```c
#include "globals.h"
#include "api.h"

#define ISREADY                                 \
  if (!xbee_ready) {                            \
    printf("XBee: Run xbee_setup() first!...\n"); \
    exit(1);                                    \
  }

/* ready flag.
   needs to be set to -1 so that the listen thread can begin.
   then 1 so that functions can be used (after setup of course...) */
int xbee_ready = 0;

/* ######################################################################## */
/* ### Memory Handling ################################################## */
/* ######################################################################## */

/* malloc wrapper function */
void *Xmalloc(size_t size) {
  void *t;
  t = malloc(size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:malloc()");
    exit(1);
  }
  return t;
}

/* calloc wrapper function */
void *Xcalloc(size_t size) {
  void *t;
  t = calloc(1, size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:calloc()");
    exit(1);
  }
  return t;
}

/* realloc wrapper function */
void *Xrealloc(void *ptr, size_t size) {
  void *t;
  t = realloc(ptr,size);
  if (!t) {
    /* uhoh... thats pretty bad... */
    perror("xbee:realloc()");
    exit(1);
  }
  return t;
}

/* free wrapper function (uses the Xfree macro and sets the pointer to NULL after freeing it) */
void Xfree2(void **ptr) {
  free(*ptr);
  *ptr = NULL;
}

/* ######################################################################## */
/* ### XBee Functions ################################################### */
/* ######################################################################## */

/* ########################################################################
   xbee_setup
   opens xbee serial port & creates xbee read thread
   the xbee must be configured for API mode 2
   THIS MUST BE CALLED BEFORE ANY OTHER XBEE FUNCTION */
int xbee_setup(char *path, int baudrate) {
  t_info info;
  struct flock fl;
  struct termios tc;
  speed_t chosenbaud;

  /* select the baud rate */
  switch (baudrate) {
    case 1200:  chosenbaud = B1200;   break;
    case 2400:  chosenbaud = B2400;   break;
    case 4800:  chosenbaud = B4800;   break;
    case 9600:  chosenbaud = B9600;   break;
    case 19200: chosenbaud = B19200;  break;
    case 38400: chosenbaud = B38400;  break;
    case 57600: chosenbaud = B57600;  break;
    case 115200:chosenbaud = B115200; break;
```

```
  86:     default:
  87:       printf("XBee: Unknown or incompatiable baud rate specified... (%d)\n",baudrate);
  88:       return -1;
  89:   };
  90:
  91:   /* setup the connection mutex */
  92:   xbee.conlist = NULL;
  93:   if (pthread_mutex_init(&xbee.conmutex,NULL)) {
  94:     perror("xbee_setup():pthread_mutex_init(conmutex)");
  95:     return -1;
  96:   }
  97:
  98:   /* setup the packet mutex */
  99:   xbee.pktlist = NULL;
 100:   if (pthread_mutex_init(&xbee.pktmutex,NULL)) {
 101:     perror("xbee_setup():pthread_mutex_init(pktmutex)");
 102:     return -1;
 103:   }
 104:
 105:   /* setup the send mutex */
 106:   if (pthread_mutex_init(&xbee.sendmutex,NULL)) {
 107:     perror("xbee_setup():pthread_mutex_init(sendmutex)");
 108:     return -1;
 109:   }
 110:
 111:   /* take a copy of the XBee device path */
 112:   if ((xbee.path = malloc(sizeof(char) * (strlen(path) + 1))) == NULL) {
 113:     perror("xbee_setup():malloc(path)");
 114:     return -1;
 115:   }
 116:   strcpy(xbee.path,path);
 117:
 118:   /* open the serial port as a file descriptor */
 119:   if ((xbee.ttyfd = open(path,O_RDWR | O_NOCTTY | O_NONBLOCK)) == -1) {
 120:     perror("xbee_setup():open()");
 121:     Xfree(xbee.path);
 122:     xbee.ttyfd = -1;
 123:     xbee.tty = NULL;
 124:     return -1;
 125:   }
 126:
 127:   /* lock the file */
 128:   fl.l_type = F_WRLCK | F_RDLCK;
 129:   fl.l_whence = SEEK_SET;
 130:   fl.l_start = 0;
 131:   fl.l_len = 0;
 132:   fl.l_pid = getpid();
 133:   if (fcntl(xbee.ttyfd, F_SETLK, &fl) == -1) {
 134:     perror("xbee_setup():fcntl()");
 135:     Xfree(xbee.path);
 136:     close(xbee.ttyfd);
 137:     xbee.ttyfd = -1;
 138:     xbee.tty = NULL;
 139:     return -1;
 140:   }
 141:
 142:
 143:   /* setup the baud rate and other io attributes */
 144:   tcgetattr(xbee.ttyfd, &tc);
 145:   cfsetispeed(&tc, chosenbaud);       /* set input baud rate to 57600 */
 146:   cfsetospeed(&tc, chosenbaud);       /* set output baud rate to 57600 */
 147:   /* input flags */
 148:   tc.c_iflag |= IGNBRK;             /* enable ignoring break */
 149:   tc.c_iflag &= ~(IGNPAR | PARMRK);/* disable parity checks */
 150:   tc.c_iflag &= ~INPCK;             /* disable parity checking */
 151:   tc.c_iflag &= ~ISTRIP;            /* disable stripping 8th bit */
 152:   tc.c_iflag &= ~(INLCR | ICRNL);  /* disable translating NL <-> CR */
 153:   tc.c_iflag &= ~IGNCR;             /* disable ignoring CR */
 154:   tc.c_iflag &= ~(IXON | IXOFF);   /* disable XON/XOFF flow control */
 155:   /* output flags */
 156:   tc.c_oflag &= ~OPOST;             /* disable output processing */
 157:   tc.c_oflag &= ~(ONLCR | OCRNL);  /* disable translating NL <-> CR */
 158:   tc.c_oflag &= ~OFILL;             /* disable fill characters */
 159:   /* control flags */
 160:   tc.c_cflag |= CREAD;              /* enable reciever */
 161:   tc.c_cflag &= ~PARENB;            /* disable parity */
 162:   tc.c_cflag &= ~CSTOPB;            /* disable 2 stop bits */
 163:   tc.c_cflag &= ~CSIZE;             /* remove size flag... */
 164:   tc.c_cflag |= CS8;                /* ...enable 8 bit characters */
 165:   tc.c_cflag |= HUPCL;              /* enable lower control lines on close - hang up */
 166:   /* local flags */
 167:   tc.c_lflag &= ~ISIG;              /* disable generating signals */
