GUÍA POSIX SEÑALES Y TEMPORIZADORES

Se incluyen las dependencias:

```c
#include <stdlib.h>
#include <stddef.h>
#include <assert.h>
#include <string.h>
#include <stdio.h>

#include <unistd.h>
#include <time.h>
#include <errno.h>
#include <sched.h>
#include <pthread.h>
#include <sys/mman.h>
#include <signal.h>
```

Se definen los periodos de las tareas. Todos los periodos en ms se pasan a nsec.

Se definen las prioridades. Las tareas con menor plazo son las más prioritarias.

Se define el incremento o decremento.

```c
#define PERIODO_A_SEC 1
#define PERIODO_A_NSEC 0
#define PRIO_A 28
#define INC_A 1

#define PERIODO_B_SEC 2
#define PERIODO_B_NSEC 0
#define PRIO_B 26
#define INC_B 2

#define PRIO_C 24
```

Se crea un struct por cada mutex, con la variable compartida (contador, valor) y el mutex.

```c
struct Data{
    int contA;
    pthread_mutex_t mutexA;
    int contB;
    pthread_mutex_t mutexB;
};
```

OPCIONAL: FASE DE SOPORTE

OPCIONAL: Se definen los CHKN y CHKE, para controlar los errores al llamar al sistema.

```
// #define CHKN(syscall) \
//     do { \
//         int err = syscall; \
//         if (err != 0) { \
//             fprintf(stderr, "%s: %d: SysCall Error: %s\n", \
//                     __FILE__, __LINE__, strerror(errno)); \
//             exit(EXIT_FAILURE); \
//         } \
//     } while (0)

// #define CHKE(syscall) \
//     do { \
//         int err = syscall; \
//         if (err != 0) { \
//             fprintf(stderr, "%s: %d: SysCall Error: %s\n", \
//                     __FILE__, __LINE__, strerror(err)); \
//             exit(EXIT_FAILURE); \
//         } \
//     } while (0)
```

OPCIONAL: Se define la función getTime, para obtener el tiempo actual y pasarlo a string. Esto se usa para mensajes en depuración.

```
// const char *get_time (char *buf){
//     time_t t = time(0); //Tiempo actual
//     char *f = ctime_r(&t, buf); //Convertir el tiempo en una cadena y
almacenarla en buf
//     f[strlen(f)-1] = '\0'; //Eliminar el salto de línea
//     return f; //Retornar la cadena
// }
```

FASE DE LAS TAREAS

Se crean las tareas mediante métodos void:

- Se crea un periodo mediante “const struct timespec”, que es igual a {periodo_seg, periodo_nseg}
- Se crea un temporizador “timer_t” timerid.
- Se crea un evento señal “struct sigevent” sgev, que le dice al temporizador que mande una señal de tipo SIGRTMIN.
    - sgev.sigev_notify le dice que el evento es de tipo señal.
    - sgev.sigev_signo le dice la señal que es
    - sgev.sigev_value.sival_ptr indica a dónde tiene que mandar esta señal (&timerid).
    - Finalmente, se crea un temporizador mediante “timer_create” con los parámetros (CLOCK_MONOTIC, &sgev, &timerid);
- Se crea un iterador “struct itimerspec” its, necesario para el temporizador.

    - Se le asigna el intervalo mediante “its.it_interval”, que será el periodo creado anteriormente.
    - Se le asigna el valor 0 al “its.it_value.tv_sec”
    - Se le asigna el valor 1 al “its.it_value.tv_nsec”
    - Se le asigna el tiempo al temporizador mediante “timer_settime” con los parámetros (timerid, 0, &its, NULL)
- Se crea un valor “int signum” para almacenar la señal posteriormente.
- OPCIONAL: Se crea un char buf[30] para mostrar en depuración la hora actual.
- Se asigna al puntero struct de Data el arg recibido por parámetro. “struct Data *data = arg;
- Se crea un parámetro para la política de planificación “struct sched_param”.
- OPCIONAL: Se crea un “const char *pol” y un “int policy”.
    - Posteriormente se obtiene la política de planificación mediante “pthread_getschedparam” recibiendo como parámetros (pthread_self(), &policy, &param).
    - Finalmente, se asigna a “pol” la política obtenida mediante condicionales:

```
-   pol = (policy == SCHED_FIFO) ? "FF" : (policy == SCHED_RR) ? "RR" :
    "--"; //Obtener la política de planificación
```

