**GUÍA POSIX SEÑALES Y TEMPORIZADORES**

Se incluyen las dependencias:

#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>

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.

#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.

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;

}