 168:   tc.c_lflag &= ~ICANON;            /* disable canonical mode - line by line */
 169:   tc.c_lflag &= ~ECHO;              /* disable echoing characters */
 170:   tc.c_lflag &= ~NOFLSH;            /* disable flushing on SIGINT */
```

```
171:   tc.c_lflag &= ~IEXTEN;           /* disable input processing */
172:   tcsetattr(xbee.ttyfd, TCSANOW, &tc);
173:
174:   /* open the serial port as a FILE* */
175:   if ((xbee.tty = fdopen(xbee.ttyfd,"r+")) == NULL) {
176:     perror("xbee_setup():fdopen()");
177:     Xfree(xbee.path);
178:     close(xbee.ttyfd);
179:     xbee.ttyfd = -1;
180:     xbee.tty = NULL;
181:     return -1;
182:   }
183:
184:   /* flush the serial port */
185:   fflush(xbee.tty);
186:
187:   /* allow the listen thread to start */
188:   xbee_ready = -1;
189:
190:   /* can start xbee_listen thread now */
191:   if (pthread_create(&xbee.listent,NULL,(void *(*)(void *))xbee_listen,(void *)&info) != 0) {
192:     perror("xbee_setup():pthread_create()");
193:     Xfree(xbee.path);
194:     fclose(xbee.tty);
195:     close(xbee.ttyfd);
196:     xbee.ttyfd = -1;
197:     xbee.tty = NULL;
198:     return -1;
199:   }
200:
201:   /* allow other functions to be used! */
202:   xbee_ready = 1;
203:
204:   /* make a txStatus connection */
205:   xbee.con_txStatus = xbee_newcon('*',xbee_txStatus);
206:
207:   return 0;
208: }
209:
210: /* ######################################################################
211:    xbee_con
212:    produces a connection to the specified device and frameID
213:    if a connection had already been made, then this connection will be returned */
214: xbee_con *xbee_newcon(unsigned char frameID, xbee_types type, ...) {
215:   xbee_con *con, *ocon;
216:   unsigned char tAddr[8];
217:   va_list ap;
218:   int t;
219: #ifdef DEBUG
220:   int i;
221: #endif
222:
223:   ISREADY;
224:
225:   if (!type || type == xbee_unknown) type = xbee_localAT; /* default to local AT */
226:   else if (type == xbee_remoteAT) type = xbee_64bitRemoteAT; /* if remote AT, default to 64bit */
227:
228:   va_start(ap,type);
229:   /* if: 64 bit address expected (2 ints) */
230:   if ((type == xbee_64bitRemoteAT) ||
231:       (type == xbee_64bitData) ||
232:       (type == xbee_64bitIO)) {
233:     t = va_arg(ap, int);
234:     tAddr[0] = (t >> 24) & 0xFF;
235:     tAddr[1] = (t >> 16) & 0xFF;
236:     tAddr[2] = (t >>  8) & 0xFF;
237:     tAddr[3] = (t      ) & 0xFF;
238:     t = va_arg(ap, int);
239:     tAddr[4] = (t >> 24) & 0xFF;
240:     tAddr[5] = (t >> 16) & 0xFF;
241:     tAddr[6] = (t >>  8) & 0xFF;
242:     tAddr[7] = (t      ) & 0xFF;
243:
244:   /* if: 16 bit address expected (1 int) */
245:   } else if ((type == xbee_16bitRemoteAT) ||
246:              (type == xbee_16bitData) ||
247:              (type == xbee_16bitIO)) {
248:     t = va_arg(ap, int);
249:     tAddr[0] = (t >>  8) & 0xFF;
250:     tAddr[1] = (t      ) & 0xFF;
251:     tAddr[2] = 0;
252:     tAddr[3] = 0;
253:     tAddr[4] = 0;
254:     tAddr[5] = 0;
255:     tAddr[6] = 0;
```

```
256:     tAddr[7] = 0;
257:
258:   /* otherwise clear the address */
259:   } else {
260:     memset(tAddr,0,8);
261:   }
262:   va_end(ap);
263:
264:   /* lock the connection mutex */
265:   pthread_mutex_lock(&xbee.conmutex);
266:
267:   /* are there any connections? */
268:   if (xbee.conlist) {
269:     con = xbee.conlist;
270:     while (con) {
271:       /* if: after a modemStatus, and the types match! */
272:       if ((type == xbee_modemStatus) &&
273:           (con->type == type)) {
274:         pthread_mutex_unlock(&xbee.conmutex);
275:         return con;
276:
277:       /* if: after a txStatus and frameIDs match! */
278:       } else if ((type == xbee_txStatus) &&
279:                  (con->type == type) &&
280:                  (frameID == con->frameID)) {
281:         pthread_mutex_unlock(&xbee.conmutex);
282:         return con;
283:
284:       /* if: after a localAT, and the frameIDs match! */
285:       } else if ((type == xbee_localAT) &&
286:                  (con->type == type) &&
287:                  (frameID == con->frameID)) {
288:         pthread_mutex_unlock(&xbee.conmutex);
289:         return con;
290:
291:       /* if: connection types match, the frameIDs match, and the addresses match! */
292:       } else if ((type == con->type) &&
293:                  (frameID == con->frameID) &&
294:                  (!memcmp(tAddr,con->tAddr,8))) {
295:         pthread_mutex_unlock(&xbee.conmutex);
296:         return con;
297:       }
298:
299:       /* if there are more, move along, dont want to loose that last item! */
300:       if (con->next == NULL) break;
301:       con = con->next;
302:     }
303:
304:     /* keep hold of the last connection... we will need to link it up later */
305:     ocon = con;
306:   }
307:
308:   /* create a new connection and set its attributes */
309:   con = Xcalloc(sizeof(xbee_con));
310:   con->type = type;
311:   /* is it a 64bit connection? */
312:   if ((type == xbee_64bitRemoteAT) ||
313:       (type == xbee_64bitData) ||
314:       (type == xbee_64bitIO)) {
315:     con->tAddr64 = TRUE;
316:   }
317:   con->atQueue = 0; /* queue AT commands? */
318:   con->txDisableACK = 0; /* disable ACKs? */
319:   con->txBroadcast = 0; /* broadcast? */
320:   con->frameID = frameID;
321:   memcpy(con->tAddr,tAddr,8); /* copy in the remote address */
322:
323: #ifdef DEBUG
324:   switch(type) {
325:     case xbee_localAT:
326:       printf("XBee: New local AT connection!\n");
327:       break;
328:     case xbee_16bitRemoteAT:
329:     case xbee_64bitRemoteAT:
330:       printf("XBee: New %d-bit remote AT connection! (to: ",(con->tAddr64?64:16));
331:       for (i=0;i<(con->tAddr64?8:2);i++) {
332:         printf((i?":%02X":"%02X"),tAddr[i]);
333:       }
334:       printf(")\n");
335:       break;
336:     case xbee_16bitData:
