Tony Morelli

CPE606 Final Project

Xbox Controller Controller

5/2/2006

In a different project, it was required to remotely control an XBOX video game system. For that project, I hacked apart an XBOX controller and inserted relays controlled by a PC parallel port. When the relays tripped, they would close the circuit on the XBOX controller which would make the XBOX think someone physically pushed the button. This worked good, however there were a few problems with it. The XBOX controller was ruined, and it required it to be hard wired into a controlling computer. For this project in the *Real Time Computing* class, a mechanical pressing device was created that would overcome these issues. Mechanical arms will press the buttons on the XBOX controller, and the controlling PC will not be required to be hard wired to the mechanical arms. The project is organized into 4 parts: Hardware Design, PIC Design, ARM Receiver Design, and Remote Controller Design.

The overall hardware design is as follows. The mechanical arms are made up of thick wire that can be bent into shape, but will not bend when little force is applied to them as they press the XBOX controller buttons. For this I used thick wire coat hangars, with rubber pads as feet that make contact with the XBOX controller buttons to avoid slippage. The mechanical arms are attached to servos which move in the appropriate direction to either press or not press a certain button. For this project, 8 buttons on the controller are supported. They are Up, Down, Left, Right, A, B, X and Y. All the mapped buttons are digital (they are either on or off), the analog buttons were left out for now, although with the servos

controlling them it is possible to add them at a later date. That is all that is involved with the hardware, the real brain of the project is contained in the software modules.

A PIC controls the servos. A Basic Stamp 2 was chosen as the controller. The BS2 can support up to 15 outputs, for this project I used 8 connected to each of the servos. The BS2 receives the desired status of the servos over a serial port at 38400 Baud. The PIC code is very straight forward. It reads data from the serial port and expects it to be in the following format. A message starts with the character 'A'. Following the 'A' are 8 more characters which are the status of the 8 servos. 1 for down and 0 for up. The PIC simply reads off the serial port looking for an 8, reads the next 8 bytes, then sets the servos accordingly.

An ARM based Gumstix (www.gumstix.org) is used for the receiver of the desired status of the XBOX controller. The gumstix receives UDP commands over a wifi network, and then translates them into the protocol that the Basic Stamp 2 understands and sends the commands out the serial port. The gumstix is a very small ARM based 400 MHz computer running a stripped down version of the 2.6 linux kernel. The gumstix has an available Compact Flash socket, and a Netgear wifi card was placed into that socket to allow the gumstix to communicated wirelessly with the controller. The software running on the gumstix is a simple program written in C with 2 threads. The first thread listens for UDP broadcasts containing information from a controller about what the status of the servos should be. The status of the servos is saved in global

variables available to both threads. The second thread handles the serial com with the Basic Stamp 2. This thread sleeps for 250 ms, then wakes up and checks the status of the servo variables and sends them out the serial port. 250 ms was chosen as it seems to be the fastest that the Basic Stamp 2 could keep up with.

Any device talking the correct UDP protocol can be used as a controller. The UDP protocol is defined by a start of message code, '##', followed by key characters (U for up, D for down, L for left, R for right, A for a, B for b, X for x, Y for y) with their state (1 for down, 0 for up), followed by the end of message characters, '!!'. For this controller I used the Playstation Portable as it is wireless and simple to code for. The code for this part of the project is a simple single threaded app written in C. The code basically scans the buttons for their status, formats a UDP message and sends it to the gumstix. The app then sleeps for 250 ms to try and stay in synch with the 250 ms pulses that are being sent to the Basic Stamp 2.

Several issues were encountered during the duration of this project. None are more severe than the delay from the time a button is pushed on the controller to the time the servo pushes the button on the XBOX controller. With the 250 ms polling cycle, it is possible to miss button pushes, or have a lag of up to 500 ms. These are worst case scenarios, and for the most part it works correctly. One way to improve this is to make the servo movements quicker, which really means make their movements smaller. This is possible with a very accurately measured arm, and using a stiffer metal than a coat hanger. Future versions of this project

will experiment with those.

Overall this was a really fun project. Getting all the different embedded platforms talking together was quite the feat. I would have liked to have spent more time on the hardware aspect of this project, but the software was challenging and fun. This project utilized a Playstation Portable, a gumstix 400, and a Basic Stamp 2 to control an XBOX controller.