- Se vacía el set de señales “sigemptyset(&sigset)”;

- Se añade la señal correspondiente al set mediante “sigaddset(&sigset, SIGTRMIN+1)

- Se hace un bucle infinito “while(1)”:

    - Se espera a la señal que se desea recibir, que está indicada en “sigset”, para almacenarla en “signum”. “sigwait(&sigset, &signum)”.
    - Se bloquea el mutex “pthread_mutex_unlock(&data->mutex)”;
    - Se aumenta el contador con el INC correspondiente. “data->cont += INC”.
    - Se printea la tarea. “printf("Tarea A: [%d]\n", data->contA);”
    - Se desbloquea el mutex. “pthread_mutex_unlock(&data->mutexA);
- Se borra el temporizador. “timer_delete(timerid)”.
- Retornas NULL.

```
void *tareaA (void *arg){
    const struct timespec periodo = {PERIODO_A_SEC, PERIODO_A_NSEC};
    timer_t timerid;
    struct sigevent sgev;
    struct itimerspec its;
    sigset_t sigset;
    //char buf[30];
    struct Data *data = arg;
    struct sched_param param;
    // const char *pol;
    int signum;
    //int i, policy;
```

```
    //pthread_getschedparam(pthread_self(), &policy, &param); //Obtener
la política de planificación y los parámetros
    //pol = (policy == SCHED_FIFO) ? "FF" : (policy == SCHED_RR) ? "RR" :
"--"; //Obtener la política de planificación

    sgev.sigev_notify = SIGEV_SIGNAL;
    sgev.sigev_signo = SIGRTMIN+1;
    sgev.sigev_value.sival_ptr = &timerid;

    timer_create(CLOCK_MONOTONIC, &sgev, &timerid);

    its.it_interval = periodo;
    its.it_value.tv_sec = 0;
    its.it_value.tv_nsec = 1;
    timer_settime(timerid, 0, &its, NULL);

    sigemptyset(&sigset);
    sigaddset(&sigset, SIGRTMIN+1);

    while(1){
        sigwait(&sigset, &signum);
        //printf("Tarea A [%s]\n", get_time(buf)); //Mensaje de
depuración
        pthread_mutex_lock(&data->mutexA);
        data->contA += INC_A;
        printf("Tarea A: [%d]\n", data->contA);

        if(data->contA % 10 == 0){
            kill(getpid(), (SIGRTMIN+3));
        }
        pthread_mutex_unlock(&data->mutexA);
    }
    timer_delete(timerid);
    return NULL;
}
```

FASE PRINCIPAL

- OPCIONAL: Función usage para mostrar si hay error cuando introduces en consola un parámetro equivocado.

```
-   // void usage (const char *nm){
-   //     fprintf(stderr, "usage: %s [-h] [-ff] [-rr] [-p1] [-p2]\n",
    nm); //Imprimir mensaje de uso
-   //     exit(EXIT_FAILURE); //Salir con error
-   // }
```

- OPCIONAL: Función get_args, que se complementa con usage, para asociar el parámetro recibido por consola con la política correcta.

```
- // void get_args (int argc, const char *argv[], int *policy, int
  *prio1, int *prio2){
- //     int i; //Contador
- //     if (argc < 2){
- //         usage(argv[0]); //Imprimir mensaje de uso
- //     }else{
- //         for(i = 1; i<argc; i++){
- //             if (strcmp(argv[i], "-h") == 0){
- //                 usage(argv[0]); //Imprimir mensaje de uso
- //             }else if (strcmp(argv[i], "-ff") == 0){
- //                 *policy = SCHED_FIFO; //Asignar la política de
  planificación FIFO
- //             }else if (strcmp(argv[i], "-rr") == 0){
- //                 *policy = SCHED_RR; //Asignar la política de
  planificación RR
- //             }else if (strcmp(argv[i], "-p1") == 0){
- //                 *prio1 = PRIORIDAD_A; //Asignar la prioridad de
  la tarea A
- //                 *prio2 = PRIORIDAD_B; //Asignar la prioridad de
  la tarea B
- //             }else if (strcmp(argv[i], "-p2") == 0){
- //                 *prio1 = PRIORIDAD_B; //Asignar la prioridad de
  la tarea B
- //                 *prio2 = PRIORIDAD_A; //Asignar la prioridad de
  la tarea A
- //             }else{
- //                 usage(argv[0]); //Imprimir mensaje de uso
- //             }
- //         }
- //     }
- // }
```

