Examen de Sistemas Informáticos en Tiempo Real
IAEI-II (5/9/11)

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 a) Qué es un proceso, y qué estructuras de datos lo componen. b) Qué es un hilo y cuáles son sus diferencias con respecto a un proceso.
2. Para el siguiente programa explique **en general** cómo funciona y qué mensajes imprime por pantalla. En particular, explique qué sucede cuando el programa se invoca desde el intérprete de comandos como "**c2 1 2 3**", siendo "**c2**" el nombre del ejecutable.

```
<cabeceras correctas>
sigset_t s;
void f(pid_t a, int d) {
  siginfo_t b; union sigval c; int j;
  sigwaitinfo(&s, &b);
  d = d + b.si_value.sival_int;
  c.sival_int = d;
  sigqueue(a, SIGRTMIN, c);
}
void h(int s, siginfo_t *i, void *p) {}
int main(int a1, char **a2)
{
  pid_t *b; int i;  int d;
  union sigval v; siginfo_t x;
  struct sigaction sa;
  sa.sa_flags = SA_SIGINFO;
  sa.sa_sigaction = h;
  sigemptyset(&sa.sa_mask);
  sigaction(SIGRTMIN, &sa, NULL);
  sigemptyset(&s); sigaddset(&s, SIGRTMIN);
  sigprocmask(SIG_BLOCK, &s, NULL);
  b = malloc(a1*sizeof(pid_t));
  b[0] = getpid();
  for(i=1; i<a1; i++)  {
    b[i] = fork();
    if(!b[i])  {
      sscanf(a2[i], "%d", &d);
      f(b[i-1], d); exit(0);
    }
  }
  v.sival_int = 0;
  sigqueue(b[a1-1], SIGRTMIN, v);
  sigwaitinfo(&s, &x);
  printf("%d\n", x.si_value.sival_int);
}
```

3. Para el siguiente programa explique **en general** cómo funciona y qué mensajes imprime por pantalla. En particular explique qué sucede si recibe tres señales SIGRTMAX con datos 1, 2 y 2 a los 100, 200 y 300 ms desde el comienzo, respectivamente. Explique cómo cambia el funcionamiento si se sustituye **pthread_cond_broadcast** por **pthread_cond_signal**.

```
<cabeceras correctas>
#define N 5
int n; int n1 = 0;
pthread_mutex_t m =
  PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c =
  PTHREAD_COND_INITIALIZER;
void *g(void *p)  {
  int k = 0;
  pthread_mutex_lock(&m);
  while(n == 0) {
    pthread_cond_wait(&c, &m);
  }
  n--; n1++; if(n1==N) k = 1;
  pthread_mutex_unlock(&m);
  printf("Hilo saliendo\n");
  if(k==1) kill(getpid(), SIGALRM);
  return NULL;
}
void h1(int s, siginfo_t *i, void *p) {};
int main(int a1, char **a2)  {
  pthread_t h; int i, j;
  struct sigaction sa; sigset_t s; siginfo_t v;
  sa.sa_flags = SA_SIGINFO;
  sa.sa_sigaction = h1;
  sigemptyset(&sa.sa_mask);
  sigaction(SIGRTMAX, &sa, NULL);
  sigemptyset(&s); sigaddset(&s, SIGRTMAX);
  sigaddset(&s, SIGALRM);
  pthread_sigmask(SIG_BLOCK, &s, NULL);
  for(i=0; i<N; i++)  {
    pthread_create(&h, NULL, g, NULL);
  }
  alarm(5);
  do  {
    j = sigwaitinfo(&s, &v);
    if(j == SIGRTMAX)  {
      printf("Recibido %d\n",
            v.si_value.sival_int);
      pthread_mutex_lock(&m);
      n += v.si_value.sival_int;
      pthread_cond_broadcast(&c);
      pthread_mutex_unlock(&m);
    }
  } while(j == SIGRTMAX);
  exit(0);
}
```

4. Explique todo lo referente a la cancelación de hilos.
5. En la tabla se muestran para los procesos A, B, y C el periodo (T) y el coste (C). El "deadline" o tiempo límite (D) es igual a C. Se pide a) Asigne prioridades de manera óptima, explicando el criterio utilizado b) Aplique el criterio de utilización mínima garantizada y explicar qué indica el resultado c) Determine si se cumplen o no las restricciones temporales calculando el tiempo de respuesta d) Dibuje el cronograma a partir del instante crítico, y compruebe con él el resultado anterior. (**Atención: NO se pide un ejecutivo cíclico**).

Proceso	T (ms)	C (ms)
A	20	8
B	40	10
C	10	2

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 pthread_cancel(pthread_t thread); void pthread_testcancel(void);
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 pthread_setcancelstate(int state, int *oldstate);
state: PTHREAD_CANCEL_ENABLE, PTHREAD_CANCEL_DISABLE
```

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

```
int 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);
type: PTHREAD_CANCEL_DEFERRED, PTHREAD_CANCEL_ASYNCRONOUS
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);
```

Examen de Sistemas Informáticos en Tiempo Real
IAEI-II (5/9/11)

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

Resumen: Se pide realizar en C y con llamadas POSIX un programa para generar una salida digital a partir de las especificaciones que recibe por medio de una cola de mensajes.

Especificación detallada:

- **Generación de la señal**: La señal está caracterizada por su ciclo y su "duty cycle" (porcentaje de tiempo del ciclo durante el que la señal está a 1). El ciclo comienza con la salida a 1 (ver figura). Los dos valores (ciclo y "duty cycle") cambian de acuerdo con las especificaciones que se reciben por la cola de mensajes, pero siempre **después de haber terminado un ciclo completo**; mientras no se reciban especificaciones nuevas, se siguen generando ciclos con los valores anteriores. **Inicialmente** (antes de recibir ningún mensaje) la señal tendrá un ciclo de 1 segundo y un "duty cycle" del 50%. Para cambiar el valor de la señal se dispone de la función **cambiar_salida**, especificada en el fichero de cabecera "**programa.h**" que se incluye más adelante.
- **Cola de mensajes**: El programa crea la cola de mensajes. El nombre de la cola viene dado por el **argumento 1 de la línea de comandos**. La cola tendrá capacidad para 10 mensajes con tamaño igual al de la estructura **param** (ver "**programa.h**"). Cada mensaje es una estructura de dicho tipo, en la que **ciclo** es el ciclo de la salida en milisegundos, y **dcic** el "duty cycle" en tanto por ciento. Por ejemplo, si **ciclo** vale 2500 y **dcic** 50 tendrá que generarse una salida de 2.5 segundos de periodo que está a 1 durante 1.25 segundos en la primera parte del ciclo ("duty cycle" del 50%).
- **Condición de fin**: Si no se recibe ninguna especificación de salida durante **timeout** segundos el programa destruye la cola de mensajes y acaba. El valor **timeout** es un entero que se recibe por medio del **argumento 2 de la línea de comando**.

NOTAS:

- La generación de la salida debe estar basada en un **temporizador o temporizadores POSIX**. Para la condición de fin no existe esa restricción.
- No es necesario considerar tratamiento de errores.
- Es preciso acompañar el programa de pseudocódigo o explicación de su funcionamiento.

Fichero de cabecera:

```
/* Cabecera problema.h  */
struct param
{
   long ciclo;  /* Ciclo en milisegundos */
   float dcic;  /* "Duty cicle" en % del ciclo: 50 es el 50% */
};
void cambiar_salida(int valor); /* Cambiar salida a valor (0 ó 1) */
```