HITO3 PRÁCTICA FSO 24-25

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>


typedef struct { // estructura del buffer circular
    char cadena[32]; // cadena que almacena
    int longitud; // la longitud de la cadena
} Buffer;

typedef struct Nodo{ // lista enlazada
    int id; // id del hilo consumidor
    long suma; // suma parcial
    struct Nodo* siguiente; // puntero al siguiente nodo
} Nodo;

typedef struct { // estructura para los argumentos del consumidor
    int id; // el id de cada hilo
} ConsumidorArgumentos;


// VARIABLES COMPARTIDAS
int TAM; // tamano del buffer
Buffer *bufferCircular; // buffer circular
Nodo *lista; // lista enlazada
sem_t hay_espacio; // semaforo que controla los espacios del buffer
circular
sem_t hay_dato; // semaforo que controla los datos del buffer circular
sem_t mutex; // semaforo que controla el acceso de los consumidores al
buffer circular
sem_t mutexLista; // semaforo que controla el acceso a la lista enlazada
sem_t hay_nodo; // semaforo que controla los datos de los nodos
int contadorConsumidores = 0; // índice de los consumidores

void *productores(void *arg);
void *consumidores(void *arg);
void *sumador(void *arg);
bool es_binario(char str[]);
long binario_decimal(char *num);
Nodo* insertarFinal(Nodo* cabeza, int id, long suma);
void liberarLista(Nodo* cabeza);
```

```c
int main(int argc, char **argv) {
    // -- VARIABLES DEL HITO 3 -- //

    char *path = "./procesa";
    char *comando = "procesa";
    char *arg1 = argv[1]; // fichero de entrada
    char *arg2 = argv[2]; // fichero de salida
    int estado; // estado del hijo
    __pid_t pid;
    int codigo_salida; // codigo de salida del exit del hijo

    FILE *salida; // fichero de salida
    int nhilos = atoi(argv[3]); // numero de hilos consumidores
    TAM = atoi(argv[4]); // tamaño del buffer circular
    pthread_t tidp, tidc[nhilos], tids;
    ConsumidorArgumentos *argConsumidor;

    FILE *resultados; // fichero de resultados

    // -------------------------- //

    // comprobamos los argumentos de entrada
    if(argc != 6) {
        fprintf(stderr, "Argumentos incorrectos. Se necesitan 6
parametros para ejecutar el hito 3\n");
        exit(1);
    }

    // comprobamos que el numero de hilos este dentro del rango permitido
    if(nhilos < 2 || nhilos > 1000) {
        fprintf(stderr, "El número de hilos no esta dentro del rango
permitido [2-1000]\n");
        exit(1);
    }

    // comprobamos que el tamaño del buffer circular este dentro del
rango permitido
    if(TAM < 10 || TAM > 1000) {
        fprintf(stderr, "El tamaño del buffer no esta dentro del rango
permitido [10-1000]\n");
        exit(1);
    }

    // creamos un hijo
    pid = fork();
    if(pid == -1) {
        printf("Error al crear el hijo\n");
        exit(1);
```

```
    }

    // el hijo ejecuta el programa procesa
    if(pid == 0) {
        if((execl(path, comando, arg1, arg2, NULL)) == -1) {
            fprintf(stdout, "Error en execl\n");
            exit(1);
        }

    // esperamos a que termine el hijo y comprobamos su salida
    } else {
        wait(&estado);
        if(WIFEXITED(estado)) {
            codigo_salida = WEXITSTATUS(estado);
            if(codigo_salida == 0) {

                // reserva de memoria para el buffer circular
                bufferCircular = (Buffer*)malloc(TAM * sizeof(Buffer));
                if(bufferCircular == NULL) {
                    fprintf(stderr, "Error al asignar memoria al buffer
circular\n");
                    exit(1);
                }

                // reserva de memoria para el argumento de los
consumidores
                argConsumidor =
(ConsumidorArgumentos*)malloc(sizeof(ConsumidorArgumentos)*nhilos);
                    if (argConsumidor == NULL) {
                        fprintf(stderr, "Error al asignar memoria a los
argumentos de los consumidores\n");
                        free(bufferCircular);
                        exit(1);
                    }

                // abrimos el fichero de salida
                salida = fopen(argv[2], "r");
                if(salida == NULL) {
                    fprintf(stderr, "Error al abrir el fichero de
salida\n");
                    free(bufferCircular);
                    free(argConsumidor);
                    exit(1);
                }

                // abrimos el fichero de los resultados en modo escritura
                resultados = fopen(argv[5],"w");
                if(resultados == NULL){
```