Basic Stamp 2 Code:

```
'{$STAMP BS2}
'{$PBASIC 2.5}

counter VAR Word
BUT VAR Byte
DIR VAR Byte

but1 VAR Byte
but2 VAR Byte
but3 VAR Byte
but4 VAR Byte
but5 VAR Byte
but6 VAR Byte
but7 VAR Byte
but8 VAR Byte



SYNCH CON "A"    'Establish synchronization byte
'BAUD  CON 16780  'N2400 baud (MAX)
BAUD  CON 6  'N2400 baud (MAX)
DAT   VAR Byte   'Data storage variable
DIRH = %11111111 'All outputs

START:
   DEBUG "At start"
'  SERIN 0,BAUD,[WAIT(SYNCH),DAT]
'  DEBUG "hello2"
'  SERIN 16,84,[BUT, DIR]
'  SERIN 16,84,[BUT]
  SERIN 16,6,[BUT]

  IF BUT <> "A" THEN
    GOTO START
  ENDIF
'  DEBUG "Start of message"
  SERIN 16,6,[but1,but2,but3,but4,but5,but6,but7,but8]
'  but1 = "1"


  FOR counter = 1 TO 2
    IF (but1 = "1") THEN
 '     DEBUG "BUT 1 is 1"
      PULSOUT 14,100
    ENDIF
    IF (but2 = "1") THEN
      PULSOUT 13,100
    ENDIF
    IF (but3 = "1") THEN
      PULSOUT 12,100
    ENDIF
    IF (but4 = "1") THEN
      PULSOUT 11,100
```

```
    ENDIF
    IF (but5 = "1") THEN
      PULSOUT 10,100
    ENDIF
    IF (but6 = "1") THEN
      PULSOUT 9,100
    ENDIF
    IF (but7 = "1") THEN
      PULSOUT 8,100
    ENDIF
    IF (but8 = "1") THEN
      PULSOUT 7,100
    ENDIF

 '   PAUSE 5
  NEXT
  FOR counter = 1 TO 2
    IF (but1 = "1") THEN
      PULSOUT 14,150
    ENDIF
    IF (but2 = "1") THEN
      PULSOUT 13,150
    ENDIF
    IF (but3 = "1") THEN
      PULSOUT 12,150
    ENDIF
    IF (but4 = "1") THEN
      PULSOUT 11,150
    ENDIF
    IF (but5 = "1") THEN
      PULSOUT 10,150
    ENDIF
    IF (but6 = "1") THEN
      PULSOUT 9,150
    ENDIF
    IF (but7 = "1") THEN
      PULSOUT 8,150
    ENDIF
    IF (but8 = "1") THEN
      PULSOUT 7,150
    ENDIF
 '   PAUSE 5
  NEXT
  GOTO START
END
```

Playstation Portable Code:

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspctrl.h>
#include <stdio.h>

#include <pspkernel.h>
#include <pspdebug.h>
#include <pspsdk.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pspnet.h>
#include <pspnet_inet.h>
#include <pspnet_apctl.h>
#include <pspnet_resolver.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <errno.h>

#define BUFLEN 512
#define NPACK 10
#define PORT 9930
char SRV_IP[16];

#define bool int
#define true 1
#define false 0

PSP_MODULE_INFO("xbox_ctrl", 0x1000, 1, 1);
PSP_MAIN_THREAD_ATTR(0);
#define printf pspDebugScreenPrintf
/*****************************************************/
void diep(char *s)
{
  perror(s);
  exit(1);
}

struct sockaddr_in si_me, si_other;
int playerSocket;
bool done = false;
enum
{
```

```
  up = 0,
  down,
  left,
  right,
  cross,
  circle,
  square,
  triangle,
  totalButtons,

};
int buttonStatus[totalButtons];



void InitScreen();
int InitClientSocket();
void ProcessInputs();
int connect_to_apctl(int config);
void ConvertIpToChar(int * _ipDigits, char * _ip);
void GetServerIp(char * _ip);
int GetInput();
void InitButtons();

int exit_callback(int arg1, int arg2, void * common)
{
  sceKernelExitGame();
  return 0;

}

int CallbackThread(SceSize args, void * argp)
{
  int cbid;
  cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
  sceKernelRegisterExitCallback(cbid);
  sceKernelSleepThreadCB();
  return 0;
}

int SetupCallbacks(void)
{
  int thid = 0;
  thid = sceKernelCreateThread("update_thread", CallbackThread, 0x11,
                               0xFA0, 0, 0);
  if (thid >= 0)
```

