Examen de Sistemas Informáticos en Tiempo Real
IAEI-II(3/2/09)

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. Explique en qué consiste el direccionamiento virtual y paginado y qué problemas soluciona.
2. Para el siguiente programa Explique **en general** su funcionamiento, suponiendo que recibe señales entre SIGRTMIN y SIGRTMIN+N1-1, ambas inclusive, acompañadas de datos. Explique en particular qué sucede y qué mensajes aparecen en pantalla cuando el proceso recibe durante los primeros 300 ms 3 señales SIGRTMIN+1 con dato 2 y 7 señales SIGRTMIN+3 con dato 1.

```
<cabeceras correctas>
#define N1 5
int t[N1];
void g(int a, siginfo_t *b, void *c) {
   t[a-SIGRTMIN] += b->si_value.sival_int;
}
int main(int argc, char **argv) {
   struct timespec h = {0, 500000000L};
   int i, j; sigset_t d; struct sigaction a;
   siginfo_t f; union sigval u; pid_t b;
   sigemptyset(&d);
   a.sa_flags = SA_SIGINFO;
   a.sa_sigaction = g;
   sigemptyset(&a.sa_mask);
   for(i=0; i<N1;i++) {
     t[i]= 0;
     sigaddset(&d, SIGRTMIN+i);
     sigaction(SIGRTMIN+i, &a, NULL);
   }
   sigprocmask(SIG_UNBLOCK, &d, NULL);
   b = fork();
   if(b==0)  {
     sigprocmask(SIG_BLOCK, &d, NULL);
     sigwaitinfo(&d, &f);
     printf("%d con %d\n",
       f.si_signo, f.si_value.sival_int);
     exit(1);
   }
   while(nanosleep(&h, NULL) == -1);
   u.sival_int = -1;
   for(i=0; i<N1; i++) {
     if(t[i] > u.sival_int) {
       j=i; u.sival_int = t[i];
     }
   }
   sigqueue(b, SIGRTMIN+j, u);
   exit(0);
}
```

3. Explique el funcionamiento de los temporizadores POSIX: a) Notificación de eventos b) Programación en sus diferentes posibilidades c) Significado de la cuenta de "overrun" y acceso a ella.
4. Para el siguiente código La cola **"/cin"** (que ya existe, y está vacía inicialmente) se abre correctamente, y tiene **capacidad para 2 mensajes de tamaño sizeof(int)**. Explique como funciona el programa y qué aparece por pantalla a) Tal y como está b) Suprimiendo O_NONBLOCK en **mq_open**. Deben especificarse los mensajes contenidos en la cola en cada iteración y su prioridad, antes y después de llamar a **mq_receive**.

```
<cabeceras correctas>
#define N1 6
int main(int a, char **b) {
   mqd_t cola;
   int i, j, p;
   cola = mq_open("/cin", O_RDWR |
          O_NONBLOCK, 0, NULL);
   for(i=0; i<N1; i++)  {
   mq_send(cola, (char *)&i,  sizeof(int), 2*i);
   if(i>3) {
     mq_receive(cola, (char *)&j, sizeof(int), &p);
     printf("i: %d, j: %d, p: %d\n", i, j,p);
   }
}
printf("despues de for:\n");
while(mq_receive(cola, (char *)&j,
                 sizeof(int), &p) != -1)
   printf("i: %d, j: %d, p: %d\n", i, j, p);
}
```

5. Dada la siguiente especificación de tareas, para las que el plazo (o "deadline") es en todos los casos igual al periodo, a) Asignar prioridades de manera óptima b) Comprobar la condición de garantía de los plazos basada en la utilización mínima garantizada. ¿Qué nos indica el resultado obtenido? c) Determinar si se cumplen los plazos mediante la ecuación del cálculo del tiempo de respuesta, y dibujar el cronograma a partir del instante crítico (Atención: **No se pide un ejecutivo cíclico**).

Nombre	Periodo (ms)	Coste (ms)
A	10	5
B	30	9
C	50	5

Datos que pueden ser útiles:

```
int sigaction(int sig, struct sigaction *act, struct sigaction *oact);
int kill(pid_t pid, int sig);
```

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

```
int 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_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;
        };
struct timespec {
          time_t tv_sec;
          long tv_nsec;
        };
struct mq_attr {   long mq_maxmsg;
            long mq_msgsize;
            long mq_flags;
            long mq_curmsgs; };

int sigaddset(sigset_t *pset, int sig); int sigdelset(sigset_t *pset, int sig);
int sigprocmask(int how, const sigset_t *set, sigset *oset);
int sigqueue(pid_t pid, int sig, const union sigval val);
int mq_notify(mqd_t cola, const struct sigevent *espec); pid_t fork(void);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
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_getattr(mqd_t cola, struct mq_attr *atributos);
int mq_close(mqd_t cola); int mq_unlink(const char *nombre);
int mq_notify(mqd_t cola, const struct sigevent *espec);
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);
mqd_t mq_open(const char *mq_name, int oflag, mode_t modo, struct mq_attr *atributos);
Modo: S_I + (R, W, X) + (USR, GRP, OTH). También S_IRWXU, S_IRWXG, S_IRWXO
Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_APPEND, O_TRUNC,
O_NONBLOCK
CLOCK_REALTIME, DELAYTIMER_MAX, TIMER_ABSTIME
```

Examen de Sistemas Informáticos en Tiempo Real
IAEI-II(3/2/09)

PROBLEMA. TIEMPO: 2 HORAS (VALORACIÓN: 50%).

