

## Examen de Sistemas Informáticos en Tiempo Real

### 4º II (10/6/13)

**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 en qué consisten y para qué se utilizan el puntero de pila y la pila del sistema.
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 SIGRTMIN acompañadas de datos. En particular, explique qué sucede cuando se reciben señales SIGRTMIN con datos 1, 5, 2 y 0 por ese orden, la primera a los 500 ms de arrancar y el resto separadas por 500 ms.

```
<cabeceras correctas>
#define N 4
struct s { char e1; int e2; };
void h(int a, siginfo_t *b, void *c) {}
int main(int b, char **argv) {
    struct s s1[N] = { {'a', 2}, {'b', 1},
                       {'c', 4}, {'d', 5} };
    sigset_t v; siginfo_t t; struct sigaction s;
    int c, i; struct timespec m;
    sigemptyset(&v); sigaddset(&v, SIGRTMIN);
    s.sa_flags = SA_SIGINFO;
    s.sa_sigaction = h;
    sigemptyset(&s.sa_mask);
    sigaction(SIGRTMIN, &s, NULL);
    sigprocmask(SIG_BLOCK, &v, NULL);
    do {
        sigwaitinfo(&v, &t);
        for(i=0, c=1; i<N && c==1; i++)
        {
            if(s1[i].e2 == t.si_value.sival_int) {
                c = 0;
                printf("%c\n", s1[i].e1);
                m.tv_sec = s1[i].e2; m.tv_nsec = 0;
                nanosleep(&m, NULL);
            }
        } while(c == 0);
    }
```

3. Explique la manera recomendada de esperar una condición mediante variables de condición, y qué sucede desde que el hilo o proceso entra en la espera de dicha variable hasta que sale de ella.
4. Para el siguiente programa explique **en general** cómo **funciona**, cuánto **tarda** en ejecutarse y qué **mensajes** imprime por pantalla. En particular, explique **a)** qué sucede cuando se ejecuta con el comando **c5 100 2 200 5 200 4 300 5**, siendo **c5** el nombre del ejecutable **b)** qué sucede cuando se ejecuta simplemente como **c5**. Para contestar correctamente debe especificar el contenido de la cola de mensajes después de cada cambio. La cola **“/c1”** existe y tiene capacidad para 6 mensajes del tamaño de un **int**.

```
<cabeceras correctas>
#define NMSG 6
void *r1(void *p) {
    struct timespec t;
    t.tv_sec = 0; t.tv_nsec = (long)p*1000000;
    nanosleep(&t, NULL);
    return NULL;
}
int main(int b, char **c) {
    int i, k1 = 100, k2;
    int b1, b2, r;
    mqd_t c1; pthread_t h1;
    c1 = mq_open("/c1", O_RDWR |
                 O_NONBLOCK, 0, NULL);
    for(i=1; i<b; i+=2) {
        if(i+1 < b) {
            sscanf(c[i], "%d", &k1);
            sscanf(c[i+1], "%d", &k2);
            mq_send(c1, (char *)&k1, sizeof(int), k2);
        }
        r = mq_receive(c1, (char *)&k1, sizeof(int), &k2);
        if(r != -1 && b2 == k2) {
            printf("%d %d\n", k1, k2);
            do {
                r = mq_receive(c1, (char *)&b1,
                              sizeof(int), &b2);
                if(r != -1 && b2 == k2) {
                    printf("%d %d\n", b1, b2);
                    k1 = k1 + b1;
                } while(r != -1 && b2 == k2);
            }
            printf("k1 %d\n", k1);
            pthread_create(&h1, NULL, r1, (void *)&k1);
            pthread_join(h1, NULL);
        }
    }
```

5. En la tabla se muestran para los procesos A, B, y C el periodo (T) el coste (C) y el tiempo límite, plazo o “deadline” (D). Se pide a) Asigne prioridades de manera óptima, explicando el criterio utilizado b) Determine si se cumplen o no las restricciones temporales calculando el tiempo de respuesta c) 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)	D (ms)
A	10	4	9
B	15	5	7
C	30	4	28