337:     case xbee_64bitData:
338:       printf("XBee: New %d-bit data connection! (to: ",(con->tAddr64?64:16));
339:       for (i=0;i<(con->tAddr64?8:2);i++) {
340:         printf((i?":%02X":"%02X"),tAddr[i]);
```

```
341:       }
342:       printf(")\n");
343:       break;
344:     case xbee_16bitIO:
345:     case xbee_64bitIO:
346:       printf("XBee: New %d-bit IO connection! (to: ",(con->tAddr64?64:16));
347:       for (i=0;i<(con->tAddr64?8:2);i++) {
348:         printf((i?":%02X":"%02X"),tAddr[i]);
349:       }
350:       printf(")\n");
351:       break;
352:     case xbee_txStatus:
353:       printf("XBee: New Tx status connection!\n");
354:       break;
355:     case xbee_modemStatus:
356:       printf("XBee: New modem status connection!\n");
357:       break;
358:     case xbee_unknown:
359:     default:
360:       printf("XBee: New unknown connection!\n");
361:   }
362: #endif
363:
364:   /* make it the last in the list */
365:   con->next = NULL;
366:   /* add it to the list */
367:   if (xbee.conlist) {
368:     ocon->next = con;
369:   } else {
370:     xbee.conlist = con;
371:   }
372:
373:   /* unlock the mutex */
374:   pthread_mutex_unlock(&xbee.conmutex);
375:   return con;
376: }
377:
378: /* ################################################################
379:    xbee_senddata
380:    send the specified data to the provided connection */
381: xbee_pkt *xbee_senddata(xbee_con *con, char *format, ...) {
382:   xbee_pkt *p;
383:   va_list ap;
384:
385:   ISREADY;
386:
387:   /* xbee_vsenddata() wants a va_list... */
388:   va_start(ap, format);
389:   /* hand it over :) */
390:   p = xbee_vsenddata(con,format,ap);
391:   va_end(ap);
392:   return p;
393: }
394:
395: xbee_pkt *xbee_vsenddata(xbee_con *con, char *format, va_list ap) {
396:   t_data *pkt;
397:   int i, length;
398:   unsigned char buf[128]; /* max payload is 100 bytes... plus a bit for the headers etc... */
399:   unsigned char data[128]; /* ditto */
400:   xbee_pkt *p = NULL; /* response packet */
401:   int to = 10; /* resonse timeout */
402:
403:   ISREADY;
404:
405:   if (!con) return (void *)-1;
406:   if (con->type == xbee_unknown) return (void *)-1;
407:
408:   /* make up the data and keep the length, its possible there are nulls in there */
409:   length = vsnprintf((char *)data,128,format,ap);
410:
411: #ifdef DEBUG
412:   printf("XBee: --== TX Packet ============--\n");
413:   printf("XBee: Length: %d\n",length);
414:   for (i=0;i<length;i++) {
415:     printf("XBee: %3d | 0x%02X ",i,data[i]);
416:     if ((data[i] > 32) && (data[i] < 127)) printf("'%c'\n",data[i]); else printf(" _\n");
417:   }
418: #endif
419:
420:   /* ######################################### */
421:   /* if: local AT */
422:   if (con->type == xbee_localAT) {
423:     /* AT commands are 2 chars long (plus optional parameter) */
424:     if (length < 2) return (void *)-1;
425:
```

```
426:     /* use the command? */
427:     buf[0] = ((!con->atQueue)?0x08:0x09);
428:     buf[1] = con->frameID;
429:
430:     /* copy in the data */
431:     for (i=0;i<length;i++) {
432:       buf[i+2] = data[i];
433:     }
434:
435:     /* setup the packet */
436:     pkt = xbee_make_pkt(buf,i+2);
437:     /* send it on */
438:     xbee_send_pkt(pkt);
439:
440:     /* wait for a response packet */
441:     for (; p == NULL && to > 0; to--) {
442:       usleep(25400); /* tuned so that hopefully the first time round will catch the response */
443:       p = xbee_getpacket(con);
444:     }
445:
446:     /* if: no txStatus packet was recieved */
447:     if (to == 0) {
448: #ifdef DEBUG
449:       printf("XBee: No AT status recieved before timeout\n");
450: #endif
451:       return NULL;
452:     }
453:
454: #ifdef DEBUG
455:     switch (p->status) {
456:     case 0x00: printf("XBee: AT Status: OK!\n");               break;
457:     case 0x01: printf("XBee: AT Status: Error\n");            break;
458:     case 0x02: printf("XBee: AT Status: Invalid Command\n");  break;
459:     case 0x03: printf("XBee: AT Status: Invalid Parameter\n"); break;
460:     }
461: #endif
462:     return p;
463:   /* ######################################### */
464:   /* if: remote AT */
465:   } else if ((con->type == xbee_16bitRemoteAT) ||
466:              (con->type == xbee_64bitRemoteAT)) {
467:     if (length < 2) return (void *)-1; /* at commands are 2 chars long (plus optional parameter) */
468:     buf[0] = 0x17;
469:     buf[1] = con->frameID;
470:
471:     /* copy in the relevant address */
472:     if (con->tAddr64) {
473:       memcpy(&buf[2],con->tAddr,8);
474:       buf[10] = 0xFF;
475:       buf[11] = 0xFE;
476:     } else {
477:       memset(&buf[2],0,8);
478:       memcpy(&buf[10],con->tAddr,2);
479:     }
480:     /* queue the command? */
481:     buf[12] = ((!con->atQueue)?0x02:0x00);
482:
483:     /* copy in the data */
484:     for (i=0;i<length;i++) {
485:       buf[i+13] = data[i];
486:     }
487:
488:     /* setup the packet */
489:     pkt = xbee_make_pkt(buf,i+13);
490:     /* send it on */
491:     xbee_send_pkt(pkt);
492:
493:     /* wait for a response packet */
494:     for (; p == NULL && to > 0; to--) {
495:       usleep(25400); /* tuned so that hopefully the first time round will catch the response */
496:       p = xbee_getpacket(con);
497:     }
498:
499:     /* if: no txStatus packet was recieved */
500:     if (to == 0) {
501: #ifdef DEBUG
502:       printf("XBee: No AT status recieved before timeout\n");
503: #endif
504:       return NULL;
505:     }
506:
507: #ifdef DEBUG
508:     switch (p->status) {
509:     case 0x00: printf("XBee: AT Status: OK!\n");               break;
510:     case 0x01: printf("XBee: AT Status: Error\n");            break;
```

```
511:     case 0x02: printf("XBee: AT Status: Invalid Command\n");   break;
