

# Control de Procesos en Tiempo Real

## Examen Ordinario - Febrero 2010

---

1)  
4 puntos

Escribir un programa que recibe 5 parámetros en línea de comandos:

- Nombre de directorio
- Nombre de archivo 1
- Nombre de archivo 2
- Dos números enteros (m y n)

El programa debe:

- Abrir en modo texto los archivos cuyos nombres están formado por el directorio + archivo (1 o 2) + “.dat”.
- Leer de dichos archivos sendas matrices ( $M_1$  y  $M_2$ ) de valores enteros de dimensiones  $m \times n$ .
- Realizar el cálculo  $M_1 \times M_2^t$ . (resultado: matriz cuadrada  $m \times m$ ).
- Obtener la norma (suma de cuadrados) del vector formado por la diagonal principal de la matriz producto.
- Escribir el resultado obtenido.

Por ejemplo, la llamada al programa con la orden:

```
MIPROG C:\USER\EXAMEN\ MATRIZA MATRIZB 3 5
```

provocará la lectura de dos matrices de  $3 \times 5$  enteros de los archivos:

```
C:\USER\EXAMEN\MATRIZA.DAT y  
C:\USER\EXAMEN\MATRIZB.DAT
```

Tras multiplicar  $M_A \times M_B^t$ , se obtendría una matriz producto  $3 \times 3$ :

$$P = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix}$$

y el resultado a escribir sería el del cálculo:  $norm = p_{00}^2 + p_{11}^2 + p_{22}^2$

Condiciones:

- No se puede suponer un tamaño máximo para las matrices, ni para los nombres de directorio y de archivo.
- Escribir como funciones aquellas partes de código que se considere serían reutilizables en otros problemas de cálculo matricial.
- Recordar liberar la memoria asignada.

# Control de Procesos en Tiempo Real

## Examen Ordinario - Febrero 2010

---

2)  
4 puntos

En un Sistema de T.R. se desea crear un hilo que, cada “Tm” ms, lea de un canal A/D un nuevo valor de velocidad de un motor, lo convierta a r.p.m. según la regla que se indica a continuación, y envíe a otra aplicación un mensaje mediante socket UDP/IP si la media de los 5 valores más recientes está entre 8000 y 9000 r.p.m, y además la media de los 5 anteriores es menor a 5000 r.p.m.

Para ello, se dispone en la librería “ConvAD.lib” de la función siguiente, cuyo prototipo está declarado en “ConvAD.h” (ambos archivos se encuentran en el directorio C:\EXAMEN):

```
int LeeAD(int* datos,int nCanal); // Lee 1 valor del conversor A/D
// indicado en 'nCanal' y lo guarda
// en la dirección 'datos'. El valor
// devuelto es una digitalización en
// 10 bits (valores 0 a 1023) de la
// tensión en el canal A/D según la
// la relación lineal:
// valor 0 (leído) = min = -10V,
// valor 1023 (leído) = max = +10V
```

Condiciones:

- El hilo recibe como parámetros en la inicialización (ver doc. en última página), el periodo de conversión Tm y el nº de canal A/D (utilizar una estructura).
- El hilo debe activar periódicamente la lectura A/D mediante una temporización cada Tm milisegundos.
- La velocidad del motor se mide con un tacómetro lineal con relación 1000 rpm/V.
- La secuencia valores<sub>k</sub> necesaria para los cálculos puede ser utilizada por otros hilos, por lo que se debe proteger el acceso compartido a la misma.
- Escribir la parte de código que lanzaría a este hilo.
- El mensaje UDP/IP se deberá enviar a una aplicación conectada al puerto 5000 de la dirección IP 66.77.88.99, y contendrá un texto como:  

```
ALARM ACTUAL=media_5_ultimos ANTERIOR=media_5_anteriores
```
- Indicar los pasos que se realizarían en Visual C++ para incluir en el proyecto la librería necesaria, y poder compilar y enlazar sin errores.
- ¿La temporización utilizada es síncrona o asíncrona? Explicar su diferencia.

3)  
2 puntos

Contestar **razonadamente** a las siguientes preguntas:

- a) ¿Qué es una función ‘callback’? ¿En qué casos se utiliza? ¿La utilizaría para E/S de un stream con funciones de propósito general (fopen, fprintf, ...)? Razonar la respuesta.
- b) ¿Qué combinación hardware/software debería ser utilizadas para un sistema de T.R. embebido en un dispositivo del que se espera vender una gran cantidad de unidades (consumo masivo), y con necesidades de procesamiento reducidas? Razonar la respuesta.
  - DSP con Sistema Operativo de Tiempo Real.
  - Microcontrolador con Sistema Operativo de Tiempo Real.
  - PC industrial con Windows.
  - Microcontrolador con selección de tareas por Function Queue Scheduling.

# Control de Procesos en Tiempo Real

## Examen Ordinario - Febrero 2010

---

### 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 2º, etc.
```

### Algunas funciones de cadenas de caracteres:

```
int atoi(const char* cadena);
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);
char* strchr(const char* cad, char c);
char* strstr(const char* cad, const char* busca);
```

### 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, 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 nº de puerto);
```

# Control de Procesos en Tiempo Real

## Examen Ordinario - Febrero 2010

---

### Documentación de funciones y estructuras POSIX utilizables:

```
int pthread_create(pthread_t* id, pthread_attr* attr, void *(*fn)(void*), void* arg);
    // arg es un puntero al parámetro que se quiere pasar al hilo
    // usar attr=NULL

int pthread_cancel(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);
    // Usar attr=NULL
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);
    // Usar attr=NULL
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
};

mqd_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.

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