# Control de Procesos en Tiempo Real

## Examen Ordinario - Junio 2010

1) *4 puntos*

Escribir un programa que recibe 4 parámetros en línea de comandos:

- Nombre de directorio
- Nombre base de archivo
- Dos números enteros 'n' (entre 1 y 9) y 'm'.

El programa debe:

- Si no se cumple 1<=n<=9, escribir un mensaje de error y terminar.
- Crear una matriz de reales con 'n' filas y 'm' columnas.
- Abrir en **modo binario** cada uno de los archivos cuyos nombres están formado por el directorio + archivo + índice (1 a n) + ".dat".
- Leer en **modo binario** de cada uno de los archivos una tabla de 'm' floats sobre la fila i-ésima de la matriz.
- Realizar el calculo $M \times M^t$. (resultado: matriz cuadrada n × n).
- Llamar a la función Determinante(…) con la matriz producto. No es necesario realizar esta función, sólo indicar su prototipo.
- Escribir el resultado obtenido.

Por ejemplo, la llamada al programa con la orden:

```
MIPROG  C:\EXAMEN\  TABLA 3  5
```

provocará la lectura de una matriz de n=3 × m=5 floats, donde:

La fila M[0] contendrá los 5 datos leídos del archivo `C:\EXAMEN\TABLA1.DAT`
La fila M[1] contendrá los 5 datos leídos del archivo `C:\EXAMEN\TABLA2.DAT`
La fila M[2] contendrá los 5 datos leídos del archivo `C:\EXAMEN\TABLA3.DAT`

Tras multiplicar $M \times M^t$, se obtendría una matriz producto 3 × 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: det=Determinante(P,n);

Condiciones:

- No se puede suponer un tamaño máximo para las tablas, 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.
- Usar `sprintf(char* dest,const char* fmt,…);` para crear los nombres de archivo.
- Recordar liberar la memoria asignada.

# Control de Procesos en Tiempo Real

## Examen Ordinario - Junio 2010

2) *4 puntos*

En un Sistema de T.R. se precisan dos hilos:

El primer hilo debe leer cada "Tm" ms de un canal A/D un nuevo valor de velocidad de un motor, convertirlo a r.p.m. según la regla que se indica a continuación, y avisar mediante una variable de condición al segundo hilo 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. (por tanto, es necesaria una tabla $y_k$ que guarde los 10 valores más recientes).

El segundo hilo esperará por la variable de condición, y cuando se produzca escribirá en pantalla un mensaje de alarma con ambas medias, volviendo a esperar otra vez.

Para la conversión A/D, se dispone en la librería "ConvAD.lib" de la función siguiente, cuyo prototipo está declarado en "ConvAD.h" (ambos se encuentran en el directorio C:\EXAMEN):

```
void LeeAD(int* datos,int nCanal);  // Lee un 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 (leido) = min = -10V,
                                    // valor 1023 (leido) = max = +10V
```

Condiciones:

- Los valores Tm y nº de canal son solicitados por consola en el hilo principal.
- La velocidad del motor se mide con un tacómetro lineal con relación 1000 rpm/V.
- La secuencia $y_k$ 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 ambos hilos.
- 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) Poner un ejemplo sencillo en que se utilizaría una función callback en una operación de E/S, y otro en que no se puede utilizar. ¿Cuál es la diferencia en el funcionamiento de ambos casos?

b) Describir brevemente los pasos necesarios para programar un socket servidor TCP/IP.

c) Indicar el error en el siguiente trozo de código, y corregirlo:

```
pthread_mutex_t *mutex;
pthread_mutex_init(mutex,NULL);
```

d) Indicar el error en el siguiente trozo de código, y corregirlo:

```
char *txt;
strcpy(txt,"EXAMEN");
```

# Control de Procesos en Tiempo Real

## Examen Ordinario - Junio 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 - Junio 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
} ;

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;
```