```
  {
    sceKernelStartThread(thid, 0, 0);
  }
  return thid;
}

int main()
{
  pspDebugScreenInit();
  SetupCallbacks();
  printf("Setting up client socket\n");

  GetServerIp(SRV_IP);
  pspDebugScreenSetXY(0,10);
  InitClientSocket();

  sceKernelDelayThread(500000);

  InitButtons();

  while (!done)
  {
    ProcessInputs();
  }
  sceKernelSleepThread();
  return 0;
}
void InitButtons()
{
  int x = 0;
  for (x = 0; x< totalButtons; x++)
    buttonStatus[x] = 0;
}

/*****************************************************************/
int InitClientSocket()
{
  int s, err, i, j, slen=sizeof(si_other);
  char buf[BUFLEN];
  printf("INITCLIENTSOCKET!\n");
  if (pspSdkLoadInetModules() < 0)
  {
    printf("COULD NOT LOAD INET MODULES!\n");
    sceKernelSleepThread();
    return 0;
  }
```

```
  printf("InetModules Loaded!\n");

/*
  thid = sceKernelCreateThread("net_thread", net_thread, 0x18, 0x10000,
PSP_THREAD_ATTR_USER, NULL);
  if (thid < 0)
  {
    printf("Error could not create thread!\n");
    sceKernelSleepThread();
    return 0;
  }
  sceKernelStartThread(thid, 0, NULL);
  sceKernelExitDeleteThread(0);
  sceKernelSleepThread();
*/
  if((err = pspSdkInetInit()))
  {
    printf(": Error, could not initialise the network \n", err);
  }
  printf("Now connecting to 0\n");
  if(connect_to_apctl(0))
  {
    // connected, get my IPADDR and run test
    char szMyIPAddr[32];
    printf("Success!\n");
    if (sceNetApctlGetInfo(8, szMyIPAddr) != 0)
      strcpy(szMyIPAddr, "unknown IP address");
  }

  if ((playerSocket=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
   diep("socket");

  memset((char *) &si_other, sizeof(si_other), 0);
  si_other.sin_family = AF_INET;
  si_other.sin_port = htons(PORT);
  if (inet_aton(SRV_IP, &si_other.sin_addr)==0)
  {
    fprintf(stderr, "inet_aton() failed\n");
    exit(1);
  }
  printf("Sending packet %d\n", i);
  sprintf(buf, "This is packet %d\n", i);
  if (sendto(playerSocket, buf, BUFLEN, 0, &si_other, slen)==-1)
    diep("sendto()");
}
int connect_to_apctl(int config)
```

```
{
  int err;
  int stateLast = -1;
  printf("Inside connect_to_ap\n");
  /* Connect using the first profile */
  err = sceNetApctlConnect(config);
  if (err != 0)
  {
    printf(": sceNetApctlConnect returns %08X\n", err);
    return 0;
  }

  printf(": Connecting...\n");
  while (1)
  {
    int state;
    err = sceNetApctlGetState(&state);
    if (err != 0)
    {
      printf(": sceNetApctlGetState returns $%x\n" , err);
      break;
    }
    if (state > stateLast)
    {
      printf("  connection state %d of 4\n", state);
      stateLast = state;
    }
    if (state == 4)
      break;  // connected with static IP

    // wait a little before polling again
    sceKernelDelayThread(50*1000); // 50ms
  }
  printf(": Connected!\n");

  if(err != 0)
  {
    return 0;
  }
  return 1;
}
#if 0
  char buf[512];
  memset(buf, 0, 512);
  BuildMessage(buf);
  if (sendto(playerSocket,buf,BUFLEN,0,&si_other,slen) == -1)
```

```
    printf("Could not send packet!\n");
  printf("\x1b[19;10H");
  printf("Message Sent      ");
#endif

