Examen de Sistemas Informáticos en Tiempo Real
IAEI-II (3/12/10)

CUESTIONES. TIEMPO: 1 HORA 15 MINUTOS (VALORACIÓN: 50%).

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

1. Para el siguiente programa explique **en general** cómo funciona, qué mensajes imprime y qué debe suceder para que su comportamiento esté definido, suponiendo que sólo se reciben señales **SIGRTMIN**. En particular, explique qué sucede y cuanto tardan en ejecutarse todos los procesos resultantes cuando el programa se invoca desde el intérprete de comandos como "**prog1 1 12**", siendo "**prog1**" el nombre del ejecutable, y todos los procesos creados reciben una señal **SIGRTMIN** cada 200 ms con dato 1; la primera llega a los 50 ms de arrancar.

```
<cabeceras correctas>
int c = 0;
void a(int s1, siginfo_t *s2, void *p) {
  c += s2->si_value.sival_int;
}
int main(int argc, char **argv) {
   sigset_t s; pid_t p; struct sigaction s1;
   struct timespec t = {1, 0};
   int i; int n; char h[20];
   sigemptyset(&s);
   sigaddset(&s, SIGRTMIN);
   s1.sa_sigaction = a;
   s1.sa_flags = SA_SIGINFO;
   sigemptyset(&s1.sa_mask);
   sigaction(SIGRTMIN, &s1, NULL);
   sigprocmask(SIG_UNBLOCK, &s, NULL);
   sscanf(argv[1], "%d", &i);
   sscanf(argv[2], "%d", &n);
   t.tv_sec = i;
   while(nanosleep(&t, &t) == -1);
   printf("c vale %d\n", c);
   if(c < n) {
     sprintf(h, "%d", i+1);
     p = fork();
     if(p==0) execl(argv[0], argv[0],  h, argv[2], NULL);
   }
   return 0;
}
```

2. Para el siguiente programa explique **en general** cómo funciona, qué mensajes imprime por pantalla y cuanto tiempo tarda en ejecutarse. En particular, explique qué sucede cuando se invoca desde el intérprete de comandos como "**prog2 1 2 3 A**", siendo "**prog2**" el nombre del ejecutable.

```
<cabeceras correctas>
int n1; int n = 0; pthread_t h1;
pthread_mutex_t m1 =
   PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t    c1 =
   PTHREAD_COND_INITIALIZER;
void *f1(void *p) {
    while(1) {
      sleep(1);
      pthread_mutex_lock(&m1);
      n++;
      if(n>= n1) pthread_cond_signal(&c1);
      pthread_mutex_unlock(&m1);
    }
    return 0;
}
void *f2(void *p) {
     pthread_mutex_lock(&m1);
     while(n<n1) pthread_cond_wait(&c1, &m1);
     pthread_cancel(h1);
     pthread_mutex_unlock(&m1);
     return 0;
}
int main(int argc, char **argv) {
   pthread_t h2; n1 = argc;
   pthread_create(&h1, NULL, f1, (void *)argc);
   pthread_create(&h2, NULL, f2, (void *)argc);
   pthread_join(h1, NULL);
   printf("n vale %d\n", n);
   exit(0);
}
```

3. Explique cómo funcionan estos fragmento de programa en ADA, y cómo podría conseguirse una funcionalidad lo más equivalente posible utilizando colas de mensajes POSIX.

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

4. Explique en qué se modifica la recepción de señales cuando existen varios hilos, y lo que es necesario hacer para definir completamente el comportamiento, tanto en el caso síncrono como en el asíncrono.
5. Explique en qué consiste la condición de garantía de plazos basada en la utilización, y cuando puede aplicarse.