```
                    fprintf(stderr,"Error al abrir el fichero de los
resultados\n");
                    free(bufferCircular);
                    free(argConsumidor);
                    fclose(salida);
                    exit(1);
                }

                // INCIALIZACION DE SEMAFOROS
                sem_init(&hay_espacio, 0, TAM);
                sem_init(&hay_dato, 0, 0);
                sem_init(&mutex, 0, 1);
                sem_init(&mutexLista,0,1);
                sem_init(&hay_nodo,0,0);

                 // lista enlazada se inicializa vacia
                lista = NULL;

                // para cada hilo creo un nodo que tiene su id y su suma
incializada a -1
                for(int i =0;i<nhilos;i++){
                    lista = insertarFinal(lista,-1,-1);
                }

                //Creamos el hilo productor
                pthread_create(&tidp, NULL, productores, salida);

                //Creamos los hilos consumidores
                for(int i = 0; i < nhilos; i++) {
                    argConsumidor[i].id = i;
                    pthread_create(&tidc[i], NULL, consumidores,
&argConsumidor[i]);
                }

                // creamos el hilo sumador
                pthread_create(&tids,NULL,sumador,resultados);

                // Esperamos a que terminen el hilo productor, los
consumidores, y el sumador
                pthread_join(tidp, NULL);

                for(int i = 0; i < nhilos; i++) {
                    pthread_join(tidc[i], NULL);
                }

                pthread_join(tids,NULL);

                // liberamos memoria, cerramos los fichero y destruimos
los semaforos
```

```
                free(argConsumidor);
                free(bufferCircular);
                liberarLista(lista);
                fclose(salida);
                fclose(resultados);
                sem_destroy(&hay_espacio);
                sem_destroy(&hay_dato);
                sem_destroy(&mutex);
                sem_destroy(&mutexLista);
                sem_destroy(&hay_nodo);

                fprintf(stdout, "main : Procesado de fichero
terminado\n");

            } else {
                fprintf(stderr, "main : Procesado de fichero con
error\n");
            }
        } else {
            fprintf(stderr, "main : Proceso hijo finalizó con
errores\n");
        }
    }
}

void *productores(void *arg) {
    ssize_t i;
    char *linea = NULL;
    size_t espacio;
    FILE *fichero = (FILE*)arg;
    int contadorProductor = 0; //contador local de productores

    while((i = getline(&linea, &espacio, fichero)) != -1) {
        if(linea[i-1] == '\n') { // remplazamos el \n por el fin de
cadena
            linea[i-1] = '\0';
            i--;
        }

        if(i >= 1 && i <= 32 && es_binario(linea)) {
            sem_wait(&hay_espacio); //  entramos en la seccion critica
            strcpy(bufferCircular[contadorProductor].cadena, linea); //
copiamos la cadena en el buffer circular y su longitud
            bufferCircular[contadorProductor].longitud = i;
            contadorProductor = (contadorProductor + 1) % TAM;
            sem_post(&hay_dato); // señalamos que hay dato para salir de
la seccion critica

        }
```

```
    }

    sem_wait(&hay_espacio); // si hemos llegado al final marcamos la
longitud como -1
    bufferCircular[contadorProductor].longitud = -1;
    sem_post(&hay_dato);

    free(linea);
    pthread_exit(NULL);
}

void *consumidores(void *arg) {
    ConsumidorArgumentos *args = (ConsumidorArgumentos*)arg;
    int id = args->id;
    long suma = 0;
    long numero_decimal;
    bool sigue = true;
    Buffer dato;
    Nodo* inicio; // nodo que apunta a la cabeza de la lista

    while(sigue) {
        sem_wait(&hay_dato);
        sem_wait(&mutex);

        if (bufferCircular[contadorConsumidores].longitud == -1) {
            sem_post(&mutex);
            sem_post(&hay_dato);
            sigue = false; // si hemos llegado al fin del archivo que los
consumidores dejen de consumir
        }
        else {
                dato = bufferCircular[contadorConsumidores]; // cogemos
el dato

                if(dato.longitud == 32 && dato.cadena[0] != '1'){ //
comprobamos que sea de 32 bits y positiva
                    if(id % 2 == 0){ // comprobamos que el hilo sea par
                        numero_decimal = binario_decimal(dato.cadena);
                        if(numero_decimal % 2 == 0){ // si el numero es
par y su hilo tambien lo transforma(suma)

                            suma = (suma + numero_decimal) % (RAND_MAX /
2);

