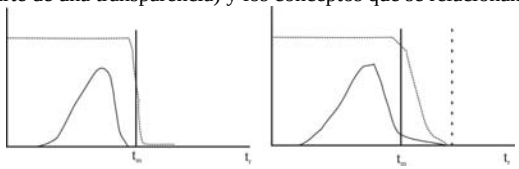
Examen de Informática Industrial
4º GITI (23/1/14)

CUESTIONES. TIEMPO: 1 HORA 30 MINUTOS (VALORACIÓN: 50%).

1. Defina los conceptos a) Tiempo de respuesta b) Coste de ejecución c) Meta o "deadline". Explique la figura adjunta (parte de una transparencia) y los conceptos que se relacionan directamente con ella.



2. Explique las particularidades del tratamiento de señales POSIX cuando existen varios hilos en un proceso.
3. Para el siguiente programa explique **en general cómo funciona**, cuánto **tarda** en ejecutarse y qué **mensajes** imprime por pantalla, suponiendo que el proceso que comienza en **main** puede recibir de otros procesos señales de tiempo real SIGRTMIN acompañadas de datos con valores entre 0 y M-1. En particular, explique qué sucede cuando se ejecuta como **c3 1 4 2 3** y recibe señales SIGRTMIN con datos 1, 3, 1 y 0 a los 0.5, 2., 3. y 3.5 segundos del comienzo respectivamente.

```
<cabeceras correctas>
#define M 5
int m = 0;
pthread_mutex_t x =
     PTHREAD_MUTEX_INITIALIZER;
void *f(void *p) {
  int a = (int)p; struct timespec t;
  printf("a vale %d\n", a);
  t.tv_sec = a; t.tv_nsec = 0;
  nanosleep(&t, NULL);
  pthread_mutex_lock(&x);
  m = m-1;
  pthread_mutex_unlock(&x);
  kill(getpid(), SIGUSR1);
}
void f1(int a, siginfo_t *b, void *c) {}
int main(int b1, char **b2) {
  sigset_t a; siginfo_t c; struct sigaction d;
  int i, j, k, n; int v[M]; int y[M]; pthread_t h[M];
  for(k=0; k<M; k++) {
    y[k] = 0; v[k] = 0;
    if(k<b1-1) sscanf(b2[k+1], "%d", &v[k]);
  }
  d.sa_flags = SA_SIGINFO; d.sa_sigaction = f1;
  sigemptyset(&d.sa_mask);
  sigaction(SIGRTMIN, &d, NULL);
  sigemptyset(&a); sigaddset(&a, SIGRTMIN);
  sigaddset(&a, SIGUSR1);
  pthread_sigmask(SIG_BLOCK, &a, NULL);
  do {
    j = sigwaitinfo(&a, &c);
    if(j == SIGRTMIN) {
      i = c.si_value.sival_int;
      if(y[i] == 0) {
        pthread_create(&h[i], NULL, f,
                       (void *)v[i]);
        y[i] = 1;
      } else {
        pthread_cancel(h[i]);
        pthread_join(h[i], NULL);
        y[i] = 0;
      }
      pthread_mutex_lock(&x);
      if(y[i] == 1) m++; else m--;
      pthread_mutex_unlock(&x);
    }
    pthread_mutex_lock(&x);
    n = m;
    pthread_mutex_unlock(&x);
    printf("n: %d\n", n);
  } while(n != 0);
  return 0;
}
```

4. Explique cómo funcionan este fragmento de programa en ADA, y cómo podría conseguirse una funcionalidad lo más equivalente posible utilizando "sockets" POSIX (**no se pide un programa**, sólo una explicación de cómo se haría).

```
select
    accept entrada1(I: in integer) do
      <Sentencias A>
    end entrada1;
or
   delay 5.0;
   <sentencias B>
end select;
```