Resumen: Se pide realizar en C y con llamadas POSIX un programa para crear un proceso **multihilo** que controla el sistema de la figura. Deberán existir los siguientes hilos:

- **Hilo de recepción de órdenes:** Recibe órdenes de transporte mediante pares de señales SIGRTMIN y SIGRTMIN+1, y se las notifica a los hilos de vehículo a través de los datos compartidos. El dato de la señal SIGRTMIN indica el índice punto de carga, y el de SIGRTMIN+1 el índice del punto de descarga. Estos dos índices definen la orden, y tienen valores entre 0 y **N_PUNTOS**-1 (**N_PUNTOS** es el número de puntos de carga o descarga). **No se reconocerá otra orden hasta que la anterior haya sido aceptada por algún hilo de vehículo** (no importa cual de ellos).
- **Hilos de control de vehículo:** Estos hilos **deben compartir el mismo código**, pero controlan los diferentes vehículos; el índice del vehículo controlado lo determinará el **argumento de la función de arranque**. El número de vehículos y de hilos de control lo define el macro **N_VEH**. El funcionamiento de cada vehículo es el siguiente:
    - Los vehículos están parados en el punto de aparcamiento hasta que deben ejecutar una orden. Inicialmente puede suponerse que se encuentran en ese punto.
    - Cada hilo de control cuyo vehiculo está parado **compite con el resto por una orden**; cuando la consigue, realiza el movimiento que se indica en la figura: Se dirige al punto de carga definido por la orden, recoge la pieza, la lleva al punto de descarga definido por la orden, la descarga y finalmente vuelve al punto de aparcamiento. Se utilizarán las siguientes funciones de biblioteca, definidas en la cabecera **problema.h**:
        - void **mueve**(int **veh**, int **fila**, int **col**). Permite mover el vehiculo **veh** a una posición definida for fila y columna. Cada cuadrado del dibujo tiene dimensión 1, y los **macros que definen los puntos clave de la figura** están en la cabecera **problema.h**. Por ejemplo, el aparcamiento del vehículo **i** está en el punto de fila **FILA_VUELTA** y columna **COL_APARC0+i**.
        - void **carga**(int **veh**) y void **descarga**(int **veh**): Permiten realizar la carga y la descarga del vehiculo **veh**.
    - Para salir o entrar al aparcamiento o pasar de la zona de carga a la de descarga hay que usar **un pasillo por el que sólo puede circular un vehículo a la vez**. Cuando el pasillo está ocupado los vehículos esperan en su punto de aparcamiento o en las columnas **COL_ESPERA1** y **COL_ESPERA2**, según el punto del recorrido en el que se encuentren. **No se considera la posibilidad de otras colisiones de vehículos que las que podrían suceder en el pasillo**, porque los vehículos se paran solos cuando encuentran un obstáculo.

Condición de parada:

Cuando recibe la señales **SIGTERM** o **SIGINT el proceso acaba**, **pero antes deben terminar las operaciones de transporte en ejecución**.

NOTAS:

- Las operaciones deben ser tan concurrentes (simultáneas) como sea posible.
- Se recomienda tratar las señales sincronamente.
- **Utilice mutex y variables de condición** para **sincronizar el acceso a las variables compartidas,** evitando (cuando sea posible) permanecer un tiempo largo en la sección crítica.
- Las constantes y prototipos de funciones se encuentran en la cabecera **problema.h**, que se supone accesible.
- No es necesario considerar tratamiento de errores en las llamadas al sistema.
- Es preciso acompañar el programa de pseudocódigo o explicación de su funcionamiento.

```
/* Cabecera problema.h */
/* Constantes simbólicas (valores para el
   ejemplo de la figura) */
#define N_VEH          3
#define N_PUNTOS       3
#define FILA_IDA       0
#define FILA_VUELTA    1
#define COL_ESPERA1    3
#define COL_ESPERA2    11
#define COL_APARC0     6
#define COL_CARGA0     0
#define COL_CARGA1     12

/* Funciones disponibles en biblioteca */
void mueve(int veh, int fila, int col);
void carga(int veh);
void descarga(int veh);
```