                            contadorConsumidores = (contadorConsumidores
+ 1) % TAM;
                            sem_post(&mutex);
                            sem_post(&hay_espacio);
```

```
                        } else { // si no es par lo devuelve al
bufferCircular

                            bufferCircular[contadorConsumidores] = dato;
                            sem_post(&mutex);
                            sem_post(&hay_dato);
                        }
                    }else { // el hilo es impar
                        numero_decimal = binario_decimal(dato.cadena);
                        if(numero_decimal % 2 != 0){ // si el numero es
impar lo transforma(suma)

                            suma = (suma + numero_decimal) % (RAND_MAX /
2);

                            contadorConsumidores = (contadorConsumidores
+ 1) % TAM;
                            sem_post(&mutex);
                            sem_post(&hay_espacio);
                        } else{ // si no es impar lo devuelve al
bufferCircular

                            bufferCircular[contadorConsumidores] = dato;
                            sem_post(&mutex);
                            sem_post(&hay_dato);
                        }
                    }
                } else { // si el numero no es de 32 bits ni positivo lo
descarta

                    contadorConsumidores = (contadorConsumidores + 1) %
TAM;
                    sem_post(&mutex);
                    sem_post(&hay_espacio);
                }
            }
        }

    // seccion critica para modificar la lista enlazada
    sem_wait(&mutexLista);
    inicio = lista;
    while(inicio->suma != -1){
        inicio = inicio->siguiente;
    }

    // almaceno en cada nodo su suma y su id
    inicio->suma = suma;
    inicio->id = id;
    sem_post(&mutexLista);
```

```
    // marco que hay un nodo con datos
    sem_post(&hay_nodo);

    pthread_exit(NULL);
}

void *sumador(void *arg){
    FILE *fichero = (FILE*)arg;
    long suma_parcial=0;
    long suma_total = 0;
    int id;
    bool sigue = true;
    Nodo *actual; // nodo actual

    // seccion critica para coger los resultados de la lista enlazada
    while(sigue){
        sem_wait(&hay_nodo);
        sem_wait(&mutexLista);

        // si el nodo esta vacio hemos llegado al final
        if(lista->siguiente == NULL){
            sem_post(&mutexLista);
            sigue=false;

        }else{

            //paso los parametros al nodo actual y salgo de la seccion
critica
            actual = lista;
            lista = actual->siguiente;
            sem_post(&mutexLista);

            suma_parcial = actual->suma;
            id = actual->id;
            suma_total = (suma_total + suma_parcial) % (RAND_MAX / 2);
            free(actual);

            //escribo en el fichero de resultados
            fprintf(fichero,"Hilo %d suma parcial:
%ld\n",id,suma_parcial);
            fflush(fichero);
        }

    }

    //escribo en el fichero de resultados
    fprintf(fichero,"Suma total: %ld\n",suma_total);
    fflush(fichero);
```

```
    pthread_exit(NULL);

}

 // metodo que comprueba si un numero es binario
bool es_binario(char str[]) {
    for(int i = 0; str[i] != '\0'; i++) {
        if(str[i] != '0' && str[i] != '1') {
            return false;
        }
    }
    return true;
}

 // metodo que convierte el numero a decimal
long binario_decimal(char *num) {
   int n = 0;

  for (int i = strlen(num)-2; i > 0; i--)
  {
    if (num[i] == '1')
    {
      n += pow(2, strlen(num) - i - 2);
    }
  }

  return n;
}

// Función para insertar un nodo al final de la lista
Nodo* insertarFinal(Nodo* cabeza, int id, long suma) {
    Nodo* nuevoNodo = (Nodo*)malloc(sizeof(Nodo));
    if (nuevoNodo == NULL) {
        fprintf(stderr, "Error: No se pudo asignar memoria al nodo\n");
        exit(1);
    }

    nuevoNodo->id = id;
    nuevoNodo->suma = suma;
    nuevoNodo->siguiente = NULL;

    if (cabeza == NULL) {
        // Si la lista está vacía, el nuevo nodo es la cabeza
        return nuevoNodo;
    }

    // Encontrar el último nodo de la lista
    Nodo* temp = cabeza;
```

```
    while (temp->siguiente != NULL) {
        temp = temp->siguiente;
    }
    temp->siguiente = nuevoNodo;

    return cabeza;
}

// Función para liberar la lista
void liberarLista(Nodo* cabeza) {
    Nodo* indice = cabeza;
    while (indice != NULL) {
        Nodo* aux = indice;
        indice = indice->siguiente;
        free(aux);
    }
}
```