/*************************************************************/

void ProcessInputs()
{
  SceCtrlData pad;
  bool success = false;
  while(!success)
  {
    sceCtrlReadBufferPositive(&pad, 1);
    InitButtons();
    if (pad.Buttons !=0)
    {
      if (pad.Buttons & PSP_CTRL_SQUARE)
      {
        buttonStatus[square] = 1;
        //Square
      }
      if (pad.Buttons & PSP_CTRL_TRIANGLE)
      {
        buttonStatus[triangle] = 1;
        //Triangle
      }
      if (pad.Buttons & PSP_CTRL_CIRCLE)
      {
        buttonStatus[circle] = 1;
        //done = true;
        //Circle
      }
      if (pad.Buttons & PSP_CTRL_CROSS)
      {
        buttonStatus[cross] = 1;
        //Cross
      }
      if (pad.Buttons & PSP_CTRL_UP)
      {
        buttonStatus[up] = 1;
        //Up
      }
      if (pad.Buttons & PSP_CTRL_DOWN)
      {
        buttonStatus[down] = 1;
```

```
      //DOWN
    }
    if (pad.Buttons & PSP_CTRL_LEFT)
    {
      buttonStatus[left] = 1;
      //Left
    }
    if (pad.Buttons & PSP_CTRL_RIGHT)
    {
      buttonStatus[right] = 1;
      //Right
    }
    if (pad.Buttons & PSP_CTRL_START)
    {
      //Start
    }
    if (pad.Buttons & PSP_CTRL_SELECT)
    {
      //Select
    }
    if (pad.Buttons & PSP_CTRL_LTRIGGER)
    {
      //LTRIGGER
    }
    if (pad.Buttons & PSP_CTRL_RTRIGGER)
    {
      //RTrigger
    }

    sceKernelDelayThread(100000);

   }
   SendButtonStatus();
  }

}
void SendButtonStatus()
{

  int slen = sizeof(si_other);
  char buf[512];
  memset(buf, 0, 512);
  sprintf(buf,"##U%dD%dL%dR%dA%dB%dX%dY%d!!",
              buttonStatus[up],
              buttonStatus[down],
              buttonStatus[left],
```

```
            buttonStatus[right],
            buttonStatus[cross],
            buttonStatus[circle],
            buttonStatus[square],
            buttonStatus[triangle]);

  if (sendto(playerSocket,buf,BUFLEN,0,&si_other,slen) == -1)
    printf("Could not send packet!\n");

}
/*************************************************************/
void GetServerIp(char * _ip)
{
  int xPos = 0;
  int x;
  bool ipDone = false;
  int button = 0;
  int ipDigits[16] ;
  for (x = 0; x< 16; x++)
    ipDigits[x] = 0;
  pspDebugScreenSetXY(0,5);
  printf("Please Enter Server Ip Address and press X when done");
  pspDebugScreenSetXY(0,6);
  printf("000.000.000.000");
  while (!ipDone)
  {
    sceKernelDelayThread(250000);
    button = GetInput();
    if (button == PSP_CTRL_CROSS)
    {
      ipDone = true;
    }
    if (button == PSP_CTRL_UP)
    {
      ipDigits[xPos] ++;
      if (ipDigits[xPos] > 9)
        ipDigits[xPos] = 0;
      pspDebugScreenSetXY(xPos,6);
      printf("%d", ipDigits[xPos]);
    }
    if (button == PSP_CTRL_DOWN)
    {
      ipDigits[xPos] --;
      if (ipDigits[xPos] < 0)
        ipDigits[xPos] = 9;
      pspDebugScreenSetXY(xPos,6);
```

```
      printf("%d", ipDigits[xPos]);
    }
    if (button == PSP_CTRL_RIGHT)
    {
      pspDebugScreenSetXY(xPos,7);
      printf(" ");
      xPos ++;
      if (xPos == 3 || xPos == 7 || xPos == 11)
        xPos++;
      if (xPos > 14)
        xPos = 0;
      pspDebugScreenSetXY(xPos,7);
      printf("^");
    }
    if (button == PSP_CTRL_LEFT)
    {
      pspDebugScreenSetXY(xPos,7);
      printf(" ");
      xPos --;
      if (xPos == 3 || xPos == 7 || xPos == 11)
        xPos--;
      if (xPos <0 )
        xPos = 14;
      pspDebugScreenSetXY(xPos,7);
      printf("^");
    }
    ConvertIpToChar(ipDigits,_ip);


  }


}
/*****************************************************************/
void ConvertIpToChar(int * _ipDigits, char * _ip)
{
  int x = 0;
  int totalChars = 0;
  bool start = true;
  for (x = 0; x<15; x++)
  {
    if (x == 3 || x == 7 || x == 11)
    {
      _ip[totalChars] = '.';
      start = true;
      totalChars ++;
```

