PROYECTO NO. 1

Introducción: El siguiente trabajo muestra utilizando un ejemplo el uso de los hilos y sus estados en un sistema operativo. El trabajo consiste en la lectura y conteo de caracteres del abecedario en un archivo .csv para posteriormente mostrar el resultado del conteo para cada carácter. El mismo se ha realizado en C con el fin de aplicar procesos a bajo nivel, que tenga una aplicación en la interfaz en terminal y familiarizarse con el lenguaje C al igual que comandos y uso en Linux.

Métodos

Para el proyecto se utilizaron diferentes librerías las cuales brindan servicios al sistema para su correcto funcionamiento. Estas estarán en el código del proyecto al igual que la declaración de variables y algunos métodos básicos, así como funciones estándar de hilos. Como el archivo que se utiliza es una tabla, es importante conocer la cantidad de columnas que tendrá para que el análisis que se de conozca cuantas veces debe de avanzar en una fila, el siguiente método hace la función descrita antes para luego analizar el archivo ya conociendo esta información.

```
int charcount(FILE *const fin,int cas)
{
    int c, count;
    char compare;
    count = 0;
     switch (cas)
      {

    case 1:
    for( ;; )
    {
        c = fgetc( fin );
        if( c == EOF || c == '\n' )
            break;
        ++count;
    }
    case 2:
    for( ;; )
    {
        c = fgetc( fin );

        if(c == 44) {++count;}
         //EOF equivale a fin de línea
        if( c == EOF || c == '\n' )
            break;
    }
        ++count;
       }

    return count;
}
```

El siguiente método es para que los hilos hagan la lectura del archivo, esto se realiza por medio de secciones lo cual permite que diferentes hilos trabajen al mismo tiempo para hacer el mismo procedimiento sin contar los mismos datos, tener un resultado más rápido y que a su vez sea beneficioso con los recursos que tenga el hardware para que este no sea ocioso ni que este sobrecargado. Inicia los hilos, los bloquea para que hagan solo una sección y luego los libera para que entre el siguiente a hacer el siguiente segmento.

```
//Metodo de threads que lee el archivo
void *Analisis_Threads(void *entrada)
{
    //Obtiene el valor de entrada del argumento del struct
    //almacena el valor de las variables globales de inicio de byte, fin byte y la posicion.
    double Posicion_Byte_Inicio = ((struct LeerBytes*)entrada)->Inicio_Bytes;
    double Posicion_Byte_Fin = ((struct LeerBytes*)entrada)->Fin_Bytes;
    int threadPosicion = ((struct LeerBytes*)entrada)->posicion;

    char Varptr[1024];
    char c;
    double conteoInterno[255]; //ASCII
    int Varnmemb = 1024;
    //bloquea el archivo para que solo un thread lo lea a la vez
    //Se crearan n threads, por lo que cada thread leera cierta porcion del archivo, se le asignara en que
posicion de bytes comience y en que posicion de bytes finalice.

   /*Se utiliza la funcion pthread_mutex_lock para que unicamente un thread pueda realizar una accion a la vez y
no realicen los n threads la misma tarea, sino que cada uno realice una tarea cuando
   el thread n termine el n+1 comience y asi, esto se hace para que no cuente los mismos datos */
    for (double i = Posicion_Byte_Inicio; i <= Posicion_Byte_Fin; i += 1024)
         {
         if((i+ 1024)> Posicion_Byte_Fin)
         {
         Varnmemb = (Posicion_Byte_Fin - i);
         }
```

```
          else {
          Varnmemb = 1024;
          }

          //////*CRITICAL SECTION*/////////
          //hacemos lock con mutex para que ningun otro thread acceda a esta región de codigo
         pthread_mutex_lock(&lock);

          /*
          fseek() reposiciona un puntero de tipo FILE
          void fseeko(FILE * steam, long offset, int whence)

          //PARAMETROS
          stream: Objeto de tipo FILE que identifica la secuencia
          offset: Es la nueva posicion del archivo
          whence: Es la posicion desde donde se agrega el desplazamiento --> Utilizamos SEEK_SET: Posición respecto
al inicio del archivo
          */
             if (fseek(ArchivoEntrada, i, SEEK_SET) == 0)
                   {
                            /*
                            fread() lee los datos del stream dado al array apuntado por el ptr(Varptr)
                            size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)

                            //PARAMETROS
                            ptr – es un puntero a un bloque de memoria con un tamaño minimo de *nmemb bytes
                            size – es el tamaño de bytes que cada elemeto va a leer
                            nmemb – es el numero de elementos, cada uno con un tamaño de size bytes
                            stream – puntero a un objeto FILE
                            */
                     fread(&Varptr, 1, Varnmemb, ArchivoEntrada);

                              //se desbloquea, el thread le pasa la llave a otro thread para acceder a esta region
de codigo
                    pthread_mutex_unlock(&lock);
                     } else
                            {
                                     //ERROR
                            }

         // Realiza la cuenta
         for (int i = 0; i <= Varnmemb; i++)
          {
             conteoInterno[(int) ((char) Varptr[i])]++;
         }
     }
         for (int i = 0; i <= 255; i++)
          {
             conteo[i] += conteoInterno[i];
         }

          //Almacena en el arreglo de finalizados las banderas en que los threads acabaron su proceso.
         Finalizados[threadPosicion] = true;
         bool flagTerminados = true;
         for (int i = 0; i < Cantidad_Threads; i++)
          {
             //double free or corruption (!prev) -error
             if (!Finalizados[i]) flagTerminados = false;
         }
```