5. Para el siguiente programa explique **en general cómo funciona**, cuánto **tarda** en ejecutarse y qué **imprime por pantalla**, suponiendo que no se produce ningún error. En particular, explique qué sucede cuando **a)** se ejecuta el programa como **c5 10000 5**.

```
<cabeceras correctas>
#define N 4
void f1(struct sockaddr_in c, int i) {
  int s; int n;
  struct timespec b = {0, 200000000L};
  c.sin_port = htons(i+1);
  s = socket(AF_INET, SOCK_DGRAM, 0);
  bind(s, (struct sockaddr *)&c, sizeof(c));
  do {
    recv(s, (void *)&n, sizeof(n),
            MSG_WAITALL);
    printf("%d con %d\n", i, n);
    if(n >= 0) n++;
    nanosleep(&b, NULL);
    c.sin_port = htons(i);
    sendto(s, (void *)&n, sizeof(n), 0,
           (struct sockaddr *)&c, sizeof(c));
  } while(n > 0);
  exit(0);
}
int main(int a1, char **a2) {
  pid_t b[N]; int i, d, e, f, g;
  struct sockaddr_in c;
  sscanf(a2[1], "%d", &d);
  sscanf(a2[2], "%d", &e);
  g = socket(AF_INET, SOCK_DGRAM, 0);
  memset((char *)&c, 0, sizeof(c));
  c.sin_family = AF_INET;
  c.sin_port = htons(d);
  c.sin_addr.s_addr = INADDR_ANY;
  bind(g, (struct sockaddr *)&c, sizeof(c));
  for(i=0; i<N; i++) {
    b[i] = fork(); if(b[i] == 0) f1(c, d+i);
  }
  sleep(1);
  c.sin_port = htons(d+N); f = 0;
  do {
    sendto(g, (void *)&f, sizeof(f), 0,
           (struct sockaddr *)&c, sizeof(c));
    recv(g, (void *)&f, sizeof(f), MSG_WAITALL);
    printf("f: %d\n", f);
  } while(f < e);
  f = -1;
  sendto(g, (void *)&f, sizeof(f), 0,
           (struct sockaddr *)&c, sizeof(c));
  waitpid(b[0], &i, 0);
  return 0;
}
```

Datos que pueden ser útiles:

```
int sigaction(int sig, struct sigaction *act, struct sigaction *oact);
pid_t fork(void); int execl(const char *ejecutable, const char *arg0, ...., NULL);
int kill(pid_t pid, int sig); int sigqueue(pid_t pid, int sig, const union sigval val);
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(* rut_com)(void *), void *arg);
int pthread_join(pthread_t thread, void **valor); int pthread_cancel(pthread_t thread);
int sigwaitinfo(const sigset_t *estas_sg, siginfo_t *infop);
int sscanf(char  *fuente, char *formato, …);  int sprintf(char *destino, char *formato,…);
int pthread_sigmask(int how, const sigset_t *set, sigset_t *oset); int sigemptyset(sigset_t *pset);
int sigfillset(sigset_t *pset); int sigaddset(sigset_t *pset, int sig); int sigdelset(sigset_t *pset, int sig);
struct sigaction {
  void(* sa_handler) ();
  void (* sa_sigaction)
          (int numsen,
          siginfo_t *datos,
          void *extra);
  sigset_t sa_mask;
  int sa_flags;
};
typedef struct {
  int si_signo;
  int si_code;
  union sigval si_value;
} siginfo_t;
union sigval {
  int sival_int;
  void *sival_ptr;
};
struct timespec {
  time_t tv_sec;
  long tv_nsec;
};
struct sockaddr_in {
  sa_family_t    sin_family;
  in_port_t      sin_port;
  struct in_addr sin_addr;
}
int nanosleep(struct timespec *retraso, struct timespec *queda);
pid_t getpid(void); pid_t getppid(void); pid_t wait(int *estado); pid_t waitpid(pid_t, int *estado, int options);
int pthread_mutex_lock(pthread_mutex_t *mutex); int pthread_mutex_unlock(pthread_mutex_t *mutex);
int socket(int dominio, int tipo, int protocolo); ssize_t recv(int id_socket, void *buffer, size_t tam, int flagsr);
int bind(int socket, const struct sockaddr *direccion, socklen_t long_dir);
ssize_t sendto(int id_socket, void *pdatos, size_t tam, int flagss, struct sockaddr *destino, socklen_t tam_dest);
uint32_t htonl(uint32_t local); uint16_t htons(uint16_t local);
int listen(int socket, int tam_cola); int accept(int socket, struct sockaddr *direccion, socklen_t *lon_dir);
int connect(int socket, const struct sockaddr *direccion, socklen_t tam);
```

