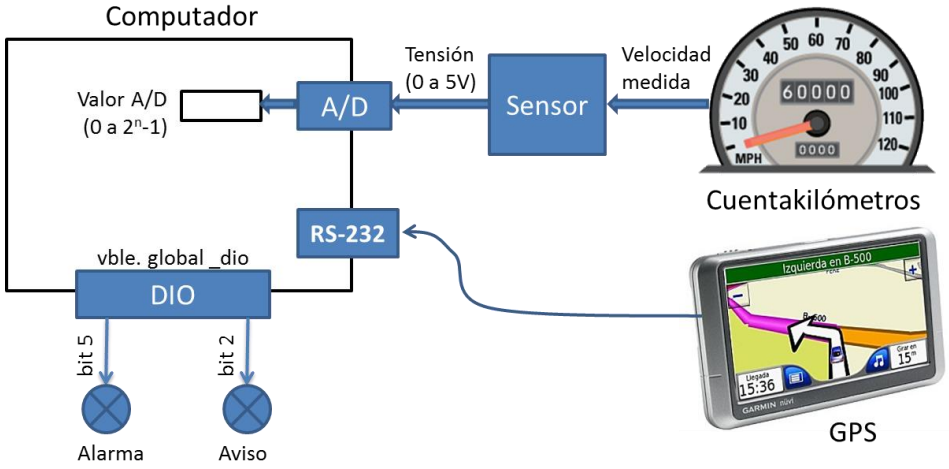
Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2013

Se desea realizar un sistema de control para comprobar la exactitud de la medición de velocidad del cuentakilómetros de un vehículo. Para ello, se dispone del montaje siguiente:



La velocidad medida por el cuentakilómetros es captada por un sensor que entrega una tensión entre 1V (para 0Km/h) y 4V (para 180Km/h), y adquirida mediante un conversor A/D de 12 bits.

El GPS está conectado por RS-232 a la entrada estándar (consola), y envía cada segundo una cadena de caracteres (longitud máxima 50) con el siguiente formato:

campo0 : campo1 : campo2 : campo3 : campo4 : campo5 \n

Dónde:

campo4 es la velocidad en Km/h (ej: 75.2).

y el nº de caracteres de cada campo puede variar.

La comprobación de funcionamiento se realizará de la siguiente manera:

- Se adquiere cada 50 ms la velocidad del cuentakilómetros, calculando el valor absoluto de las diferencias entre la medida del cuentakilómetros y la medida actual del GPS.
- Se guardan los *n* valores más recientes de las diferencias anteriores.
- Si el máximo de los datos guardados es mayor que 2 Km/h, se activa la señal de aviso escribiendo un 1 en el bit 2 del puerto de E/S digital DIO (accesible mediante variable global *_dio*). En caso contrario, se borra la señal de aviso.
- Si el máximo de los datos guardados es mayor que 10 Km/h, se activa la señal de alarma, escribiendo un 1 en el bit 5 del puerto de E/S digital DIO. En caso contrario, se borra la señal de alarma.
- Se dispone de las siguientes funciones para la conversión A/D:

```
// Inicialización de un temporizador que llame a la función indicada en
// FnCallback cada cierto intervalo (en ms)
void InitTemporizador(int tiempo,void (*FnCallback)();

// Obtención del valor de conversion A/D del canal deseado utilizando
// nBits : resultado de 0 a 2^n-1, para tensión de entrada 0 a 5V.
int ConversionAD(int nCanal,int n);
```

- La obtención de datos del GPS se realiza mediante la función gets() en main().
- El valor de *n* (nº de medidas a guardar y comprobar) se pide por teclado al principio del programa. No se puede suponer un valor máximo para *n*.

Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2013

1) *1 punto* Escribir una función que, dada una cadena de caracteres con el formato enviado por el GPS, extraiga de la misma el valor real del campo deseado, indicado mediante su índice (índice 0 para el primer campo).

Ej: cadena = "0.4 : -2.75 : 0 : -42.54 : 5 : 1.5"
ExtraerValor(cadena,3) debe devolver -42.54

2) *1 punto* Realizar una función que calcule el máximo de una tabla de reales.

3) *1 punto* Realizar una función que, dado un valor real a comprobar, su límite superior y un nº de bit, active la salida DIO correspondiente al nº de bit indicado si el valor a comprobar supera el límite, y la desactive en caso contrario. Los bits de salida digital se encuentran en la variable global *_dio*. No se puede modificar más que el bit deseado.

Ej:

Valor anterior de _dio	Código ejecutado	Valor posterior de _dio
00…00001010110	ActivarSalida(27.5,22.3,3);	00…0000101**1**110
00…00001011110	ActivarSalida(27.5,32.3,1);	00…0000101111**0**0

4) *3 puntos* Realizar, utilizando las funciones anteriores y otras que se considere necesarias, el programa que adquiere los datos del GPS y del cuentakilómetros, realiza las comprobaciones, y activa las alarmas.

Se desea una mejora sobre el programa anterior: que los límites de alarma y aviso puedan ser modificados mediante una comunicación TCP/IP desde otro equipo.

5) *2 puntos* Realizar el código necesario para escribir un hilo con un socket servidor que espere por una conexión en el puerto 3500, dirección ip 10.20.30.40, y cuando se establezca una conexión espere por una cadena de caracteres (tamaño máximo 20) con los valores de aviso y alarma separados por espacio (ej: "4.5 19"), actualizando los valores de aviso y alarma que se utilizan en la comprobación. Tras recibir este mensaje, se cierra la conexión y se espera por una nueva.

6) *1 punto* Describir las modificaciones necesarias en el programa del ejercicio 4) para lanzar el nuevo hilo, y que se puedan compartir los límites de alarma y aviso de forma segura.