512:     case 0x03: printf("XBee: AT Status: Invalid Parameter\n"); break;
513:     case 0x04: printf("XBee: AT Status: No Response\n");       break;
514:     }
515: #endif
516:     return p;
517:   /* ######################################### */
518:   /* if: 16 or 64bit Data */
519:   } else if ((con->type == xbee_16bitData) ||
520:              (con->type == xbee_64bitData)) {
521:     int offset;
522:
523:     /* if: 16bit Data */
524:     if (con->type == xbee_16bitData) {
525:       buf[0] = 0x01;
526:       offset = 5;
527:       /* copy in the address */
528:       memcpy(&buf[2],con->tAddr,2);
529:
530:     /* if: 64bit Data */
531:     } else { /* 64bit Data */
532:       buf[0] = 0x00;
533:       offset = 11;
534:       /* copy in the address */
535:       memcpy(&buf[2],con->tAddr,8);
536:     }
537:
538:     /* copy frameID */
539:     buf[1] = con->frameID;
540:
541:     /* disable ack? broadcast? */
542:     buf[offset-1] = ((con->txDisableACK)?0x01:0x00) | ((con->txBroadcast)?0x04:0x00);
543:
544:     /* copy in the data */
545:     for (i=0;i<length;i++) {
546:       buf[i+offset] = data[i];
547:     }
548:
549:     /* setup the packet */
550:     pkt = xbee_make_pkt(buf,i+offset);
551:     /* send it on */
552:     xbee_send_pkt(pkt);
553:
554:     /* wait for a response packet */
555:     for (; p == NULL && to > 0; to--) {
556:       usleep(25400); /* tuned so that hopefully the first time round will catch the response */
557:       p = xbee_getpacket(xbee.con_txStatus);
558:     }
559:
560:     /* if: no txStatus packet was recieved */
561:     if (to == 0) {
562: #ifdef DEBUG
563:       printf("XBee: No txStatus recieved before timeout\n");
564: #endif
565:       return NULL;
566:     }
567:
568: #ifdef DEBUG
569:     switch (p->status) {
570:     case 0x00: printf("XBee: txStatus: Success!\n");    break;
571:     case 0x01: printf("XBee: txStatus: No ACK\n");      break;
572:     case 0x02: printf("XBee: txStatus: CCA Failure\n"); break;
573:     case 0x03: printf("XBee: txStatus: Purged\n");      break;
574:     }
575: #endif
576:     /* return the packet */
577:     return p;
578:   /* ######################################### */
579:   /* if: I/O */
580:   } else if ((con->type == xbee_64bitIO) ||
581:              (con->type == xbee_16bitIO)) {
582:     /* not currently implemented... is it even allowed? */
583:     printf("******* TODO ********\n");
584:   }
585:
586:   return (void *)-1;
587: }
588:
589: /* ####################################################################
590:    xbee_getpacket
591:    retrieves the next packet destined for the given connection
592:    once the packet has been retrieved, it is removed for the list! */
593: xbee_pkt *xbee_getpacket(xbee_con *con) {
594:   xbee_pkt *l, *p, *q;
595: #ifdef DEBUG
```

```
596:   int c;
597:   printf("XBee: --== Get Packet ==========--\n");
598: #endif
599:
600:   /* lock the packet mutex */
601:   pthread_mutex_lock(&xbee.pktmutex);
602:
603:   /* if: there are no packets */
604:   if ((p = xbee.pktlist) == NULL) {
605:     pthread_mutex_unlock(&xbee.pktmutex);
606: #ifdef DEBUG
607:     printf("XBee: No packets avaliable...\n");
608: #endif
609:     return NULL;
610:   }
611:
612:   l = NULL;
613:   q = NULL;
614:   /* get the first avaliable packet for this socket */
615:   do {
616:     /* if: the connection type matches the packet type OR
617:        the connection is 16/64bit remote AT, and the packet is a remote AT response */
618:     if ((p->type == con->type) || /* -- */
619:         ((p->type == xbee_remoteAT) && /* -- */
620:          ((con->type == xbee_16bitRemoteAT) ||
621:           (con->type == xbee_64bitRemoteAT)))) {
622:
623:       /* if: the packet is modem status OR
624:          the packet is tx status or AT data and the frame IDs match OR
625:          the addresses match */
626:       if ((p->type == xbee_modemStatus) ||
627:           (((p->type == xbee_txStatus) ||
628:             (p->type == xbee_localAT) ||
629:             (p->type == xbee_remoteAT)) &&
630:            (con->frameID == p->frameID)) ||
631:           (!memcmp(con->tAddr,p->Addr64,8))) {
632:         q = p;
633:         break;
634:       }
635:     }
636:
637:     /* move on */
638:     l = p;
639:     p = p->next;
640:   } while (p);
641:
642:   /* if: no packet was found */
643:   if (!q) {
644:     pthread_mutex_unlock(&xbee.pktmutex);
645: #ifdef DEBUG
646:     printf("XBee: No packets avaliable (for connection)...\n");
647: #endif
648:     return NULL;
649:   }
650:
651:   /* if it was the first packet */
652:   if (!l) {
653:     /* move the chain along */
654:     xbee.pktlist = p->next;
655:   } else {
656:     /* otherwise relink the list */
657:     l->next = p->next;
658:   }
659:
660: #ifdef DEBUG
661:   printf("XBee: Got a packet\n");
662:   for (p = xbee.pktlist,c = 0;p;c++,p = p->next);
663:   printf("XBee: Packets left: %d\n",c);
664: #endif
665:
666:   /* unlock the packet mutex */
667:   pthread_mutex_unlock(&xbee.pktmutex);
668:
669:   /* and return the packet (must be freed by caller!) */
670:   return q;
671: }
672:
673: /* ###################################################################
674:    xbee_listen - INTERNAL
675:    the xbee xbee_listen thread
676:    reads data from the xbee and puts it into a linked list to keep the xbee buffers free */
677: void xbee_listen(t_info *info) {
678:   unsigned char c, t, d[128];
679:   unsigned int l, i, chksum, o;
680: #ifdef DEBUG
```

```
681:   int j;
682: #endif
683:   xbee_pkt *p, *q, *po;
684:
685:   /* just falls out if the proper 'go-ahead' isn't given */
686:   if (xbee_ready != -1) return;
687:
688:   /* do this forever :) */
689:   while(1) {
690:     /* wait for a valid start byte */
691:     if (xbee_getRawByte() != 0x7E) continue;
692:
