```
 ...
 36: /*
 37:  * Write options.
 38:  */
 39: #define SNDZERO        0x001        /* send a zero length message */
 40: #define SNDPIPE        0x002        /* send SIGPIPE on write and */
 41:                      /* putmsg if sd_werror is set */
 42:
 43: /*
 44:  * Read options
 45:  */
 46:
 47: #define RNORM        0x000        /* read msg norm */
 48: #define RMSGD        0x001        /* read msg discard */
 49: #define RMSGN        0x002        /* read msg no discard */
 50:
 51: #define RMODEMASK    0x003        /* all above bits */
 52:
 ...
 72:
 73: /*
 74:  * Flush options
 75:  */
 76:
 77: #define FLUSHR        0x01        /* flush read queue */
 78: #define FLUSHW        0x02        /* flush write queue */
 79: #define FLUSHRW        0x03        /* flush both queues */
 80: #define FLUSHBAND    0x04        /* flush only band specified */
 81:                      /* in next byte */
 82:
 83: /*
 84:  * Events for which to be sent SIGPOLL signal and for which events
 85:  * can be posted by the I_SETEV ioctl.
 86:  */
 87: #define S_INPUT        0x0001        /* any msg but hipri on read Q */
 88: #define S_HIPRI        0x0002        /* high priority msg on read Q */
 89: #define S_OUTPUT    0x0004        /* write Q no longer full */
 90: #define S_MSG        0x0008        /* signal msg at front of read Q */
 91: #define    S_ERROR        0x0010        /* error msg arrived at stream head */
 92: #define    S_HANGUP    0x0020        /* hangup msg arrived at stream head */
 93: #define S_RDNORM    0x0040        /* normal msg on read Q */
 94: #define S_WRNORM    S_OUTPUT
 95: #define    S_RDBAND    0x0080        /* out of band msg on read Q */
 96: #define S_WRBAND    0x0100        /* can write out of band */
 97: #define S_BANDURG    0x0200        /* modifier to S_RDBAND, to generate */
 98:                      /* SIGURG instead of SIGPOLL */
 99:
 ...
133:
134: /*
135:  * Flag definitions for the I_ATMARK ioctl.
136:  */
137: #define ANYMARK        0x01
138: #define LASTMARK    0x02
139:
140: /*
141:  *  Stream Ioctl defines
142:  */
143: #define    STR        ('S'<<8)
144: /* (STR|000) in use */
145: #define I_NREAD        (STR|01)
146: #define I_PUSH        (STR|02)
147: #define I_POP        (STR|03)
148: #define I_LOOK        (STR|04)
149: #define I_FLUSH        (STR|05)
150: #define I_SRDOPT    (STR|06)
151: #define I_GRDOPT    (STR|07)
152: #define I_STR        (STR|010)
153: #define I_SETSIG    (STR|011)
154: #define I_GETSIG    (STR|012)
155: #define I_FIND        (STR|013)
156: #define I_LINK        (STR|014)
157: #define I_UNLINK    (STR|015)
158: /* (STR|016) in use */
159: #define I_PEEK        (STR|017)
160: #define I_FDINSERT    (STR|020)
161: #define I_SENDFD    (STR|021)
162: #if defined(_KERNEL)
```

```
  1: /* Copyright (C) 1998, 1999, 2000 Free Software Foundation, Inc.
  2:    This file is part of the GNU C Library.
  3:
  4:    The GNU C Library is free software; you can redistribute it and/or
  5:    modify it under the terms of the GNU Lesser General Public
  6:    License as published by the Free Software Foundation; either
  7:    version 2.1 of the License, or (at your option) any later version.
  8:
  9:    The GNU C Library is distributed in the hope that it will be useful,
 10:    but WITHOUT ANY WARRANTY; without even the implied warranty of
 11:    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 12:    Lesser General Public License for more details.
