts;                /* timestamp at start */
  86 static volatile uint64_t saved;         /* count of pages written */
  87 static volatile uint64_t zpages;        /* count of zero pages not written */
  88 static dumpdatahdr_t datahdr;           /* compression info */
  89 static long     coreblksize;            /* preferred write size (st_blksize) */
  90 static int      cflag;                  /* run as savecore -c */
  91 static int      mflag;                  /* run as savecore -m */

  93 /*
  94  * Payload information for the events we raise.  These are used
  95  * in raise_event to determine what payload to include.
  96  */
  97 #define SC_PAYLOAD_SAVEDIR      0x0001  /* Include savedir in event */
  98 #define SC_PAYLOAD_INSTANCE     0x0002  /* Include bounds instance number */
  99 #define SC_PAYLOAD_IMAGEUUID    0x0004  /* Include dump OS instance uuid */
 100 #define SC_PAYLOAD_CRASHTIME    0x0008  /* Include epoch crashtime */
 101 #define SC_PAYLOAD_PANICSTR     0x0010  /* Include panic string */
 102 #define SC_PAYLOAD_PANICSTACK   0x0020  /* Include panic string */
 103 #define SC_PAYLOAD_FAILREASON   0x0040  /* Include failure reason */
 104 #define SC_PAYLOAD_DUMPCOMPLETE 0x0080  /* Include completeness indicator */
 105 #define SC_PAYLOAD_ISCOMPRESSED 0x0100  /* Dump is in vmdump.N form */
 106 #define SC_PAYLOAD_DUMPADM_EN   0x0200  /* Is dumpadm enabled or not? */
 107 #define SC_PAYLOAD_FM_PANIC     0x0400  /* Panic initiated by FMA */
 108 #define SC_PAYLOAD_JUSTCHECKING 0x0800  /* Run with -c flag? */

 110 enum sc_event_type {
 111         SC_EVENT_DUMP_PENDING,
 112         SC_EVENT_SAVECORE_FAILURE,
 113         SC_EVENT_DUMP_AVAILABLE
 114 };
______unchanged_portion_omitted_

 168 #define SC_SL_NONE      0x0001  /* no syslog */
 169 #define SC_SL_ERR       0x0002  /* syslog if !interactive, LOG_ERR */
 170 #define SC_SL_WARN      0x0004  /* syslog if !interactive, LOG_WARNING */
 171 #define SC_IF_VERBOSE   0x0008  /* message only if -v */
 172 #define SC_IF_ISATTY    0x0010  /* message only if interactive */
 173 #define SC_EXIT_OK      0x0020  /* exit(0) */
 174 #define SC_EXIT_ERR     0x0040  /* exit(1) */
 175 #define SC_EXIT_PEND    0x0080  /* exit(2) */
 176 #define SC_EXIT_FM      0x0100  /* exit(3) */

 178 #define _SC_ALLEXIT     (SC_EXIT_OK | SC_EXIT_ERR | SC_EXIT_PEND | SC_EXIT_FM)
```

```
180 static void
181 logprint(uint32_t flags, char *message, ...)
182 {
183         va_list args;
184         char buf[1024];
185         int do_always = ((flags & (SC_IF_VERBOSE | SC_IF_ISATTY)) == 0);
186         int do_ifverb = (flags & SC_IF_VERBOSE) && verbose;
187         int do_ifisatty = (flags & SC_IF_ISATTY) && interactive;
188         int code;
189         static int logprint_raised = 0;

191         if (do_always || do_ifverb || do_ifisatty) {
192                 va_start(args, message);
193                 /*LINTED: E_SEC_PRINTF_VAR_FMT*/
194                 (void) vsnprintf(buf, sizeof (buf), message, args);
195                 (void) fprintf(stderr, "%s: %s\n", progname, buf);
196                 if (!interactive) {
197                         switch (flags & (SC_SL_NONE | SC_SL_ERR | SC_SL_WARN)) {
198                         case SC_SL_ERR:
199                                 /*LINTED: E_SEC_PRINTF_VAR_FMT*/
200                                 syslog(LOG_ERR, buf);
201                                 break;

203                         case SC_SL_WARN:
204                                 /*LINTED: E_SEC_PRINTF_VAR_FMT*/
205                                 syslog(LOG_WARNING, buf);
206                                 break;

208                         default:
209                                 break;
210                         }
211                 }
212                 va_end(args);
213         }

215         switch (flags & _SC_ALLEXIT) {
216         case 0:
217                 return;

219         case SC_EXIT_OK:
220                 code = 0;
221                 break;