Datos que pueden ser útiles:

```
int sigaction(int sig, struct sigaction *act, struct sigaction *oact);
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 pthread_mutex_lock(pthread_mutex_t *mutex); int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_cond_broadcast(pthread_cond_t *cond); int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, const struct timespec *abstime);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
```

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \; ; \; U = \sum_{i=1}^{N} \frac{C_i}{T_i} \; ; \; U_0 = N\left(2^{1/N} - 1\right)$$

```
int mq_send(mqd_t cola, const char *datos, size_t longitud, unsigned int prioridad);
int mq_receive(mqd_t cola, const char *datos, size_t longitud, unsigned int *prioridad);
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 mq_notify(mqd_t cola, const struct sigevent *espec); pid_t fork(void);
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 mq_close(mqd_t cola); int mq_unlink(const char *nombre);
int timer_create(clockid_t reloj, struct sigevent *aviso, timer_t *tempo);
int timer_settime(timer_t tempo, int flags, const struct itimerspec *spec, struct itimerspec *spec_ant);
Constantes: TIMER_ABSTIME, CLOCK_REALTIME, DELAYTIMER_MAX
mqd_t mq_open(const char *mq_name, int oflag, mode_t modo, struct mq_attr *atributos);
Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_APPEND, O_TRUNC, O_NONBLOCK
struct mq_attr { long mq_maxmsg; long mq_msgsize; long mq_flags; long mq_curmsgs; };
```

Examen de Sistemas Informáticos en Tiempo Real
IAEI-II (3/12/10)

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

Resumen: Se pide realizar en C y con llamadas POSIX un programa para crear el proceso **receptor** de la figura. El proceso **receptor** crea un conjunto de procesos hijos y **recibe de ellos señales de tiempo real**, cuyos datos reenvía por la **cola de mensajes** de salida. El proceso receptor **funciona durante un tiempo** definido por sus argumentos, y finalmente acaba, después de imprimir el número de señales que cada proceso hijo ha enviado por unidad de tiempo y de hacer que todos los procesos hijos acaben. En el fichero de cabecera **receptor.h** que se incluye al final están definidos los macros y tipos citados en el enunciado.

Especificación detallada:

- **Argumentos de línea de comandos para el proceso receptor:**
  - Argumento 1: Número de segundos durante los cuales estará funcionando el proceso receptor (entero codificado en caracteres).
  - Argumento 2: Nombre de la cola de mensajes de salida.
  - Argumentos 3 al último: Nombres de los ejecutables de los procesos hijos, desde el 0 hasta el último de ellos.
- **Inicialización**: El proceso **receptor** crea la cola de mensajes de salida y los procesos hijos.
  - La cola de mensajes se crea con el nombre que define el argumento 2 del proceso **receptor**, asegurándose de que no se utilizará una cola anteriormente creada con el mismo nombre. Las colas tendrán capacidad para **LCOLA mensajes del tamaño de una variable de tipo struct mensaje**, que se define en la cabecera **receptor.h** (incluida más adelante).
  - Los procesos hijos desde el 0 al último se crean a partir de los ejecutables definidos por los argumentos del proceso **receptor** a partir del 3; por tanto, el número de procesos creados depende del número de argumentos del proceso **receptor**. El i-ésimo proceso hijo ha de recibir como argumento 1 el número de señal **SIGRTMIN+i**.
- **Funcionamiento normal:** El proceso **receptor** recibe señales de los procesos hijos durante el número de segundos que define su argumento 1. El i-ésimo proceso hijo envía la señal cuyo número ha recibido como argumento 1, es decir, **SIGRTMIN+i**. Cada vez que el proceso **receptor** recibe una señal, envía una estructura de tipo **mensaje** conteniendo el dato recibido con la señal y el número del proceso que la ha enviado. También actualiza los datos necesarios para calcular al final el promedio de señales recibidas por unidad de tiempo desde cada proceso hijo.
- **Terminación:** El proceso **receptor** comienza las operaciones de terminación cuando se agota su tiempo de funcionamiento, definido por su argumento 1. Cuando esto sucede deja de recibir señales de los procesos hijos, y **para cada uno de ellos imprime por pantalla el promedio de señales recibidas por unidad de tiempo** (un valor en punto flotante). También envía a cada proceso hijo una señal **SIGTERM** y espera a que acabe. Finalmente destruye la cola de salida y acaba. Para detectar la condición de fin **se utilizará un temporizador POSIX.**

Fichero de cabecera:

```
/* Cabecera receptor.h */
#define LCOLA 100
struct mensaje {
   int iproc;
   int dato;
};
```

NOTAS:

- Es necesario utilizar **temporizadores POSIX** para cumplir las condiciones de tiempos.
- No es necesario considerar tratamiento de errores.
- Es preciso acompañar el programa de pseudocódigo o explicación de su funcionamiento.