- Función “int main(int argc, const char *argv)”:

    - Se crea un conjunto de señales “sigset_t” sigset.
    - Se crea una variable del struct Data, que podemos llamarle “shared_data”, por ejemplo.
    - Se crea una variable “pthread_attr_t” attr.
    - Se crea una variable “struct sched_param” param.
    - Se crean t+1 variables de prioridad, siendo t el número de tareas.
        - Int prio0=1, el resto se le asigna la prioridad definida globalmente.
        - Se crean t variables de hebras. “pthread_t” t1, t2, etc.
    - “mlockall(MCL_CURRENT | MCL_FUTURE)”. Siempre igual
    - Se le asigna a prio0 la prioridad más alta +1. Ej: “prio0 = PRIO_A +1”
    - Se le asigna a param la prioridad 0, mediante “param.sched_priority”.

- Se almacena en param la prioridad deseada. Ej: “pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
- Se vacía el set. “sigemptyset(&sigset)”.
- Se añaden al set las señales correspondientes. “sigaddset(&sigset, SIGTRMIN+1)”, etc.
- “pthread_sigmask(SIG_BLOCK, &sigset, NULL)”.
- Se inicializa el struct
  - Se setean los valores a 0. “shared_data.cont=0”.
  - Se inicializan los mutex. “pthread_mutex_init(&shared_data.mutex, NULL)”.
- Se inicializa el attr.
  - “pthread_attr_init(&attr)”
  - “pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED)”
  - “pthread_attr_setschedpolicy(&attr, SCHED_FIFO)”
- Se crean las hebras
  - “param.sched_priority = prio”
  - “pthread_attr_setschedparam(&attr, &param)”
  - “pthread_create(&t, &attr, tarea, &shared_data)”
  - Este proceso se repite para todas las hebras.
- Se destruye attr. “pthread_attr_destroy(&attr)”.
- Se hace un join PARA CADA hebra. “pthread_join(t, NULL);
- Se destruye el/los mutex. “pthread_mutex_destroy(&shared_data.mutex)
- Se retorna 0.

```
int main(int argc, const char *argv){
    sigset_t sigset;
    struct Data shared_data;
    pthread_attr_t attr;
    struct sched_param param;

    int prio0 = 1, prio1 = PRIO_A, prio2 = PRIO_B, prio3 = PRIO_C;
    pthread_t t1, t2, t3;

    mlockall(MCL_CURRENT | MCL_FUTURE);
    prio0 = PRIO_A + 1;
    param.sched_priority = prio0;
    pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);

    sigemptyset(&sigset);
    sigaddset(&sigset, SIGTRMIN+1);
    sigaddset(&sigset, SIGTRMIN+2);
    sigaddset(&sigset, SIGTRMIN+3);
    sigaddset(&sigset, SIGTRMIN+4);
    pthread_sigmask(SIG_BLOCK, &sigset, NULL);

    shared_data.contA=0;
    shared_data.contB=0;
```

```
    pthread_mutex_init(&shared_data.mutexA, NULL);
    pthread_mutex_init(&shared_data.mutexB, NULL);

    pthread_attr_init(&attr);
    pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr, SCHED_FIFO);

    param.sched_priority = prio1;
    pthread_attr_setschedparam(&attr, &param);
    pthread_create(&t1, &attr, tareaA, &shared_data);

    param.sched_priority = prio2;
    pthread_attr_setschedparam(&attr, &param);
    pthread_create(&t2, &attr, tareaB, &shared_data);

    param.sched_priority = prio3;
    pthread_attr_setschedparam(&attr, &param);
    pthread_create(&t3, &attr, tareaC, &shared_data);

    pthread_attr_destroy(&attr);

    printf("Tarea principal con política FIFO \n");

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);

    pthread_mutex_destroy(&shared_data.mutexA);
    pthread_mutex_destroy(&shared_data.mutexB);
    return 0;

}
```