```
#include <px4_config.h>
#include <px4_time.h>
#include <px4_tasks.h>
#include <px4_defines.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <fcntl.h>
#include <string.h>
#include <drivers/drv_hrt.h>
#include <drivers/drv_accel.h>
#include <drivers/drv_gyro.h>
#include <drivers/drv_mag.h>
#include <drivers/drv_baro.h>
#include <drivers/drv_range_finder.h>
#include <drivers/drv_rc_input.h>
#include <time.h>
#include <float.h>
#include <unistd.h>
#include <termios.h>
#include <errno.h>
#include <stdlib.h>
#include <poll.h>


#include <px4_config.h>
#include <nuttx/sched.h>
#include <systemlib/systemlib.h>
#include <systemlib/err.h>
#include <uORB/uORB.h>
#include <uORB/topics/probe_parameters.h>

static bool thread_should_exit = false;       /**< daemon exit flag */
static bool thread_running = false;           /**< daemon status flag */
static int daemon_task;                       /**< Handle of daemon task
/ thread */


__EXPORT int probe_pub_main(int argc, char *argv[]);


int probe_pub_thread_main(int argc, char *argv[]);

static void usage(const char *reason);

static void usage(const char *reason)
{
      if (reason)
      {
            warnx("%s\n", reason);
      }
      warnx("usage: daemon {start|stop|status} [-p <additional
params>]\n\n");
}

int probe_pub_main(int argc, char *argv[])
{
      if (argc < 2) {
            usage("missing command");
            return 1;
      }

      if (!strcmp(argv[1], "start")) {

            if (thread_running) {
                  warnx("daemon already running\n");
                  /* this is not an error */
                  return 0;
            }

            thread_should_exit = false;
            daemon_task = px4_task_spawn_cmd("daemon",
                                  SCHED_DEFAULT,
                                  SCHED_PRIORITY_DEFAULT,
                                  2000,
                                 probe_pub_thread_main,
                                  (argv) ? (char *const *)&argv[2] :
(char *const *)NULL);
            return 0;
      }

      if (!strcmp(argv[1], "stop")) {
            thread_should_exit = true;
            return 0;
      }

      if (!strcmp(argv[1], "status")) {
            if (thread_running) {
                  warnx("\trunning\n");

            } else {
                  warnx("\tnot started\n");
            }

            return 0;
      }

      usage("unrecognized command");
      return 1;
}

int probe_pub_thread_main(int argc, char *argv[])
{

      warnx("[daemon] starting\n");
      thread_running = true;
      int ser_port =-1;
      char port[] = "/dev/ttyS1"; /* port to connect to */
            speed_t baud = B115200; /* baud rate */

            ser_port=open(port,O_RDWR|O_NOCTTY |O_NDELAY |O_NONBLOCK);;
/* connect to port */

            /* set the other settings (in this case, 115200 8N1) */
            struct termios settings;
            tcgetattr(ser_port, &settings);

            cfsetispeed(&settings, baud);
            cfsetospeed(&settings, baud); /* baud rate */
            settings.c_cflag &= ~PARENB; /* no parity */
            settings.c_cflag &= ~CSTOPB; /* 1 stop bit */
            settings.c_cflag &= ~CSIZE;
            settings.c_cflag |= CS8 | CLOCAL; /* 8 bits */
            //settings.c_lflag = ICANON; /* canonical mode */
            //settings.c_oflag &= ~OPOST; /* raw output */

            tcsetattr(ser_port,TCSANOW, &settings); /* apply the settings
*/
            tcflush(ser_port,TCOFLUSH);

      struct probe_parameters_s att;
      memset(&att,0,sizeof(att));
      orb_advert_t att_pub = orb_advertise(ORB_ID(probe_parameters),
&att);


      uint8_t buf[32]={0};
      uint8_t temp_data[32]={0};uint8_t data[32]={0};
      struct pollfd fds[1];
      bool pkt_rx=false;
      uint8_t probe_alpha[4];uint8_t probe_beta[4];uint8_t probe_v[4];

      fds[0].fd = ser_port;
      fds[0].events = POLLIN;
      int timeout=5;
      ssize_t nread = 0;
      int i=0,ctr=0;

      while (!thread_should_exit)
      {
            if(poll(&fds[0],1,timeout) > 0)
            {
                  nread =read(ser_port,buf,sizeof(buf));
                  if(nread>0)
                  {
                        memcpy(&temp_data,&buf,nread);
                        for(i=0;i<nread;i++)
                        {
                              if(temp_data[i]==0xFF)
                              {
                                    ctr=0;
                              }
                              else if(temp_data[i]==0xFE)
                              {
                                    pkt_rx=true;
                                    data[ctr]=temp_data[i];
                              }

                              else
                              {
                                    data[ctr]=temp_data[i];
                                    ctr++;
                              }
                              if(pkt_rx && (data[12]==0xFE))
                              {
                                    memcpy(&probe_v,&data[0],4);
                                    memcpy(&probe_alpha,&data[4],4);
                                    memcpy(&probe_beta,&data[8],4);



      //memcpy(&att.probe_velocity,&probe_v,4);

      //memcpy(&att.probe_alpha,&probe_alpha,4);

      //memcpy(&att.probe_beta,&probe_beta,4);

      orb_publish(ORB_ID(probe_parameters),att_pub, &att);

                                    /*for(int k=0;k<4;k++)
                                          printf("%2x_",probe_v[k]);
                                    for(int k=0;k<4;k++)
                                          printf("%2x_",probe_alpha[k]);
                                    for(int k=0;k<4;k++)
                                          printf("%2x_",probe_beta[k]);
                                    printf("\n");*/
                                    pkt_rx=false;
                                    for(int k=0;k<=ctr;k++)
                                          data[k]='\0';

                              }
                        }

                  }
            }
      }
      warnx("[daemon] exiting.\n");
      close(ser_port);
      thread_running = false;

      return 0;
}
```