693: #ifdef DEBUG
694:     printf("XBee: --== RX Packet ==========--\nXBee: Got a packet!...\n");
695: #endif
696:
697:     /* get the length */
698:     l = xbee_getByte() << 8;
699:     l += xbee_getByte();
700:
701:     /* check it is a valid length... */
702:     if (!l) {
703: #ifdef DEBUG
704:       printf("XBee: Recived zero length packet!\n");
705: #endif
706:       continue;
707:     }
708:     if (l > 100) {
709: #ifdef DEBUG
710:       printf("XBee: Recived oversized packet! Length: %d\n",l - 1);
711: #endif
712:       continue;
713:     }
714:
715: #ifdef DEBUG
716:     printf("XBee: Length: %d\n",l - 1);
717: #endif
718:
719:     /* get the packet type */
720:     t = xbee_getByte();
721:
722:     /* start the checksum */
723:     chksum = t;
724:
725:     /* suck in all the data */
726:     for (i = 0; l > 1 && i < 128; l--, i++) {
727:       /* get an unescaped byte */
728:       c = xbee_getByte();
729:       d[i] = c;
730:       chksum += c;
731: #ifdef DEBUG
732:       printf("XBee: %3d | 0x%02X | ",i,c);
733:       if ((c > 32) && (c < 127)) printf("'%c'\n",c); else printf(" _\n");
734: #endif
735:     }
736:     i--; /* it went up too many times!... */
737:
738:     /* add the checksum */
739:     chksum += xbee_getByte();
740:
741:     /* check if the whole packet was recieved, or something else occured... unlikely... */
742:     if (l>1) {
743: #ifdef DEBUG
744:       printf("XBee: Didn't get whole packet... :(\n");
745: #endif
746:       continue;
747:     }
748:
749:     /* check the checksum */
750:     if ((chksum & 0xFF) != 0xFF) {
751: #ifdef DEBUG
752:       printf("XBee: Invalid Checksum: 0x%02X\n",chksum);
753: #endif
754:       continue;
755:     }
756:
757:     /* make a new packet */
758:     po = p = Xcalloc(sizeof(xbee_pkt));
759:     q = NULL;
760:     p->datalen = 0;
761:
762:     /* ########################################### */
763:     /* if: modem status */
764:     if (t == 0x8A) {
765: #ifdef DEBUG
```

```
766:       printf("XBee: Packet type: Modem Status (0x8A)\n");
767:       printf("XBee: ");
768:       switch (d[0]) {
769:       case 0x00: printf("Hardware reset"); break;
770:       case 0x01: printf("Watchdog timer reset"); break;
771:       case 0x02: printf("Associated"); break;
772:       case 0x03: printf("Disassociated"); break;
773:       case 0x04: printf("Synchronization lost"); break;
774:       case 0x05: printf("Coordinator realignment"); break;
775:       case 0x06: printf("Coordinator started"); break;
776:       }
777:       printf("...\n");
778: #endif
779:       p->type = xbee_modemStatus;
780:
781:       p->sAddr64 = FALSE;
782:       p->dataPkt = FALSE;
783:       p->txStatusPkt = FALSE;
784:       p->modemStatusPkt = TRUE;
785:       p->remoteATPkt = FALSE;
786:       p->IOPkt = FALSE;
787:
788:       /* modem status can only ever give 1 'data' byte */
789:       p->datalen = 1;
790:       p->data[0] = d[0];
791:
792:     /* ######################################### */
793:     /* if: local AT response */
794:     } else if (t == 0x88) {
795: #ifdef DEBUG
796:       printf("XBee: Packet type: Local AT Response (0x88)\n");
797:       printf("XBee: FrameID: 0x%02X\n",d[0]);
798:       printf("XBee: AT Command: %c%c\n",d[1],d[2]);
799:       if (d[3] == 0) printf("XBee: Status: OK\n");
800:       else if (d[3] == 1) printf("XBee: Status: Error\n");
801:       else if (d[3] == 2) printf("XBee: Status: Invalid Command\n");
802:       else if (d[3] == 3) printf("XBee: Status: Invalid Parameter\n");
803: #endif
804:       p->type = xbee_localAT;
805:
806:       p->sAddr64 = FALSE;
807:       p->dataPkt = FALSE;
808:       p->txStatusPkt = FALSE;
809:       p->modemStatusPkt = FALSE;
810:       p->remoteATPkt = FALSE;
811:       p->IOPkt = FALSE;
812:
813:       p->frameID = d[0];
814:       p->atCmd[0] = d[1];
815:       p->atCmd[1] = d[2];
816:
817:       p->status = d[3];
818:
819:       /* copy in the data */
820:       p->datalen = i-3;
821:       for (;i>3;i--) p->data[i-4] = d[i];
822:
823:     /* ######################################### */
824:     /* if: remote AT response */
825:     } else if (t == 0x97) {
826: #ifdef DEBUG
827:       printf("XBee: Packet type: Remote AT Response (0x97)\n");
828:       printf("XBee: FrameID: 0x%02X\n",d[0]);
829:       printf("XBee: 64-bit Address: ");
830:       for (j=0;j<8;j++) {
831:         printf((j?":%02X":"%02X"),d[1+j]);
832:       }
833:       printf("\n");
834:       printf("XBee: 16-bit Address: ");
835:       for (j=0;j<2;j++) {
836:         printf((j?":%02X":"%02X"),d[9+j]);
837:       }
838:       printf("\n");
839:       printf("XBee: AT Command: %c%c\n",d[11],d[12]);
840:       if (d[13] == 0) printf("XBee: Status: OK\n");
841:       else if (d[13] == 1) printf("XBee: Status: Error\n");
842:       else if (d[13] == 2) printf("XBee: Status: Invalid Command\n");
843:       else if (d[13] == 3) printf("XBee: Status: Invalid Parameter\n");
844:       else if (d[13] == 4) printf("XBee: Status: No Response\n");
845: #endif
846:       p->type = xbee_remoteAT;
847:
848:       p->sAddr64 = FALSE;
849:       p->dataPkt = FALSE;
850:       p->txStatusPkt = FALSE;
```

```
851:       p->modemStatusPkt = FALSE;
852:       p->remoteATPkt = TRUE;
853:       p->IOPkt = FALSE;
854:
855:       p->frameID = d[0];
856:
857:       p->Addr64[0] = d[1];
858:       p->Addr64[1] = d[2];
859:       p->Addr64[2] = d[3];
860:       p->Addr64[3] = d[4];
861:       p->Addr64[4] = d[5];
862:       p->Addr64[5] = d[6];
863:       p->Addr64[6] = d[7];
864:       p->Addr64[7] = d[8];
865:
866:       p->Addr16[0] = d[9];
867:       p->Addr16[1] = d[10];
868:
869:       p->atCmd[0] = d[11];
870:       p->atCmd[1] = d[12];
871:
872:       p->status = d[13];
873:
874:       /* copy in the data */
875:       p->datalen = i-13;