 13:
 14:    You should have received a copy of the GNU Lesser General Public
 15:    License along with the GNU C Library; if not, write to the Free
 16:    Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
 17:    02111-1307 USA.  */
 18:
 19: #ifndef _BITS_STROPTS_H
 20: #define _BITS_STROPTS_H    1
 21:
 22: #include <bits/types.h>
 23:
 24: /* Macros used as `request' argument to `ioctl'.  */
 25: #define __SID        ('S' << 8)
 26:
 27: #define I_NREAD        (__SID | 1)    /* Counts the number of data bytes in the data
 28:                       block in the first message.  */
 29: #define I_PUSH        (__SID | 2)    /* Push STREAMS module onto top of the current
 30:                       STREAM, just below the STREAM head.  */
 31: #define I_POP        (__SID | 3)    /* Remove STREAMS module from just below the
 32:                       STREAM head.  */
 33: #define I_LOOK        (__SID | 4)    /* Retrieve the name of the module just below
 34:                       the STREAM head and place it in a character
 35:                       string.  */
 36: #define I_FLUSH        (__SID | 5)    /* Flush all input and/or output.  */
 37: #define I_SRDOPT    (__SID | 6)    /* Sets the read mode.  */
 38: #define I_GRDOPT    (__SID | 7)    /* Returns the current read mode setting.  */
 39: #define I_STR        (__SID | 8)    /* Construct an internal STREAMS `ioctl'
 40:                       message and send that message downstream. */
 41: #define I_SETSIG    (__SID | 9)    /* Inform the STREAM head that the process
 42:                       wants the SIGPOLL signal issued.  */
 43: #define I_GETSIG    (__SID |10) /* Return the events for which the calling
 44:                       process is currently registered to be sent
 45:                       a SIGPOLL signal.  */
 46: #define I_FIND        (__SID |11) /* Compares the names of all modules currently
 47:                       present in the STREAM to the name pointed to
 48:                       by `arg'.  */
 49: #define I_LINK        (__SID |12) /* Connect two STREAMs.  */
 50: #define I_UNLINK    (__SID |13) /* Disconnects the two STREAMs.  */
 51: #define I_PEEK        (__SID |15) /* Allows a process to retrieve the information
 52:                       in the first message on the STREAM head read
 53:                       queue without taking the message off the
 54:                       queue.  */
 55: #define I_FDINSERT  (__SID |16) /* Create a message from the specified
 56:                       buffer(s), adds information about another
 57:                       STREAM, and send the message downstream.  */
 58: #define I_SENDFD    (__SID |17) /* Requests the STREAM associated with `fildes'
 59:                       to send a message, containing a file
 60:                       pointer, to the STREAM head at the other end
 61:                       of a STREAMS pipe.  */
 62: #define I_RECVFD    (__SID |14) /* Non-EFT definition.  */
 63: #define I_SWROPT    (__SID |19) /* Set the write mode.  */
 64: #define I_GWROPT    (__SID |20) /* Return the current write mode setting.  */
 65: #define I_LIST        (__SID |21) /* List all the module names on the STREAM, up
 66:                       to and including the topmost driver name. */
 67: #define I_PLINK        (__SID |22) /* Connect two STREAMs with a persistent
 68:                       link.  */
 69: #define I_PUNLINK   (__SID |23) /* Disconnect the two STREAMs that were
 70:                       connected with a persistent link.  */
 71: #define I_FLUSHBAND (__SID |28) /* Flush only band specified.  */
 72: #define I_CKBAND    (__SID |29) /* Check if the message of a given priority
 73:                       band exists on the STREAM head read
 74:                       queue.  */
 75: #define I_GETBAND   (__SID |30) /* Return the priority band of the first
 76:                       message on the STREAM head read queue.  */
 77: #define I_ATMARK    (__SID |31) /* See if the current message on the STREAM
 78:                       head read queue is "marked" by some module
```


(For readability, not all matches are indicated.)