7) *1 punto* Indicar la secuencia de operaciones que se debe realizar para servir un temporizador en POSIX de forma asíncrona. (No es necesario escribir el programa, sólo indicar los pasos que se darían y qué funciones se llamarían en cada caso).

Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2013

Paso de parámetros a main() desde línea de comandos:

```
int main(int argc,char* argv[])  // argv[1] es una cadena de caracteres para el
                                 // 1er parámetro, argv[2] para el 2°, etc.
```

Algunas funciones de C:

```
int atoi(const char* cadena);
double atof(const char* cadena);
double strtod(const char* cadena,char** final);

int strlen(const char* cadena);
char* strcpy(char* dst,const char* src);
char* strncpy(char* dst,const char* src,int n);
char* strcat(char* dst,const char* src);
char* strncat(char* dst,const char* src,int n);
int strcmp(const char* cad1,const char* cad2);
int strncmp(const char* cad1,const char* cad2,int n);
char* strchr(const char* cad,char c);
char* strstr(const char* cad,const char* busca);
int sprintf(char* dest,const char* fmt,...);
int sscanf(const char* src,const char* fmt,...);

FILE* fopen(const char* filename,const char* mode);
char* fgets(char* dest,int tam_max,FILE* f);
int fprintf(FILE* fid,const char* fmt,...);
int fscanf(FILE* fid,const char* fmt,...);
int fclose(FILE* fid);

void* malloc(int tam);
void free(void* pt);
```

Documentación de sockets:

```
SOCKET sock;

SOCKET socket(int address_family,int sock_type, int protocol);
// Usar address_family=AF_INET, sock_type = SOCK_DGRAM o SOCK_STREAM, protocol=0

int bind(SOCKET sock,struct sockaddr_in *addr_local,int tam_sockaddr_in);
int connect(SOCKET sock, struct sockaddr_in *addr_remoto,int tam_sockaddr_in);
int listen(SOCKET sock,int maximo);
int accept(SOCKET sock, struct sockaddr_in *addr_remoto,int tam_sockaddr_in);
int send(SOCKET sock,const char* datos,int nbytes,int flags); // Usar flags=0
int recv(SOCKET sock,char* datos,int nbytes,int flags); // Usar flags=0
int closesocket(SOCKET sock);


struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};

Para dirección IP, usar:
       addr_laquesea.sin_addr.S_un.S_addr=inet_addr(cad. caract. dirección ip);

Para el Puerto, usar:
       addr_laquesea.sin_port=htons(entero n° de puerto);
```

Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2013

Documentación de funciones y estructuras POSIX utilizables:

```
int pthread_create(pthread_t* id,pthread_attr* attr,void *(*fn)(void*),void* arg);
int pthread_cancel(pthread_t thread);
void* pthread_join(pthread_t thread);

int sem_init(sem_t *sem, int pshared, unsigned int value); // Usar pshared=0
int sem_wait(sem_t* sem);
int sem_post(sem_t* sem);
int sem_destroy(sem_t* sem);

int pthread_mutex_init(pthread_mutex_t* mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t* mutex);
int pthread_mutex_unlock(pthread_mutex_t* mutex);
int pthread_mutex_destroy(pthread_mutex_t* mutex);

int pthread_cond_init(pthread_cond_t* cond, const pthread_condattr_t *attr);
int pthread_cond_wait(pthread_cond_t* cond,pthread_mutex_t* mutex);
int pthread_cond_broadcast(pthread_cond_t* cond);
int pthread_cond_destroy(pthread_cond_t* cond);

int sigemptyset(sigset_t* set);
int sigaddset(sigset_t* set,int signo);
int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset);
                    /*     how=SIG_BLOCK, SIG_SETMASK ó SIG_UNBLOCK */
int sigaction(int sig, const struct sigaction *act,struct sigaction *oact);
int sigwait(sigset_t* set,int* sig);

struct sigaction
{
       void (*sa_sigaction)(int signo, siginfo_t *info, void *dontuse);
       sigset_t sa_mask;
       int sa_flags; /* Poner bit SA_SIGINFO */
};

int timer_create(clockid_t clockid,struct sigevent* sig,timer_t* id);
                                 // Usar clockid= CLOCK_MONOTONIC y sig=NULL (señal
                                 // por defecto SIGALRM)
int timer_settime(timer_t id,int flags,const struct itimerspec* ival,
                  struct itimerspec* oval);   // Usar flags=0, oval=NULL
int timer_delete(timer_t id);

struct timespec
{
       time_t tv_sec;      // Segundos
       long tv_nsec;// Nanosegundos
} ;

struct itimerspec
{
       struct timespec it_interval;     // Periodo de temporización
       struct timespec it_value;        // Primera activación
} ;

mdq_t mq_open(char* nombre,int apert,mode_t modo,mq_attr* attr);
       // 'apert' puede ser combinación con | de: O_CREAT, O_WRONLY, O_RDONLY. Si
       'apert' contiene O_CREAT, hay que añadir los parámetros modo=0 y attr, sino no
       se ponen.

mdq_t mq_open(char* nombre,int apert,mode_t modo,mq_attr* attr);
int mq_send(mdq_t idcola,char* datos,int nbytes,int prioridad);
int mq_receive(mdq_t idcola,char* datos,int nbytes,int* prioridad);
int mq_unlink(mdq_t idcola);

typedef struct mq_attr {
       long mq_flags; /*usar 0 */
       long mq_maxmsg; /*número máximo de mensajes*/
       long mq_msgsize; /*tamaño máximo del mensaje*/
       long mq_curmsgs; /*actual número de mensajes*/
} mq_attr_t;
```