876:       for (;i>13;i--) p->data[i-14] = d[i];
877:
878:     /* ######################################### */
879:     /* if: TX status */
880:     } else if (t == 0x89) {
881: #ifdef DEBUG
882:       printf("XBee: Packet type: TX Status Report (0x89)\n");
883:       printf("XBee: FrameID: 0x%02X\n",d[0]);
884:       if (d[1] == 0) printf("XBee: Status: Success\n");
885:       else if (d[1] == 1) printf("XBee: Status: No ACK\n");
886:       else if (d[1] == 2) printf("XBee: Status: CCA Failure\n");
887:       else if (d[1] == 3) printf("XBee: Status: Purged\n");
888: #endif
889:       p->type = xbee_txStatus;
890:
891:       p->sAddr64 = FALSE;
892:       p->dataPkt = FALSE;
893:       p->txStatusPkt = TRUE;
894:       p->modemStatusPkt = FALSE;
895:       p->remoteATPkt = FALSE;
896:       p->IOPkt = FALSE;
897:
898:       p->frameID = d[0];
899:
900:       p->status = d[1];
901:
902:       /* never returns data */
903:         p->datalen = 0;
904:
905:     /* ######################################### */
906:     /* if: 64bit address recieve */
907:     } else if (t == 0x80) {
908: #ifdef DEBUG
909:       printf("XBee: Packet type: 64-bit RX Data (0x80)\n");
910:       printf("XBee: 64-bit Address: ");
911:       for (j=0;j<8;j++) {
912:         printf((j?":%02X":"%02X"),d[j]);
913:       }
914:       printf("\n");
915:       printf("XBee: RSSI: -%ddB\n",d[8]);
916:       if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
917:       if (d[9] & 0x03) printf("XBee: Options: PAN Broadcast\n");
918: #endif
919:       p->type = xbee_64bitData;
920:
921:       p->sAddr64 = TRUE;
922:       p->dataPkt = TRUE;
923:       p->txStatusPkt = FALSE;
924:       p->modemStatusPkt = FALSE;
925:       p->remoteATPkt = FALSE;
926:       p->IOPkt = FALSE;
927:
928:       p->Addr64[0] = d[0];
929:       p->Addr64[1] = d[1];
930:       p->Addr64[2] = d[2];
931:       p->Addr64[3] = d[3];
932:       p->Addr64[4] = d[4];
933:       p->Addr64[5] = d[5];
934:       p->Addr64[6] = d[6];
935:       p->Addr64[7] = d[7];
```

```
936:
937:       /* save the RSSI / signal strength
938:          this can be used with printf as:
939:          printf("-%ddB\n",p->RSSI); */
940:       p->RSSI = d[8];
941:
942:       p->status = d[9];
943:
944:       /* copy in the data */
945:       p->datalen = i-9;
946:       for (;i>9;i--) p->data[i-10] = d[i];
947:
948:     /* ######################################### */
949:     /* if: 16bit address recieve */
950:     } else if (t == 0x81) {
951: #ifdef DEBUG
952:       printf("XBee: Packet type: 16-bit RX Data (0x81)\n");
953:       printf("XBee: 16-bit Address: ");
954:       for (j=0;j<2;j++) {
955:         printf((j?":%02X":"%02X"),d[j]);
956:       }
957:       printf("\n");
958:       printf("XBee: RSSI: -%ddB\n",d[2]);
959:       if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
960:       if (d[3] & 0x03) printf("XBee: Options: PAN Broadcast\n");
961: #endif
962:       p->type = xbee_16bitData;
963:
964:       p->sAddr64 = FALSE;
965:       p->dataPkt = TRUE;
966:       p->txStatusPkt = FALSE;
967:       p->modemStatusPkt = FALSE;
968:       p->remoteATPkt = FALSE;
969:       p->IOPkt = FALSE;
970:
971:       p->Addr16[0] = d[0];
972:       p->Addr16[1] = d[1];
973:
974:       /* save the RSSI / signal strength
975:          this can be used with printf as:
976:          printf("-%ddB\n",p->RSSI); */
977:       p->RSSI = d[2];
978:
979:       p->status = d[3];
980:
981:       /* copy in the data */
982:       p->datalen = i-3;
983:       for (;i>3;i--) p->data[i-4] = d[i];
984:
985:     /* ######################################### */
986:     /* if: 64bit I/O recieve */
987:     } else if (t == 0x82) {
988: #ifdef DEBUG
989:       printf("XBee: Packet type: 64-bit RX I/O Data (0x82)\n");
990:       printf("XBee: 64-bit Address: ");
991:       for (j=0;j<8;j++) {
992:         printf((j?":%02X":"%02X"),d[j]);
993:       }
994:       printf("\n");
995:       printf("XBee: RSSI: -%ddB\n",d[8]);
996:       if (d[9] & 0x02) printf("XBee: Options: Address Broadcast\n");
997:       if (d[9] & 0x02) printf("XBee: Options: PAN Broadcast\n");
998:       printf("XBee: Samples: %d\n",d[10]);
999: #endif
1000:       i = 13;
1001:
1002:       /* each sample is split into its own packet here, for simplicity */
1003:       for (o=d[10];o>0;o--) {
1004: #ifdef DEBUG
1005:         printf("XBee: --- Sample %3d -------------\n",o-d[10]+1);
1006: #endif
1007:         /* if we arent still using the origional packet */
1008:         if (o<d[10]) {
1009:           /* make a new one and link it up! */
1010:           q = Xcalloc(sizeof(xbee_pkt));
1011:           p->next = q;
1012:           p = q;
1013:         }
1014:
1015:         /* never returns data */
1016:         p->datalen = 0;
1017:
1018:         p->type = xbee_64bitIO;
1019:
1020:         p->sAddr64 = TRUE;
```

```
1021:         p->dataPkt = FALSE;
1022:         p->txStatusPkt = FALSE;
1023:         p->modemStatusPkt = FALSE;
1024:         p->remoteATPkt = FALSE;
1025:         p->IOPkt = TRUE;
1026:
1027:         p->Addr64[0] = d[0];
1028:         p->Addr64[1] = d[1];
1029:         p->Addr64[2] = d[2];
1030:         p->Addr64[3] = d[3];
1031:         p->Addr64[4] = d[4];
1032:         p->Addr64[5] = d[5];
1033:         p->Addr64[6] = d[6];
1034:         p->Addr64[7] = d[7];
1035:
1036:         /* save the RSSI / signal strength
1037:            this can be used with printf as:
1038:            printf("-%ddB\n",p->RSSI); */
1039:         p->RSSI = d[8];
1040:
1041:         p->status = d[9];
1042:
1043:         /* copy in the I/O data mask */
1044:         p->IOmask = (((d[11]<<8) | d[12]) & 0x7FFF);
1045:
1046:         /* copy in the digital I/O data */
1047:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
1048:
1049:         /* advance over the digital data, if its there */
1050:         i += (((d[11]&0x01)||(d[12]))?2:0);
1051:
1052:         /* copy in the analog I/O data */
1053:         if (d[11]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1054:         if (d[11]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1055:         if (d[11]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1056:         if (d[11]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1057:         if (d[11]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1058:         if (d[11]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1059: #ifdef DEBUG
1060:         if (p->IOmask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
1061:         if (p->IOmask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
1062:         if (p->IOmask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
1063:         if (p->IOmask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
1064:         if (p->IOmask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
1065:         if (p->IOmask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
1066:         if (p->IOmask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
1067:         if (p->IOmask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
1068:         if (p->IOmask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
1069:         if (p->IOmask & 0x0200) printf("XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
1070:         if (p->IOmask & 0x0400) printf("XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
1071:         if (p->IOmask & 0x0800) printf("XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
1072:         if (p->IOmask & 0x1000) printf("XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
1073:         if (p->IOmask & 0x2000) printf("XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
1074:         if (p->IOmask & 0x4000) printf("XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
1075: #endif
1076:       }
1077: #ifdef DEBUG
1078:       printf("XBee: ---------------------------\n");
1079: #endif
1080:
1081:     /* ########################################### */
1082:     /* if: 16bit I/O recieve */
1083:     } else if (t == 0x83) {
1084: #ifdef DEBUG
1085:       printf("XBee: Packet type: 16-bit RX I/O Data (0x83)\n");
1086:       printf("XBee: 16-bit Address: ");
1087:       for (j=0;j<2;j++) {
1088:         printf((j?":%02X":"%02X"),d[j]);
1089:       }
1090:       printf("\n");
1091:       printf("XBee: RSSI: -%ddB\n",d[2]);
1092:       if (d[3] & 0x02) printf("XBee: Options: Address Broadcast\n");
1093:       if (d[3] & 0x02) printf("XBee: Options: PAN Broadcast\n");
1094:       printf("XBee: Samples: %d\n",d[4]);
1095: #endif
1096:
1097:       i = 7;
1098:
1099:       /* each sample is split into its own packet here, for simplicity */
1100:       for (o=d[4];o>0;o--) {
1101: #ifdef DEBUG
1102:         printf("XBee: --- Sample %3d ------------\n",o-d[4]+1);
1103: #endif
1104:         if (o<d[4]) {
1105:           q = Xcalloc(sizeof(xbee_pkt));
```

```
1106:           p->next = q;
1107:           p = q;
1108:         }
1109:
1110:         /* never returns data */
1111:         p->datalen = 0;
1112:
1113:         p->type = xbee_16bitIO;
1114:
1115:         p->sAddr64 = FALSE;
1116:         p->dataPkt = FALSE;
1117:         p->txStatusPkt = FALSE;
1118:         p->modemStatusPkt = FALSE;
1119:         p->remoteATPkt = FALSE;
1120:         p->IOPkt = TRUE;
1121:
1122:         p->Addr16[0] = d[0];
1123:         p->Addr16[1] = d[1];
1124:
1125:         /* save the RSSI / signal strength
1126:            this can be used with printf as:
1127:            printf("-%ddB\n",p->RSSI); */
1128:         p->RSSI = d[2];
1129:
1130:         p->status = d[3];
1131:
1132:         /* copy in the I/O data mask */
1133:         p->IOmask = (((d[5]<<8) | d[6]) & 0x7FFF);
1134:
1135:         /* copy in the digital I/O data */
1136:         p->IOdata = (((d[i]<<8) | d[i+1]) & 0x01FF);
1137:
1138:         /* advance over the digital data, if its there */
1139:         i += (((d[5]&0x01)||(d[6]))?2:0);
1140:
1141:         /* copy in the analog I/O data */
1142:         if (d[5]&0x02) {p->IOanalog[0] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1143:         if (d[5]&0x04) {p->IOanalog[1] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1144:         if (d[5]&0x08) {p->IOanalog[2] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1145:         if (d[5]&0x10) {p->IOanalog[3] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1146:         if (d[5]&0x20) {p->IOanalog[4] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1147:         if (d[5]&0x40) {p->IOanalog[5] = (((d[i]<<8) | d[i+1]) & 0x03FF);i+=2;}
1148: #ifdef DEBUG
1149:         if (p->IOmask & 0x0001) printf("XBee: Digital 0: %c\n",((p->IOdata & 0x0001)?'1':'0'));
1150:         if (p->IOmask & 0x0002) printf("XBee: Digital 1: %c\n",((p->IOdata & 0x0002)?'1':'0'));
1151:         if (p->IOmask & 0x0004) printf("XBee: Digital 2: %c\n",((p->IOdata & 0x0004)?'1':'0'));
1152:         if (p->IOmask & 0x0008) printf("XBee: Digital 3: %c\n",((p->IOdata & 0x0008)?'1':'0'));
1153:         if (p->IOmask & 0x0010) printf("XBee: Digital 4: %c\n",((p->IOdata & 0x0010)?'1':'0'));
1154:         if (p->IOmask & 0x0020) printf("XBee: Digital 5: %c\n",((p->IOdata & 0x0020)?'1':'0'));
1155:         if (p->IOmask & 0x0040) printf("XBee: Digital 6: %c\n",((p->IOdata & 0x0040)?'1':'0'));
1156:         if (p->IOmask & 0x0080) printf("XBee: Digital 7: %c\n",((p->IOdata & 0x0080)?'1':'0'));
1157:         if (p->IOmask & 0x0100) printf("XBee: Digital 8: %c\n",((p->IOdata & 0x0100)?'1':'0'));
1158:         if (p->IOmask & 0x0200) printf("XBee: Analog  0: %.2fv\n",(3.3/1023)*p->IOanalog[0]);