223         case SC_EXIT_PEND:
224                 /*
225                  * Raise an ireport saying why we are exiting.  Do not
226                  * raise if run as savecore -m.  If something in the
227                  * raise_event codepath calls logprint avoid recursion.
228                  */
229                 if (!mflag && logprint_raised++ == 0)
230                         raise_event(SC_EVENT_SAVECORE_FAILURE, buf);
231                 code = 2;
232                 break;

234         case SC_EXIT_FM:
235                 code = 3;
236                 break;

238         case SC_EXIT_ERR:
239         default:
230                 /*
231                  * Raise an ireport saying why we are exiting.  Do not
232                  * raise if run as savecore -m.  If something in the
233                  * raise_event codepath calls logprint avoid recursion.
234                  */
```



```
240                 if (!mflag && logprint_raised++ == 0)
241                         raise_event(SC_EVENT_SAVECORE_FAILURE, buf);
242                 code = 1;
243                 break;
244         }

246         exit(code);
247 }
```

*_____unchanged_portion_omitted_*

```
350 static void
351 read_dumphdr(void)
352 {
353         if (filemode)
354                 dumpfd = Open(dumpfile, O_RDONLY, 0644);
355         else
356                 dumpfd = Open(dumpfile, O_RDWR | O_DSYNC, 0644);
357         endoff = llseek(dumpfd, -DUMP_OFFSET, SEEK_END) & -DUMP_OFFSET;
358         Pread(dumpfd, &dumphdr, sizeof (dumphdr), endoff);
359         Pread(dumpfd, &datahdr, sizeof (datahdr), endoff + sizeof (dumphdr));

361         pagesize = dumphdr.dump_pagesize;

363         if (dumphdr.dump_magic != DUMP_MAGIC)
364                 logprint(SC_SL_NONE | SC_EXIT_PEND, "bad magic number %x",
359                 logprint(SC_SL_NONE | SC_EXIT_OK, "bad magic number %x",
365                     dumphdr.dump_magic);

367         if ((dumphdr.dump_flags & DF_VALID) == 0 && !disregard_valid_flag)
368                 logprint(SC_SL_NONE | SC_IF_VERBOSE | SC_EXIT_OK,
369                     "dump already processed");

371         if (dumphdr.dump_version != DUMP_VERSION)
372                 logprint(SC_SL_NONE | SC_IF_VERBOSE | SC_EXIT_PEND,
367                 logprint(SC_SL_NONE | SC_IF_VERBOSE | SC_EXIT_OK,
373                     "dump version (%d) != %s version (%d)",
374                     dumphdr.dump_version, progname, DUMP_VERSION);

376         if (dumphdr.dump_wordsize != DUMP_WORDSIZE)
377                 logprint(SC_SL_NONE | SC_EXIT_PEND,
372                 logprint(SC_SL_NONE | SC_EXIT_OK,
378                     "dump is from %u-bit kernel - cannot save on %u-bit kernel",
379                     dumphdr.dump_wordsize, DUMP_WORDSIZE);

381         if (datahdr.dump_datahdr_magic == DUMP_DATAHDR_MAGIC) {
382                 if (datahdr.dump_datahdr_version != DUMP_DATAHDR_VERSION)
383                         logprint(SC_SL_NONE | SC_IF_VERBOSE | SC_EXIT_PEND,
378                         logprint(SC_SL_NONE | SC_IF_VERBOSE | SC_EXIT_OK,
384                             "dump data version (%d) != %s data version (%d)",
385                             datahdr.dump_datahdr_version, progname,
386                             DUMP_DATAHDR_VERSION);
387         } else {
388                 (void) memset(&datahdr, 0, sizeof (datahdr));
389                 datahdr.dump_maxcsize = pagesize;
390         }

392         /*
393          * Read the initial header, clear the valid bits, and compare headers.
394          * The main header may have been overwritten by swapping if we're
395          * using a swap partition as the dump device, in which case we bail.
396          */
397         Pread(dumpfd, &corehdr, sizeof (dumphdr_t), dumphdr.dump_start);

399         corehdr.dump_flags &= ~DF_VALID;
400         dumphdr.dump_flags &= ~DF_VALID;
```

```
402         if (memcmp(&corehdr, &dumphdr, sizeof (dumphdr_t)) != 0) {
403                 /*
404                  * Clear valid bit so we don't complain on every invocation.
405                  */
406                 if (!filemode)
407                         Pwrite(dumpfd, &dumphdr, sizeof (dumphdr), endoff);
408                 logprint(SC_SL_ERR | SC_EXIT_ERR,
409                     "initial dump header corrupt");
410         }
411 }
```

unchanged_portion_omitted_