Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2012

4 puntos 1) Escribir un programa que lea un valor de temperatura del canal A/D 4 cada 50 ms, y calcule y escriba por pantalla la media de los 5 valores más recientes. Las condiciones de funcionamiento son las siguientes:

- Se dispone en una librería 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 de resolución: resultado de 0 a 2^nBits-1:
int ConversionAD(int nCanal,int nBits);
```

- El conversor A/D tiene un rango de entrada de 0-5V. Se desea utilizar conversiones A/D de 12 bits.
- La medida de temperatura se obtiene a partir de un termopar, cuya salida es lineal entre 1V para T=20ºC y 5V para T=100ºC.
- Tras el inicio de adquisiciones, el programa main() se queda a la espera de un carácter por teclado ('a' o 'b'), que permita establecer el algoritmo de cálculo de media (por defecto 'a'):
  a. Media aritmética: $m_k = \frac{t_k + t_{k-1} + t_{k-2} + t_{k-3} + t_{k-4}}{5}$
  b. Media con pesos: $m_k = \frac{5.t_k + 4.t_{k-1} + 3.t_{k-2} + 2.t_{k-3} + 1.t_{k-4}}{15}$
- Se debe implementar y utilizar la misma función para calcular la media en los 2 casos:

```
float ProductoEscalar(const float* t1,const float* t2,int n);
```

.

- Otro hilo del mismo proceso puede utilizar los resultados (media calculada y tipo de cálculo utilizado): utilizar mútex para proteger el acceso a estas variables, y variable de condición para despertar al posible hilo en espera cada vez que se calcule la media.

2 puntos 2) Para el programa del ejercicio 1, realizar un segundo hilo que:

- Cree un socket cliente (dirección IP local 192.168.2.5, puerto local 9500), que se conecte a un servidor que se encuentra esperando en la dirección IP 192.168.2.100, puerto 10500.
- Cada vez que se reciba el aviso de la condición, envíe por la conexión de red el texto:

  `media=`*`valor calculado`* `, modo=`*`a ó b`*

  (utilizar `sprintf` para generar la cadena a enviar).
- Tras cada envío, espere la recepción de una respuesta por la conexión de red, que podrá ser el texto "ACK" ó el texto "CAMBIO". En el 2º caso, el hilo debe cambiar el tipo de cálculo (a ó b) a utilizar en las siguientes ocasiones.
- Escribir el código que permitiría lanzar este segundo hilo.

Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2012

3) *2 puntos*

Escribir una función que, dados como parámetros el nombre de un archivo de texto (que contiene una matriz de reales), y el tamaño en filas y columnas de dicha matriz, lea del archivo y devuelva en una variable tipo estructura el contenido de la matriz.

Condiciones:

- Se debe leer cada línea del archivo (función `fgets`), y comprobar que el nº de valores que contiene es igual al nº de columnas recibido como parámetro (utilizar `strtod`). En caso contrario, se devolverá matriz inválida (filas=0, colus=0, datos=NULL).
- Tras leer todas las líneas, se debe comprobar que el nº de líneas leídas es igual al nº de filas recibido como parámetro. En caso contrario, se devolverá matriz inválida (filas=0, colus=0, datos=NULL).
- Escribir un ejemplo de llamada a la función.

4) *2 puntos*

Contestar **razonadamente** a las siguientes preguntas:

a) Indicar si las siguientes instrucciones provocan un funcionamiento de tipo síncrono / asíncrono / ninguno de ambos:

- `fscanf(fid,"%d",&x);`
- `accept(sock,(struct sockaddr_in*) &rem,sizeof(struct sockaddr_in));`
- `InitTemporizador(50,FnCallBack); // Función del ejercicio 1`
- `typedef FILE* asinc;`

b) ¿Qué es una excepción? ¿Cómo se puede tratar en un programa POSIX bajo C?

Control de Procesos en Tiempo Real

Examen Ordinario - Enero 2012

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,SOCK_STREAM,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 2012

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
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;
```