Examen de Sistemas Informáticos en Tiempo Real
1º IAEI (5/7/13)

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

Advertencia: Se piden sobre todo **conceptos**, más que ejemplos concretos de programación, salvo que se pida expresamente.

1. Explique con ayuda de un esquema en qué consiste el direccionamiento virtual y paginado, y qué hay que hacer para que dos **procesos** (no hilos) puedan acceder a variables compartidas.
2. Para el siguiente programa explique **en general** cómo **funciona**, cuánto **tarda** en ejecutarse y qué **mensajes** imprime por pantalla, suponiendo el proceso que comienza en **main** recibe señales de tiempo real acompañadas de datos. En particular, explique qué sucede cuando dicho proceso se invoca como **c2 1 2 3 4** y recibe señales SIGRTMIN+1, SIGRTMIN+2, SIGRTMIN+3, SIGRTMIN+1, SIGRTMIN+3, SIGRTMIN+2 y SIGRTMIN con datos 1, 0, 0, 4, 3, 2 y 0 por ese orden, la primera a los 100 ms de arrancar y el resto separadas por 100 ms. La cola "**/c1**" existe, está inicialmente vacía y su capacidad es suficiente como para no llenarse en ningún momento.

```
<cabeceras correctas>
#define N1 6
char e[N1] = { 'a', 'n', ' ', 'd', 'o', 'g' };
void f(int a, siginfo_t *b, void *c) { }
void f1(int i) {
  sigset_t s; siginfo_t d; int a, j;  mqd_t c1;
  sigfillset(&s);
  c1  = mq_open("/c1", O_NONBLOCK |
                       O_WRONLY, 0, NULL);
  do {
    a = sigwaitinfo(&s, &d);
    j = d.si_value.sival_int;
    printf("%d: %d, %c\n", i, j, e[j]);
    if(a != SIGRTMIN && j >=0 && j < N1)
      mq_send(c1, &e[j], 1, i);
  } while(a != SIGRTMIN);
  exit(0);
}
int main(int a, char **b) {
  sigset_t s1; struct sigaction c; siginfo_t d;
  int i, n; pid_t *p; mqd_t c1; char g;
  sigemptyset(&s1);
  c.sa_flags = SA_SIGINFO;
  sigemptyset(&c.sa_mask); c.sa_sigaction = f;
  for(i=0; i<=a; i++) {
    sigaddset(&s1, SIGRTMIN+i);
    sigaction(SIGRTMIN+i, &c, NULL);
  }
  sigprocmask(SIG_BLOCK, &s1, NULL);
  p = malloc(sizeof(pid_t)*a);
  for(i=1; i<a; i++) {
      p[i] = fork(); if(p[i] == 0) f1(i);
  }
  do {
    n = sigwaitinfo(&s1, &d);
    if(n == SIGRTMIN) {
      for(i=1; i<a; i++)
        sigqueue(p[i], SIGRTMIN, d.si_value);
    }
    else sigqueue(p[n-SIGRTMIN], n, d.si_value);
  } while(n!= SIGRTMIN);
  for(i=1; i<a; i++) wait(&l);
  c1  = mq_open("/c1", O_NONBLOCK |
                       O_RDONLY, 0, NULL);
  while(mq_receive(c1, &g, 1, NULL) != -1)
    printf("%c", g);
  printf("\n"); return 0;
}
```

3. Explique los diferentes modelos de sincronización utilizados para paso de mensajes, cual de ellos utilizan las colas POSIX y cómo pueden utilizarse estas para emular los otros modelos.
4. Explique qué es la inversión de prioridad, y alguno de los procedimientos que permiten limitar su duración. Dibuje dos cronograma para el siguiente ejemplo, uno sin aplicar y el otro aplicando el procedimiento para limitar la inversión de prioridad. Indique sobre ambos cuando existe inversión de prioridad. En el ejemplo la prioridad con número más alto es la más alta.
   - **Proceso A**, con prioridad 3. Se activa en t=15. Fase de cálculo con un coste de 5, **acceso al recurso R1** en una sección crítica de coste 7, y segunda fase de cálculo de coste 15.
   - **Proceso B**, con prioridad 2. Se activa en t=7, tiene un coste de 27 y no accede a recursos compartidos.
   - **Proceso C**, con prioridad 1. Se activa en t=0. Primera fase de cálculo con coste 5, **acceso al recurso R1** en una sección crítica de coste 5, y finalmente ejecuta una segunda fase de cálculo con coste 5.

