Descripción general:

El proyecto está dividido en dos partes, una correspondiente a procesos y otra a hilos. El main determina que modo fue seleccionado por el usuario y pasa el control al main_procesos() si el modo es 0 o 1, o al main_hilos() si el modo es el 2. En ambos casos la estructura del codigo siguiente a ejecutarse tendrá la misma estructura, difieren en los mecanismos de comunicación empleados.

Caso I. Procesos

Para establecer la comunicación entre el proceso maestro y los procesos esclavos se utilizaron **pipes**. Se crean dos pipies, el primero para que el maestro escriba y los esclavos lean, este se utiliza para enviar las tareas que los esclavos deben realizar. En el segundo pipe los esclavos escriben y el maesto lee, se utiliza para enviar los resultados de las tareas realizadas.

La información escrita en el pipe sigue un formato, el cual se describe a continuación:

- Si es una tarea que debe realizarse, es decir el mestro la envía a los esclavos:

Instrucción (char)	String (string)	Tamaño del nombre del archivo(int)	Nombre del archivo (string)

Instruccion: es un char. Toma los siquientes valores:

- DWC 0
- DGREP 1
- DQUIT 2

String: Corresponde al string que debe ser buscado en cada archivo, cuando la instrucción es DGREP. En los demás casos el campo no es utilizado.

Nombre del archivo es un string que contiene la ruta absoluta de cada archivo.

- Si es un resultado:

<table>
<tr><td rowspan="2">Tamaño del nombre del archivo (int)</td><td rowspan="2">Nombre del archivo (string)</td><td rowspan="2">Error (int)</td><td>Número de líneas (int)</td><td>Número de palabras (int)</td></tr>
<tr><td colspan="2">Contiene (bool)</td></tr>
</table>

- Resultado de DWC: Se utiliza el campo número de líneas y número de palabras
- Resultado de DGREP: Se utiliza un booleano *contiene* para indicar si el archivo contiene o no la palabra.
- Si ocurrió un error al tratar de analizar el archivo se coloca *error* como 1 y el último campo no se utiliza.

Para la sincronización de la comunicación se utilizaron dos mutex, a los cuales se les hace referencia en el código como mutex[READ] y mutex[WRITE]. Para que el maestro y los esclavos puedan acceder a los mutex se crearon en un espacio de memoria compartida llamada “/memoria_de_procesos”.

Para evitar situaciones de interbloqueo los trabajos se envían y se reciben en grupos de n que es igual al nivel de concurrencia.

El modo 0 y el modo 1 utilizan el mismo esquema, difieren en las funciones wc()/grep() y

wc_sist()/grep_sist(). En las últimas dos, que se utilizaron en el modo 2, fue necesario realizar un redireccionamiento de la salida estandar del porceso hijo que crea cada esclavo hacia un pipe que posteriormente lee el esclavo, extrae la información y envía al maestro. Para esto se empleó la función dup2().

Los resultados que el maestro recibe a través del pipe se almacenan en dos listas, una llamada *datos* y otra *errores.* En está ultima se guardan los nombres de los archivos que llegaron correctamente a los esclavos, pero al momento de llamar a las funciones wc()/grep() o wc_sist()/grep_sist() ocurrió un error (puede ser al momento de abrir el archivo, leerlo, cerrarlo, entre otros). Los errores que ocurren durante la rutina de los esclavos, por ejemplo leer o escribir del pipe y bloquear o desbloquear el mutex, son irrecuperables. En estos caso, los esclavos cierran los extremos de los pipes que ellos utilizan y finalizan su ejecución. Cuando el maestro intente leer o escribir de uno de los dos pipes la función le devolverá 0 que indica que se perdió la comunicación con los esclavos. Cuando ocurre lo anterior el padre sale del ciclo infinito del shell y hace wait de los esclavos, el estatus devuelto indicará porque finalizó la comunicación.

Caso II. Hilos

En el modo de hilos se utilizó variables globales para la comunicación. El maestro es el programa principal y crea los n hilos esclavos. Para la transmision de tareas y resultados se emplean dos colas (t_pendientes y t_listas) y otra (t_errores) para manejar errores. Además, se define un mutex por cada lista para controlar que un solo proceso la modifique, los cuales se inicializan con los valores por defecto. También se implementaron dos semáforos, uno para indicar cuando hay datos disponibles en la cola de t_pendientes (*sem*) y otro para indicar que los trabajadores finalizaron todas las tareas (*listo*), ambos se inicializan con valor 0.

Para enviar la información sobre qué tarea deben ejecutar los esclavos y de qué tipo es el resultado que se envía al padre, los nodos de las lista poseen un apuntador a voir llamado *datos* que puede ser dos tipos:

```
struct tareaH_struct{
        char instruccion;
        char archivo[MAX_NOMBRE];
        char string [MAX_STRING];
};
typedef struct tareaH_struct tarea;
```

```
struct resultado_struct{
        char archivo[MAX_NOMBRE];
        int num_lineas;
        int num_palabras;
};
typedef struct resultado_struct resultado;
```

La estructura *tarea* es la empleada por el maestro para enviar la información pertinente a los esclavos. Dependiendo del valor de *instrucción* (utiliza los mismo valores que en el caso de procesos) los campos válidos varían. Si es DWC el campo string no es utilizado y si es DQUIT solo el campo *instrucción* es válido. En cuanto a la estructura *resultado* también depende del tipo de instrucción.

En el caso de que ocurra un error en las funciones grep() y wc() (puede ser al abrir, leer o cerrar el archivo) se crea un nodo con datos tipo *resultado* y se agrega a la cola de *t_errores*. Si ocurre un error irrecuperable en la ejecución de los hilos (puede ser al bloquear o desbloquear los semáforos o falta de memoria) entonces el hilo termina el proceso.