HITO2 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 { // estructura para los argumentos del consumidor
    int id; // el id de cada hilo
    long *resultados; // el array de los resultados
} ConsumidorArgumentos;


// VARIABLES COMPARTIDAS
int TAM; // tamano del buffer circular
Buffer *bufferCircular; // buffer circular
sem_t hay_espacio; // semáforo que controla los espacios del buffer
circular
sem_t hay_dato; // semáforo que controla los datos del buffer circular
sem_t mutex; // semáforo que controla el acceso de los consumidores al
buffer circular
int contadorConsumidores = 0; // indice de los consumidores

void *productores(void *arg);
void *consumidores(void *arg);
bool es_binario(char str[]);
long binario_decimal(char *num);

int main(int argc, char **argv) {
    // -- VARIABLES DEL HITO 2 -- //

    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];
    ConsumidorArgumentos *argConsumidor;
    long *array_resultados;
    long suma_total =0;

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

    // comprobamos los argumentos de entrada
    if(argc != 5) {
        fprintf(stderr, "Argumentos incorrectos. Se necesitan 5
parametros para ejecutar el hito 2\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 este dentro del
rango permitido
    if(TAM < 10 || TAM > 1000) {
        fprintf(stderr, "El tamaño del buffer circular 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);
                    }

                // reserva de memoria para el array_resultados
                array_resultados = (long*)malloc(nhilos * sizeof(long));
                if (array_resultados == NULL) {
                    fprintf(stderr, "Error al asignar memoria al
array_resultados\n");
                    free(bufferCircular);
                    free(array_resultados);
                    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);
                    free(array_resultados);
                    exit(-1);
                }

                // incializamos el array de resultados
                for (int i = 0; i < nhilos; i++) {
                    array_resultados[i] = 0;
                }
```

```
                // INCIALIZACION DE SEMAFOROS
                sem_init(&hay_espacio, 0, TAM);
                sem_init(&hay_dato, 0, 0);
                sem_init(&mutex, 0, 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;
                    argConsumidor[i].resultados = array_resultados;
                    pthread_create(&tidc[i], NULL, consumidores,
&argConsumidor[i]);
                }

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

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

                // Mostramos la suma truncada de cada hilo consumidor y
la suma total
                for (int i = 0; i < nhilos; i++) {
                    printf("Hilo %d: %ld\n", i, array_resultados[i]);
                    suma_total = (suma_total + array_resultados[i]) %
(RAND_MAX / 2);
                    fflush(stdout);
                }

                printf("Suma total: %ld\n",suma_total);
                fflush(stdout);

                // liberamos memoria, cerramos los fichero y destruimos
los semaforos
                free(argConsumidor);
                free(array_resultados);
                free(bufferCircular);
                fclose(salida);
                sem_destroy(&hay_espacio);
                sem_destroy(&hay_dato);
                sem_destroy(&mutex);

                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 *resultados = args->resultados;
    long suma = 0;
    long numero_decimal;
    bool sigue = true;

    Buffer dato;

    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];
                contadorConsumidores = (contadorConsumidores + 1) % TAM;

                sem_post(&mutex);
                sem_post(&hay_espacio);

                if (dato.longitud == 32 && dato.cadena[0] != '1') { // si
el numero es de 32 bits y positivo se hace la suma truncada
                    numero_decimal = binario_decimal(dato.cadena);
                    suma = (suma + numero_decimal) % (RAND_MAX / 2);
            }
        }

    }

    // Guardar el resultado en el array de resultados
    resultados[id] = suma;
    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;
}
```