```
}
else if (start && _ipDigits[x] == 0)
  continue;
else
{
  switch (_ipDigits[x])
  {
    case (0):
    {
      _ip[totalChars] = '0';
      totalChars ++;
      break;
    }
    case (1):
    {
      _ip[totalChars] = '1';
      totalChars ++;
      break;
    }
    case (2):
    {
      _ip[totalChars] = '2';
      totalChars ++;
      break;
    }
    case (3):
    {
      _ip[totalChars] = '3';
      totalChars ++;
      break;
    }
    case (4):
    {
      _ip[totalChars] = '4';
      totalChars ++;
      break;
    }

    case (5):
    {
      _ip[totalChars] = '5';
      totalChars ++;
      break;
    }
    case (6):
    {
```

```
          _ip[totalChars] = '6';
          totalChars ++;
          break;
        }
        case (7):
        {
          _ip[totalChars] = '7';
          totalChars ++;
          break;
        }
        case (8):
        {
          _ip[totalChars] = '8';
          totalChars ++;
          break;
        }
        case (9):
        {
          _ip[totalChars] = '9';
          totalChars ++;
          break;
        }
      }
      start = false;
    }
  }
  _ip[totalChars] = '\0';



}
/*************************************************************/
int GetInput()
{
  SceCtrlData pad;
  bool success = false;
  while(!success)
  {
    sceCtrlReadBufferPositive(&pad, 1);
    if (pad.Buttons !=0)
    {
      if (pad.Buttons & PSP_CTRL_SQUARE)
      {
        success = true;
        return PSP_CTRL_SQUARE;
        //Square
```

```
}
if (pad.Buttons & PSP_CTRL_TRIANGLE)
{
 success = true;
 return PSP_CTRL_TRIANGLE;
 //Triangle
}
if (pad.Buttons & PSP_CTRL_CIRCLE)
{
 success = true;
 return PSP_CTRL_CIRCLE;
 //Circle
}
if (pad.Buttons & PSP_CTRL_CROSS)
{
 success = true;
 return PSP_CTRL_CROSS;
 //Cross
}
if (pad.Buttons & PSP_CTRL_UP)
{
 success = true;
 return PSP_CTRL_UP;
 //Up
}
if (pad.Buttons & PSP_CTRL_DOWN)
{
 success = true;
 return PSP_CTRL_DOWN;
 //DOWN
}
if (pad.Buttons & PSP_CTRL_LEFT)
{
 success = true;
 return PSP_CTRL_LEFT;
 //Left
}
if (pad.Buttons & PSP_CTRL_RIGHT)
{
 success = true;
 return PSP_CTRL_RIGHT;
 //Right
}
if (pad.Buttons & PSP_CTRL_START)
{
 success = true;
```