5. Para el siguiente programa explique **en general** cómo **funciona**, cuánto **tarda** en ejecutarse y qué **mensajes** imprime por pantalla si el proceso que comienza en **main** recibe señales entre SIGRTMIN y SIGRTMIN+N-1. En particular explique qué sucede cuando el proceso que comienza en **main** recibe las señales SIGRTMIN+2, SIGRTMIN+1 y SIGRTMIN+3 con datos 200, 250 y 500 por ese orden, la primera a los 100 ms de comenzar y el resto separadas 100 ms entre sí.

```
<cabeceras correctas>
#define N 4
timer_t  t;
struct itimerspec r1; struct itimerspec r2;
void h(int a, siginfo_t *b, void *c) {}
void *f(void *p) {
  int i = (int)p; int n; sigset_t s; siginfo_t d;
  struct timespec t1 = {0, 0};
  sigemptyset(&s); sigaddset(&s, SIGRTMIN+i);
  while(1) {
    n = sigwaitinfo(&s, &d);
    printf("%d; %d con %d\n",
            i, n-SIGRTMIN, d.si_value.sival_int);
    t1.tv_nsec = d.si_value.sival_int*1000000;
    nanosleep(&t1, NULL);
    timer_settime(t, 0, &r2, NULL);
    timer_settime(t, 0, &r1, NULL);
  }
}
int main(int a, char **b) {
  struct timespec c1 = {0, 500000000L}, c2 = {0, 0};
  siginfo_t d; sigset_t s; int i; struct sigaction c;
  c.sa_flags = SA_SIGINFO; c.sa_sigaction = h;
  sigemptyset(&c.sa_mask); sigemptyset(&s);
  for(i=0; i<N; i++) {
    sigaction(SIGRTMIN+i, &c, NULL);
    sigaddset(&s, SIGRTMIN+i);
  }
  sigaddset(&s, SIGALRM);
  sigprocmask(SIG_BLOCK, &s, NULL);
  r1.it_value = c1; r1.it_interval = c2;
  r2.it_value = c2; r2.it_interval = c2;
  timer_create(CLOCK_REALTIME, NULL, &t);
  timer_settime(t, 0, &r1, NULL);
  for(i=0; i<N; i++)
    pthread_create(NULL, NULL, f, (void *)i);
  sigemptyset(&s); sigaddset(&s, SIGALRM);
  sigwaitinfo(&s, &d);
}
```