1159:         if (p->IOmask & 0x0400) printf("XBee: Analog  1: %.2fv\n",(3.3/1023)*p->IOanalog[1]);
1160:         if (p->IOmask & 0x0800) printf("XBee: Analog  2: %.2fv\n",(3.3/1023)*p->IOanalog[2]);
1161:         if (p->IOmask & 0x1000) printf("XBee: Analog  3: %.2fv\n",(3.3/1023)*p->IOanalog[3]);
1162:         if (p->IOmask & 0x2000) printf("XBee: Analog  4: %.2fv\n",(3.3/1023)*p->IOanalog[4]);
1163:         if (p->IOmask & 0x4000) printf("XBee: Analog  5: %.2fv\n",(3.3/1023)*p->IOanalog[5]);
1164: #endif
1165:       }
1166: #ifdef DEBUG
1167:       printf("XBee: ---------------------------\n");
1168: #endif
1169:
1170:     /* ########################################### */
1171:     /* if: Unknown */
1172:     } else {
1173: #ifdef DEBUG
1174:       printf("XBee: Packet type: Unknown (0x%02X)\n",t);
1175: #endif
1176:       p->type = xbee_unknown;
1177:     }
1178:     p->next = NULL;
1179:
1180:     /* lock the packet mutex, so we can savely add the packet to the list */
1181:     pthread_mutex_lock(&xbee.pktmutex);
1182:     i = 1;
1183:     /* if: the list is empty */
1184:     if (!xbee.pktlist) {
1185:       /* start the list! */
1186:       xbee.pktlist = po;
1187:     } else {
1188:       /* add the packet to the end */
1189:       q = xbee.pktlist;
1190:       while (q->next) {
```

```
1191:         q = q->next;
1192:         i++;
1193:       }
1194:       q->next = po;
1195:     }
1196:
1197: #ifdef DEBUG
1198:     while (q && q->next) {
1199:       q = q->next;
1200:       i++;
1201:     }
1202:     printf("XBee: --======================--\n");
1203:     printf("XBee: Packets: %d\n",i);
1204: #endif
1205:
1206:     po = p = q = NULL;
1207:
1208:     /* unlock the packet mutex */
1209:     pthread_mutex_unlock(&xbee.pktmutex);
1210:   }
1211: }
1212:
1213: /* ###################################################################
1214:    xbee_getByte - INTERNAL
1215:    waits for an escaped byte of data */
1216: unsigned char xbee_getByte(void) {
1217:   unsigned char c;
1218:
1219:   ISREADY;
1220:
1221:   /* take a byte */
1222:   c = xbee_getRawByte();
1223:   /* if its escaped, take another and un-escape */
1224:   if (c == 0x7D) c = xbee_getRawByte() ^ 0x20;
1225:
1226:   return (c & 0xFF);
1227: }
1228:
1229: /* ###################################################################
1230:    xbee_getRawByte - INTERNAL
1231:    waits for a raw byte of data */
1232: unsigned char xbee_getRawByte(void) {
1233:   unsigned char c;
1234:   fd_set fds;
1235:
1236:   ISREADY;
1237:
1238:   /* wait for a read to be possible */
1239:   FD_ZERO(&fds);
1240:   FD_SET(xbee.ttyfd,&fds);
1241:   if (select(xbee.ttyfd+1,&fds,NULL,NULL,NULL) == -1) {
1242:     perror("xbee:xbee_listen():xbee_getByte()");
1243:     exit(1);
1244:   }
1245:
1246:   /* read 1 character
1247:      the loop is just incase there actually isnt a byte there to be read... */
1248:   do {
1249:     if (read(xbee.ttyfd,&c,1) == 0) {
1250:       usleep(10);
1251:       continue;
1252:     }
1253:   } while (0);
1254:
1255:   return (c & 0xFF);
1256: }
1257:
1258: /* ###################################################################
1259:    xbee_send_pkt - INTERNAL
1260:    sends a complete packet of data */
1261: void xbee_send_pkt(t_data *pkt) {
1262:   ISREADY;
1263:
1264:
1265:   /* lock the send mutex */
1266:   pthread_mutex_lock(&xbee.sendmutex);
1267:
1268:   /* write and flush the data */
1269:   fwrite(pkt->data,pkt->length,1,xbee.tty);
1270:   fflush(xbee.tty);
1271:
1272:   /* unlock the mutex */
1273:   pthread_mutex_unlock(&xbee.sendmutex);
1274:
1275: #ifdef DEBUG
```

```
1276:   {
1277:     int i;
1278:     /* prints packet in hex byte-by-byte */
1279:     printf("XBee: TX Packet - ");
1280:     for (i=0;i<pkt->length;i++) {
1281:       printf("0x%02X ",pkt->data[i]);
1282:     }
1283:     printf("\n");
1284:   }
1285: #endif
1286:
1287:   /* free the packet */
1288:   Xfree(pkt);
1289: }
1290:
1291: /* ###################################################################
1292:    xbee_make_pkt - INTERNAL
1293:    adds delimiter field
1294:    calculates length and checksum
1295:    escapes bytes */
1296: t_data *xbee_make_pkt(unsigned char *data, int length) {
1297:   t_data *pkt;
1298:   unsigned int l, i, o, t, x, m;
1299:   char d = 0;
1300:
1301:   ISREADY;
1302:
1303:   /* check the data given isnt too long
1304:      100 bytes maximum payload + 12 bytes header information */
1305:   if (length > 100 + 12) return NULL;
1306:
1307:   /* calculate the length of the whole packet
1308:      start, length (MSB), length (LSB), DATA, checksum */
1309:   l = 3 + length + 1;
1310:
1311:   /* prepare memory */
1312:   pkt = Xcalloc(sizeof(t_data));
1313:
1314:   /* put start byte on */
1315:   pkt->data[0] = 0x7E;
1316:
1317:   /* copy data into packet */
1318:   for (t = 0, i = 0, o = 1, m = 1; i <= length; o++, m++) {
1319:     /* if: its time for the checksum */
1320:     if (i == length) d = M8((0xFF - M8(t)));
1321:     /* if: its time for the high length byte */
1322:     else if (m == 1) d = M8(length >> 8);
1323:     /* if: its time for the low length byte */
1324:     else if (m == 2) d = M8(length);
1325:     /* if: its time for the normal data */
1326:     else if (m > 2) d = data[i];
1327:
1328:     x = 0;
1329:     /* check for any escapes needed */
1330:     if ((d == 0x11) || /* XON */
1331:         (d == 0x13) || /* XOFF */
1332:         (d == 0x7D) || /* Escape */
1333:         (d == 0x7E)) { /* Frame Delimiter */
1334:       l++;
1335:       pkt->data[o++] = 0x7D;
1336:       x = 1;
1337:     }
1338:
1339:     /* move data in */
1340:     pkt->data[o] = ((!x)?d:d^0x20);
1341:     if (m > 2) {
1342:       i++;
1343:       t += d;
1344:     }
1345:   }
1346:
1347:   /* remember the length */
1348:   pkt->length = l;
1349:
1350:   return pkt;
1351: }
```