```
163:
164: #define I_RECVFD    (STR|022)
165: #define I_E_RECVFD    (STR|016)
166: #elif !defined(_STYPES)    /* user level definition */
167:
168: #define I_RECVFD    (STR|016)    /* maps to kernel I_E_RECVFD */
169:
170: #else
171:
172: #define I_RECVFD    (STR|022)    /* non-EFT definition */
173:
174: #endif /* defined(_KERNEL) */
175:
176: #define I_SWROPT    (STR|023)
177: #define I_GWROPT    (STR|024)
178: #define I_LIST        (STR|025)
179: #define I_PLINK        (STR|026)
180: #define I_PUNLINK    (STR|027)
181: #define I_FLUSHBAND    (STR|034)
182: #define I_CKBAND    (STR|035)
183: #define I_GETBAND    (STR|036)
184: #define I_ATMARK    (STR|037)
185: #define I_SETCLTIME    (STR|040)
186: #define I_GETCLTIME    (STR|041)
187: #define I_CANPUT    (STR|042)
188: #define    I_S_RECVFD    (STR|043)
189: /*#define STRPERF    1 */
190: #ifdef STRPERF
191: #define I_STATS        (STR|044)
192: #endif
 ...
209: /*
210:  * Value for timeouts (ioctl, select) that denotes infinity
211:  */
212: #define INFTIM        -1
213:
214: /*
215:  * Stream buffer structure for putmsg and getmsg system calls
216:  */
217: struct strbuf {
218:     int    maxlen;            /* no. of bytes in buffer */
219:     int    len;            /* no. of bytes returned */
220:     char    *buf;            /* pointer to data */
221: };
222:
223: /*
224:  * Stream I_PEEK ioctl format
225:  */
226: struct strpeek {
227:     struct strbuf ctlbuf;
228:     struct strbuf databuf;
229:     long          flags;
230: };
231:
232: /*
233:  * Stream I_FDINSERT ioctl format
234:  */
235: struct strfdinsert {
236:     struct strbuf ctlbuf;
237:     struct strbuf databuf;
238:     long          flags;
239:     int          fildes;
240:     int          offset;
241: };
242:
243: /*
244:  * Receive file descriptor structure
245:  */
246:
 ...
293:
294: #elif !defined(_STYPES)     /* EFT user definition */
295:
296: struct strrecvfd {
297:     int fd;
298:     uid_t uid;
299:     gid_t gid;
300:     char fill[8];
```

```
 79:                    downstream.  */
 80: #define I_SETCLTIME (__SID |32) /* Set the time the STREAM head will delay when
 81:                    a STREAM is closing and there is data on
 82:                    the write queues.  */
 83: #define I_GETCLTIME (__SID |33) /* Get current value for closing timeout.  */
 84: #define I_CANPUT    (__SID |34) /* Check if a certain band is writable.  */
 85:
 86:
 87: /* Used in `I_LOOK' request.  */
 88: #define FMNAMESZ    8    /* compatibility w/UnixWare/Solaris.  */
 89:
 90: /* Flush options.  */
 91: #define FLUSHR        0x01    /* Flush read queues.  */
 92: #define FLUSHW        0x02    /* Flush write queues.  */
 93: #define FLUSHRW        0x03    /* Flush read and write queues.  */
 94: #ifdef __USE_GNU
 95: # define FLUSHBAND    0x04    /* Flush only specified band.  */
 96: #endif
 97:
 98: /* Possible arguments for `I_SETSIG'.  */
 99: #define S_INPUT        0x0001    /* A message, other than a high-priority
100:                    message, has arrived.  */
101: #define S_HIPRI        0x0002    /* A high-priority message is present.  */
102: #define S_OUTPUT    0x0004    /* The write queue for normal data is no longer
103:                    full.  */
104: #define S_MSG        0x0008    /* A STREAMS signal message that contains the
105:                    SIGPOLL signal reaches the front of the
106:                    STREAM head read queue.  */
107: #define S_ERROR        0x0010    /* Notification of an error condition.  */
108: #define S_HANGUP    0x0020    /* Notification of a hangup.  */
109: #define S_RDNORM    0x0040    /* A normal message has arrived.  */
110: #define S_WRNORM    S_OUTPUT
111: #define S_RDBAND    0x0080    /* A message with a non-zero priority has
112:                    arrived.  */
113: #define S_WRBAND    0x0100    /* The write queue for a non-zero priority
114:                    band is no longer full.  */