Examen de Informática Industrial
4º GITI (23/1/14)

PROBLEMA. TIEMPO: 1 HORA 30 MINUTOS (VALORACIÓN: 50%)

Resumen: Se pide realizar en C utilizando las normas POSIX el programa **medidas** que recibe medidas de un conjunto de **NS** sensores de temperatura distribuidos por medio de **"sockets" no orientados a conexión** y genera alarmas cuando se superan los límites establecidos. El programa se realizará dedicando **un hilo a la recepción de cada medida y otro a la generación de alarmas** (pueden crearse otros adicionales si se considera necesario). Los macros, funciones y tipos mencionados **se suponen definidos** en la cabecera **medidas.h**, que se supone disponible (sólo hay que incluirla).

Especificación detallada:

- **Argumentos de la línea de comandos**:
  - **Argumento 1:** Define **npuerto** (entero codificado en caracteres; número de puerto UDP).
  - **Argumento 2:** Define **Tmax** (número en punto flotante codificado en caracteres; temperatura máxima).
  - **Argumento 3:** Define **ciclo_alarma** (entero codificado en caracteres; ciclo de alarma en milisegundos; menor que 1000).
- **Datos compartidos:** Los datos compartidos consistirán **al menos** en una **tabla de valores de temperatura** medidos en cada sensor y una **cuenta** de los sensores que superan el límite **Tmax**.
- **Situación de alarma:** Se considera que existe una situación de alarma cuando al menos **NS/4** sensores han superado la temperatura **Tmax**.
- **Hilos de recepción de medidas:** Cada hilo de recepción de medidas recibe datos de un "socket" **no orientado a conexión** con número de puerto UDP igual a **npuerto+i**, siendo **i** el índice de sensor asociado al hilo (entero entre entre **0** y **NS-1**). La temperatura se recibe por el "socket" como un número en punto flotante (**float**). Cada vez que recibe una medida cada hilo de recepción (**además** de otras posibles operaciones necesarias para resolver el problema) actualiza las variables compartidas y comprueba si existe o no una situación de alarma.
- **Hilo de generación de alarmas:** Espera a que exista una situación de alarma, y cuando esto sucede comienza un ciclo de aviso en el cual activa una señal sonora durante **TS** ms cada **ciclo_alarma** milisegundos. La alarma se desactiva cuando no existen sensores con temperatura superior a **Tmax**, y entonces el hilo pasa a esperar una nueva situación de alarma. Para activar la señal sonora se dispone de la función **beep** (ver cabecera). Para generar el ciclo se utilizará un **temporizador POSIX 1003.1b**.
- **Condición de fin:** El proceso acaba cuando se recibe una señal SIGTERM. Antes de acabar debe desactivar el sonido para evitar que se quede activado continuamente.

Otras condiciones y observaciones:

- El ciclo de alarma (**ciclo_alarma**) debe realizarse con un **temporizador POSIX 1003.1b**.
- El programa deberá funcionar **independientemente** de los valores concretos de los argumentos y las constantes simbólicas.
- Se supone que los datos se reciben por los "sockets" en el orden correcto para el computador en que se ejecuta el programa.
- Es necesario utilizar mutex y variables de condición cuando sean necesarios para gestionar las variables compartidas, según lo indicado en clase.
- No es necesario considerar tratamiento de errores. Puede suponerse que los argumentos de la línea de comandos son correctos, y que todas las funciones son "pthread-safe".
- Es preciso acompañar el programa de pseudocódigo o explicación de su funcionamiento.

Fichero de cabecera medidas.h:

```
/* Cabecera medidas.h */

/* Número de sensores */
#define NS 10

/* Duración del tono (ms) */
#define TS 200

/* Función para activar sonido: on = 1, sonido. on = 0, silencio */
void beep(int on);
```