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);
void *malloc(size_t siz); int sscanf(char *origen, char *formato, …);
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_sigmask(int how, const sigset_t *set, sigset_t *oset);
int sigwaitinfo(const sigset_t *estas_sg, siginfo_t *infop);
int mq_send(mqd_t cola, const char *datos, size_t longitud, unsigned int prioridad);
int mq_getattr(mqd_t cola, struct mq_attr *atributos);
int sscanf(char  *fuente, char *formato, …);  int sprintf(char *destino, char *formato,…);
int sigemptyset(sigset_t *pset); int sigfillset(sigset_t *pset);
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 itimerspec {
        struct timespec it_value;
        struct timespec it_interval;
      };
int sigaddset(sigset_t *pset, int sig); int sigdelset(sigset_t *pset, int sig);
int sigprocmask(int how, const sigset_t *set, siset *oset);  int pthread_setcanceltype(int type, int *oldtype);
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);
mqd_t mq_open(const char *mq_name, int oflag, mode_t modo, struct mq_attr *atributos);
int mq_receive(mqd_t cola, const char *datos, size_t longitud, unsigned int *prioridad);
int mq_close(mqd_t cola); int mq_unlink(const char *nombre);
Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_APPEND, O_TRUNC, O_NONBLOCK
```

Examen de Sistemas Informáticos en Tiempo Real
1º IAEI (5/7/13)

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

Resumen: Se pide realizar en C y con llamadas POSIX un programa que demodula NCANAL señales PWM y envía las medidas a una cola de mensajes. Las constantes simbólicas y declaraciones necesarias están definidas en la cabecera **pwm.h**, que se supone disponible.

Especificación detallada:

- **Argumentos de la línea de comandos**: Los argumentos de la línea de comandos se interpretan como **T** (argumento 1), **Tmin** (argumento 2), **Tmax** (argumento 3) y **escala** (argumento 4). Los tres primeros son valores enteros positivos menores que 1000 codificados en caracteres y expresan tiempo en **milisegundos**. El valor **escala** es un número fraccionario positivo.
- **Demodulacion de un canal PWM:** En cada canal se reciben pulsos de nivel alto con tiempo de duración variable **Ta** (ver figura), que se encuentra entre los valores extremos **Tmin** y **Tmax**. Si el canal está activo, comienza un pulso cada **T** milisegundos. El comienzo del pulso del canal i se puede detectar como la llegada de una señal SIGRTMIN+i con un dato 1, y el final como la llegada de la misma señal con dato 0. Por tanto el tiempo transcurrido entre las dos señales, **Ta**, es la duración del pulso; puede suponerse que las señales llegan siempre en el orden correcto. **Ta** representa un valor **medida** (número en punto flotante) que puede calcularse como

  medida = (Ta - Tmin)*ESCALA

  si el tiempo **Ta** medido es mayor que **Tmax** o menor que **Tmin**, se toma respectivamente **Tmax** y **Tmin** para el cálculo de **medida**. Todos los tiempos se miden en **milisegundos**.
- **Funcionamiento del programa:** El programa crea al comenzar su ejecución la cola **/cola_medida** con capacidad para 5 mensajes; cada mensaje contiene una estructura **struct medidas** (ver cabecera **pwm.h**). Cada **T** milisegundos debe enviar un mensaje por la cola con las últimas medidas disponibles en ese momento para cada canal; el primer envío se realizará **T** milisegundos después de arrancar el programa.
- **Detección de señales inactivas:** Si al llegar el momento de enviar un mensaje por la cola existe algún canal para el que no se ha obtenido ninguna medida nueva desde el ciclo anterior, se marca como inactivo en la estructura de medida y se fija la entrada **canal_activo** de su estructura **medida** a 0; en caso contrario se marca el canal como activo (valor 1). Si el canal se señala como inactivo la medida que se envíe con el mensaje no es válida, por lo que puede tener un valor arbitrario.
- **Condición de fin:** El programa acabará si al enviar un mensaje por la cola no hay ningún canal activo. El programa cerrará y destruirá la cola antes de acabar.

Otras condiciones:

- Las constantes simbólicas están en la cabecera **pwm.h**.
- El ciclo de **T** milisegundos debe realizarse con un **temporizador POSIX 1003.1b**.
- El programa deberá funcionar **independientemente** de los valores concretos de los argumentos y las constantes simbólicas.
- No es necesario considerar tratamiento de errores.
- Es preciso acompañar el programa de **pseudocódigo o explicación de su funcionamiento**.

Fichero de cabecera:

```
/* Cabecera pwm.h */

/* Definición de constantes */

#define NCANAL 8    /* Numero de canales */

/* Estructura de medidas (mensaje para la cola) */

struct medidas
{
    int canal_activo[NCANAL];      /* 0: activo; 1: inactivo */
    float medida_canal[NCANAL]; /* Medidas de cada canal */
};
```