```
      return PSP_CTRL_SQUARE;
      //Start
    }
    if (pad.Buttons & PSP_CTRL_SELECT)
    {
      success = true;
      return PSP_CTRL_SELECT;
      //Select
    }
    if (pad.Buttons & PSP_CTRL_LTRIGGER)
    {
      success = true;
      return PSP_CTRL_LTRIGGER;
      //LTRIGGER
    }
    if (pad.Buttons & PSP_CTRL_RTRIGGER)
    {
      success = true;
      return PSP_CTRL_RTRIGGER;
      //RTrigger
    }
    sceKernelDelayThread(250000);
  }
 }
}
```

Gumstix Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <string.h>

#include <pthread.h>
#include <termios.h>
#include <sys/types.h>
	#include <sys/stat.h>
	#include <fcntl.h>



#define BUFLEN 512
#define NPACK 10
#define PORT 9930
#define SRV_IP "127.0.01"
//#define SRV_IP "192.168.1.40"


#define THREAD_DELAY 100000


#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS0"



enum
{
  up = 0,
  down,
  left,
  right,
  A,
  B,
  X,
  Y,
  start,
  back,
  totalButtons,
};

int buttonStatus[totalButtons];

#define DEBUG 0


/*************************************************************/
```

```
void diep(char *s)
{
  perror(s);
  exit(1);
}

/**************************************************************/

struct sockaddr_in si_me, si_other;
int playerSocket;

void InitServerSocket();
void ProcessMsg(char * buf);
void RecvCmd();
void * SerialWriteThread(void *param);
void InitButtons();
void BuildSerialMessage(char * _buf);

/**************************************************************/
int main(int argc, char ** argv)
{
  int sw;
  pthread_t serialWriteId;

  InitButtons();

  sw = pthread_create(&serialWriteId, NULL, SerialWriteThread, NULL);
  if (sw != 0)
  {
    printf("Error creating serial thread, exiting...\n");
    exit(0);
  }
  InitServerSocket();
  while (1)
  {
//    printf("In mail while loop!\n");
//    usleep(THREAD_DELAY);
    RecvCmd();
  }
  return 0;
}
/**************************************************************/
void InitButtons()
{
  int x = 0;
  for (x = 0; x< totalButtons; x++)
  {
    buttonStatus[x] = 0;
  }
}
/**************************************************************/
void RecvCmd()
{
  int slen=sizeof(si_other);
  char buf[BUFLEN];
```

```
  if (recvfrom(playerSocket, buf, BUFLEN, 0, &si_other, &slen)==-1)
    diep("recvfrom()");
  ProcessMsg(buf);
}
/************************************************************/
void ProcessMsg(char * buf)
{
  int x = 0;
//  printf("Received msg: %s\n", buf);
  if (buf[0] == '#' && buf[1] == '#')
  {
    x = 2;
    while (buf[x] != '!' && x < BUFLEN)
    {
      if (buf[x] == 'U')
      {
        buttonStatus[up] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'D')
      {
        buttonStatus[down] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'L')
      {
        buttonStatus[left] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'R')
      {
        buttonStatus[right] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'A')
      {
        buttonStatus[A] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'B')
      {
        buttonStatus[B] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'X')
      {
        buttonStatus[X] = atoi(&buf[x+1]);
        x+=2;
      }
      else if (buf[x] == 'Y')
      {
        buttonStatus[Y] = atoi(&buf[x+1]);
        x+=2;
      }
    }
  }
```

```
}

/*************************************************************/
void InitServerSocket()
{
  int x, slen=sizeof(si_other);
  char buf[BUFLEN];

  if ((playerSocket=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
    diep("socket");

  memset((char *) &si_me, sizeof(si_me), 0);
  si_me.sin_family = AF_INET;
  si_me.sin_port = htons(PORT);
  si_me.sin_addr.s_addr = htonl(INADDR_ANY);
  if (bind(playerSocket, &si_me, sizeof(si_me))==-1)
    diep("bind");
  printf("SOCKET UP\n");
  if (recvfrom(playerSocket, buf, BUFLEN, 0, &si_other, &slen)==-1)
    diep("recvfrom()");
  printf("Client COnnected!\n");
  printf("Port: %d\n", si_other.sin_port);
  char * test = (char *)&si_other;
  for (x = 0; x< sizeof(si_other); x++)
    printf("%x ", test[x]);
}
/*************************************************************/
void * SerialWriteThread(void * param)
{
  char buf[BUFLEN];
  char msg[BUFLEN];
  int fd,c, res, x;
  struct termios oldtio,newtio;
  fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
  if (fd <0) {perror(MODEMDEVICE); exit(-1); }

  tcgetattr(fd,&oldtio); /* save current serial port settings */
  bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings */

  newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD ;

  //newtio.c_iflag = IGNPAR | ICRNL;

  newtio.c_oflag = 0;

//  newtio.c_lflag = ICANON;

  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);

#if 0

speed 38400 baud;
-brkint ixoff -imaxbel
```

```
#endif
  printf("Serial Write Thread Starting Up!\n");
  while(1)
  {
//    sprintf(buf, "AA");
//    write(fd,buf,1);
    BuildSerialMessage(buf);
    printf("Send Serial Message: %s\n", buf);
    for (x = 0; x< 10; x++)
    {
      write(fd,&buf[x],1);
      usleep(1000);
    }
//    write(fd,buf,10);
    memset(buf,0,BUFLEN);
    usleep(THREAD_DELAY);

  }
}
/*************************************************************/
void BuildSerialMessage(char * _buf)
{
//  sprintf(_buf,"##U%dD%dL%dR%dA%dB%dX%dY%d!!",
  sprintf(_buf,"A%d%d%d%d%d%d%d%d",
                  buttonStatus[0],
//                  1,
                  buttonStatus[1],
                  buttonStatus[2],
                  buttonStatus[3],
                  buttonStatus[4],
                  buttonStatus[5],
                  buttonStatus[6],
                  buttonStatus[7]);


}
/*************************************************************/
```