115: #define S_BANDURG    0x0200    /* When used in conjunction with S_RDBAND,
116:                    SIGURG is generated instead of SIGPOLL when
117:                    a priority message reaches the front of the
118:                    STREAM head read queue.  */
119:
120: /* Option for `I_PEEK'.  */
121: #define RS_HIPRI    0x01    /* Only look for high-priority messages.  */
122:
123: /* Options for `I_SRDOPT'.  */
124: #define RNORM        0x0000    /* Byte-STREAM mode, the default.  */
125: #define RMSGD        0x0001    /* Message-discard mode.  */
126: #define RMSGN        0x0002    /* Message-nondiscard mode.  */
127: #define RPROTDAT    0x0004    /* Deliver the control part of a message as
128:                    data.  */
129: #define RPROTDIS    0x0008    /* Discard the control part of a message,
130:                    delivering any data part.  */
131: #define RPROTNORM    0x0010    /* Fail `read' with EBADMSG if a message
132:                    containing a control part is at the front
133:                    of the STREAM head read queue.  */
134: #ifdef __USE_GNU
135: # define RPROTMASK    0x001C    /* The RPROT bits */
136: #endif
137:
138: /* Possible mode for `I_SWROPT'.  */
139: #define SNDZERO        0x001    /* Send a zero-length message downstream when a
140:                    `write' of 0 bytes occurs.  */
141: #ifdef __USE_GNU
142: # define SNDPIPE    0x002    /* Send SIGPIPE on write and putmsg if
143:                    sd_werror is set.  */
144: #endif
145:
146: /* Arguments for `I_ATMARK'.  */
147: #define ANYMARK        0x01    /* Check if the message is marked.  */ [#m 7
148: #define LASTMARK    0x02    /* Check if the message is the last one marked
149:                    on the queue.  */
150:
151: /* Argument for `I_UNLINK'.  */
 ...
167:
168: /* Structure used for the I_FLUSHBAND ioctl on streams.  */
169: struct bandinfo
170:   {
```

```
301: };
302:
303: #else
304:
 ...
324:
325: /*
326:  * For I_LIST ioctl.
327:  */
328: struct str_mlist {
329:     char l_name[FMNAMESZ+1];
330: };
331:
332: struct str_list {
333:     int sl_nmods;
334:     struct str_mlist *sl_modlist;
335: };
336:
337: /*
338:  * For I_FLUSHBAND ioctl.  Describes the priority
339:  * band for which the operation applies.
340:  */
341: struct bandinfo {
342:     unsigned char    bi_pri;
343:     int        bi_flag;
344: };
345:
346: #if !defined(_KERNEL)
347: #if defined(__STDC__)
348: int isastream(int);
349: #else
350: int isastream();
351: #endif
352: #endif /* !defined(_KERNEL) */
353:
354: #if defined(__cplusplus)
355:         }
356: #endif
357: #endif    /* _IO_STROPTS_H */
```

```
171:     unsigned char bi_pri;
172:     int bi_flag;
173:   };
174:
175: struct strbuf
176:   {
177:     int maxlen;        /* Maximum buffer length.  */
178:     int len;        /* Length of data.  */
179:     char *buf;        /* Pointer to buffer.  */
180:   };
181:
182: struct strpeek
183:   {
184:     struct strbuf ctlbuf;
185:     struct strbuf databuf;
186:     t_uscalar_t flags;            /* UnixWare/Solaris compatibility.  */
187:   };
188:
189: struct strfdinsert
190:   {
191:     struct strbuf ctlbuf;
192:     struct strbuf databuf;
193:     t_uscalar_t flags;            /* UnixWare/Solaris compatibility.  */
194:     int fildes;
195:     int offset;
196:   };
197:
198: struct strioctl
199:   {
200:     int ic_cmd;
201:     int ic_timout;
202:     int ic_len;
203:     char *ic_dp;
204:   };
205:
206: struct strrecvfd
207:   {
208:     int fd;
209:     uid_t uid;
210:     gid_t gid;
211:     char __fill[8];            /* UnixWare/Solaris compatibility */
212:   };
213:
214:
215: struct str_mlist
216:   {
217:     char l_name[FMNAMESZ + 1];
218:   };
219:
220: struct str_list
221:   {
222:     int sl_nmods;
223:     struct str_mlist *sl_modlist;
224:   };
225:
226: #endif /* bits/stropts.h */
```


(For readability, not all matches are indicated.)