**Instituto Tecnológico de Costa Rica**

**Escuela de Computación**

**Curso: IC6600 - Principios de Sistemas Operativos**

**Proyecto 1**

**Profesor: Armando Arce Orozco**

Maria Félix Mendez Abarca – 2022438535

Alexander Brenes Garita - 2018191805



## INTRODUCCIÓN

AQUI

## DESCRIPCIÓN DEL PROBLEMA

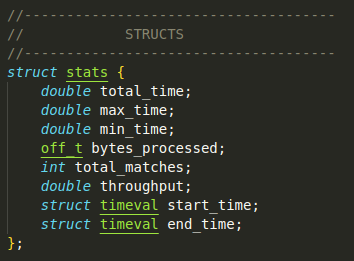
AQUI

## DEFINICIÓN DE ESTRUCTURA DE DATOS

El programa utiliza tres estructuras de datos principales para organizar y comunicar la información:

### 1. struct stats

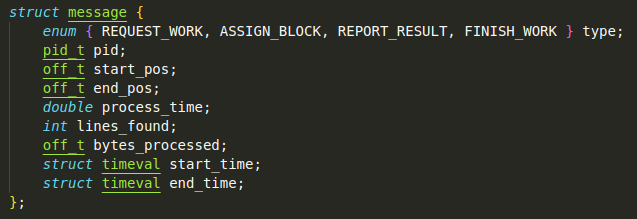
Guarda las estadísticas globales del rendimiento del programa:



**Propósito:**  
Registrar métricas como tiempo total, tiempo mínimo y máximo por bloque, volumen de datos procesados y rendimiento promedio (MB/s).

### 2. struct message

Estructura usada para la comunicación entre procesos mediante **pipes**:



**Propósito:**  
Transmitir información sobre el trabajo asignado, los resultados parciales y los tiempos de ejecución de cada proceso.

## DESCRIPCIÓN DETALLADA

El programa se ejecuta con la siguiente sintaxis:

*./psoPr "palabra1|palabra2|palabra3" archivo.txt 4*

Donde:

* La cadena entre comillas corresponde a la **expresión regular** de búsqueda.
* **archivo.txt** es el archivo sobre el cual se realizará la búsqueda.
* **4** es el número de procesos en el pool.

### Flujo general:

1. **Inicialización:**  
   El proceso principal (padre) abre el archivo y prepara los pipes de comunicación con los procesos hijos.
2. **Creación del pool:**  
   Se crean N procesos hijos mediante **fork()**. Cada hijo solicita bloques de trabajo al proceso padre.
3. **Asignación de bloques:**  
   El padre distribuye bloques de **8192 bytes** a los procesos hijos conforme estos los solicitan.
4. **Procesamiento:**  
   Cada proceso busca coincidencias usando la librería **regex.h** (**regcomp** y **regexec**) sobre el bloque recibido.
5. **Sincronización:**  
   Los hijos envían los resultados al padre mediante mensajes, evitando conflictos en la lectura y escritura.
6. **Registro de resultados:**  
   Cada bloque procesado se registra en **grep\_log.csv** con información sobre el PID, posición del bloque, tiempos y coincidencias.
7. **Cálculo de rendimiento:**  
   Al finalizar, el proceso principal llama a la función **analizar\_rendimiento()** para generar estadísticas globales y escribir el resumen en **performance\_summary.csv.**

## EXPLICACIÓN DE LOS COMPONENTES

**1. Lectura controlada del archivo:**

El archivo se lee secuencial mente en bloques de 8192 bytes.

Cada proceso solicita al padre la siguiente posición disponible mediante un mensaje **REQUEST\_WORK**.

**2. Búsqueda con expresiones regulares**

Cada bloque se analiza con las funciones de la librería **regex.h**

Si se detecta una coincidencia, el proceso imprime la línea correspondiente en la salida estándar.

### 3. ****Comunicación padre–hijo****

Los procesos se comunican mediante **pipes sincrónicos**:

* El **hijo** envía peticiones (**REQUEST\_WORK**, **REPORT\_RESULT**).
* El **padre** responde con mensajes de tipo (**ASSIGN\_BLOCK**, **FINISH\_WORK**).

Este mecanismo evita la lectura duplicada y garantiza la correcta distribución de trabajo.

### 4. ****Medición del rendimiento****

Cada proceso mide el tiempo que tarda en procesar su bloque con **gettimeofday()**.  
El proceso principal calcula métricas globales como:

* Tiempo total de ejecución
* Throughput (MB/s)
* Eficiencia de paralelización

Los resultados se escriben en **performance\_summary.csv.**

## MECANISMOS DE CREACIÓN Y COMUNICACIÓN DE PROCESOS

**1. Creación:** Como se ha visto en clase, se usa for anidado con el numero de proceso que queremos usar y luego mediante la función **fork()** crea los hijos



**2. Comunicación (pipes):**

Se establecen dos pipes:

* **request\_pipe[]:** para que los hijos soliciten trabajo.
* **assign\_pipe[]:** para que el padre asigne bloques y confirme trabajo.

**3. Sincronización:**

El padre mantiene una variable global de posición (**file\_pos**) que se actualiza cada vez que un bloque de 8192 bytes es leído completamente.  
Se usa **fseek()** para mover el puntero del archivo y evitar lecturas superpuestas.

**4. Finalización:**

Cuando se alcanza el final del archivo, el padre envía un mensaje **FINISH\_WORK** a cada proceso hijo.  
Luego espera su finalización con **wait(NULL)** y genera el resumen de rendimiento.

## ANALISIS DE RESULTADOS

AQUI

## CONCLUSIONES

AQUI