Por último, se tiene el código Main que al inicio hace la carga del archivo con su path y realiza los diferentes métodos para su manejo, se tiene el peso del archivo en GB y una instrucción para inicializar el programa para que haga los diferentes métodos y utilice las diferentes variables que son necesarias para correr los métodos y el programa.

```
int main() {

< * obtención del archivo en el computados y algunas funciones para obtener las variables deseadas. * >

  FILE *fin;
    fin = fopen(ubicacion, "r" );
    Bytes_Leer = charcount(fin,1)-1;
    conteo2 = charcount(fin,2);
    system("clear");

    // Abre el archivo en modo read only ("r)
    ArchivoEntrada = fopen(ubicacion, "r");

    //fseek() reposiciona un puntero de tipo FILE
    //void fseeko(FILE * steam, long offset, int whence)
```

```
    //whence - SEEK_END: Fin del archivo
    fseek(ArchivoEntrada, 0LL, SEEK_END);

    //Obtiene el peso del archivo
    //ftell() returna el la posicion del archivo actual dado el stream proporcionado (ArchivoEntrada)
      double fileSize = ftell(ArchivoEntrada);
      printf("El archivo con ruta %s pesa: %lf GB (%f Bytes)\n",ubicacion, fileSize / 1e+9,fileSize);
      printf("\n");
      printf("Presione ENTER para empezar el conteo\n");
      printf("-------------------------------------------------------------------------\n");
      char sc;
      scanf("%c",&sc);
      //empieza el timer
      start_t = clock();
      printf("¡Espere a que el conteo haya finalizado!\n");
      Bytes_Leer = fileSize;

    //Creamos una variable de tipo struct y un contador para el ciclo while para validar con la cantida de threads
    int i = 0;
    struct LeerBytes *Struct_Main;
    double lineaInicio = Linea_Inicio;
    Inicio_Leer = Bytes_Leer;

    //double ceil(double x)
    //ceil() retorna el entero mas pequeño que sea mayor o igual al parametro dado (Bytes_Leer /
(Cantidad_Threads))
    double bytesLeidos_Thread = ceil(Bytes_Leer / (Cantidad_Threads));
    //Variable que indicara cuando se encuentre el tope del archivo
    double lineaFin = lineaInicio + bytesLeidos_Thread;

    while (i < Cantidad_Threads)
         {
         //int contador = 0;
         // Crea el struct con la cantidad de lineas

         //void *malloc(size_t size)
        //Malloc() asigna la memoria solicitada y retorna un puntero
        Struct_Main = (struct LeerBytes *)malloc(sizeof(struct LeerBytes));
        Struct_Main->Inicio_Bytes = lineaInicio;
        Struct_Main->Fin_Bytes = lineaFin;
        Struct_Main->posicion = i;
        //se crean threads, parametros(arreglo de instancia de  threads, NULL, proceso de thread, parametros del
proceso de thread)
        pthread_create(&threads[i], NULL, Analisis_Threads, Struct_Main);


         lineaInicio = lineaFin + 1;
        lineaFin = lineaInicio + bytesLeidos_Thread;

         if (lineaFin >= Bytes_Leer)
         {
            lineaFin = Bytes_Leer;

        }
```

Comandos:

Compilar proyecto: gcc -o Proyecto Proyecto.c -lcruses -lpthread -lm

Obtener #Proceso : ps -u iban -efl

Obtener threads: cat /proc/#Proceso/status

Ver en la terminal rendimento del cpu al realizar proceso: ps -p #Proceso -L -o pid,tip,pcpu,pmem

Conclusiones

- El uso de mutex en los hilos permite que una sección del código permitiendo así que solo uno se encuentre haciendo ese paso para no repetir ni leer caracteres repetidos es decir que lo lea dos veces.
- Se determino que al hacer proceso con la utilización de hilos se puede reducir el tiempo por lo que se maximiza los recursos.
- Al utilizar hilos se regulan procesos para facilitar el control y manejo de la data.

Referencias

- Abraham Silberschatz 2010 - Fundamentos de Sistemas Operativos. Publisher: MC GRAW HILL; edition (2010) Language: Spanish ISBN-10: 8448146417 ISBN-13: 978-8448146412
- Stallings, W. (2005). sistemas operativos aspectos internos y principios de diseño (5th ed.). Madrid: Person.