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 mq_getattr(mqd_t cola, struct mq_attr *atributos);
int pthread_setcancelstate(int state, int *oldstate);
state: PTHREAD_CANCEL_ENABLE, PTHREAD_CANCEL_DISABLE
```

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

```
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;
int sigaddset(sigset_t *pset, int sig); int sigdelset(sigset_t *pset, int sig);
int sigprocmask(int how, const sigset_t *set, sigset_t *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);
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);
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
int pthread_mutex_lock(pthread_mutex_t *mutex); pthread_mutex_trylock(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); i
nt 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);
```

## Examen de Sistemas Informáticos en Tiempo Real 4º II (10/6/13)

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

**Resumen:** Se pide realizar en C y con llamadas POSIX un programa multihilo que recibe los valores de **N\_SIG** señales físicas externas codificados en frecuencia, los decodifica y genera una alarma cuando se detecta una determinada condición. Las constantes simbólicas y declaraciones necesarias están definidas en la cabecera **problema.h**, que se supone disponible.

### Especificación detallada:

- **Argumentos de la línea de comandos:** Los argumentos de la línea de comandos se interpretan como **tciclo** (argumento 1, entero que expresa un número de milisegundos), **vlimite** (argumento 2, número fraccionario en la unidad de medida de las señales físicas, equivalente a número de señales por segundo) y **nmaxcic** (argumento 3, entero que indica número de ciclos). Todos ellos están codificados en caracteres.
- **Recepción y decodificación de señales:** Las señales físicas externas se reciben mediante otras tantas señales POSIX desde **SIGRTMIN** a **SIGRTMIN+N\_SIG-1**. La frecuencia de llegada de la señal **SIGRTMIN+i** expresada en número de señales por segundo es el valor de la señal física **i**-ésima. Para calcular tal valor se contarán las señales recibidas de cada tipo durante un periodo fijo, **tciclo**. Al final de cada ciclo de recepción se calculará cada uno de los **N\_SIG** valores, expresados en **número de señales por segundo** y con tipo **float**.
- **Alarma:** La condición de alarma consiste en que el valor calculado en al menos la mitad de las señales alcance o supere el valor **vlimite**. Cuando esto sucede se activa un indicador luminoso que permanece en ese estado (independientemente de los valores calculados en ciclos posteriores) hasta que el proceso recibe una señal **SIGRTMAX**, que desactiva el indicador. La activación y la desactivación del indicador debe ser independiente de la recepción y decodificación de señales. Lógicamente si después de desactivar el indicador se detecta inmediatamente la condición de alarma, se tendrá que activar de nuevo. Para activar y desactivar el indicador se dispone de la función de biblioteca **void indicador(int valor)**.
- **Condiciones de fin:** Con cualquiera de las siguientes condiciones el proceso acaba, pero antes debe terminar la última decodificación de valores (si está en curso) y desactivar el indicador de alarma:
  - Llegada de una señal **SIGTERM**.
  - No llega ninguna señal durante al menos **nmaxcic** ciclos.

### Otras condiciones:

- El ciclo debe realizarse con un **temporizador POSIX 1003.1b**.
- El programa consistirá en **al menos dos hilos**, uno para gestionar el cálculo de los valores y otro para gestionar la activación y la desactivación de la alarma.
- El programa deberá funcionar **independientemente** de los valores concretos de los argumentos y las constantes simbólicas.
- Es necesario utilizar mutex y variables de condición según lo indicado en clase para gestionar el acceso y la sincronización a datos compartidos.
- Se recomienda tratar las señales síncronamente, aunque se admiten otras soluciones.
- 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 */
```

```
#define N_SIG 4
```

```
/* Función para activar y desactivar el indicador de alarma  
(valor=0: desactivar, valor=1: activar) */
```

```
void indicador(int valor);
```

