==
=======

PREGUNTAS Y RESPUESTAS DEL PROYECTO
Sistema de Etiquetado Automático de Mangos

==
=======

Autor: Sebastian Escudero
Fecha: Noviembre 2025

==
=======

PREGUNTAS GUÍA DEL PROYECTO

==
=======

1. ¿Algún parámetro de operación se puede escribir en términos de W?

Sí, varios parámetros están relacionados con W (longitud de banda):

a) POSICIÓN DE LOS ROBOTS:
   La posición de cada robot se calcula en función de W:

   separacion = W / num_robots
   posicion[i] = separacion * (i + 0.5)

   Ejemplo: Si W=200cm y tenemos 4 robots:
   - separacion = 200/4 = 50cm
   - Robot 0: posición = 50 * 0.5 = 25cm
   - Robot 1: posición = 50 * 1.5 = 75cm
   - Robot 2: posición = 50 * 2.5 = 125cm
   - Robot 3: posición = 50 * 3.5 = 175cm

b) TIEMPO TOTAL DE SIMULACIÓN:
   tiempo_total = (W + Z) / X

   Donde:
   - W es la longitud de banda
   - Z es el tamaño de la caja
   - X es la velocidad de la banda

   La caja debe recorrer W más su propio tamaño Z para salir
completamente.

c) ZONA DE TRABAJO POR ROBOT:
   Si los robots están equidistantes, cada uno tiene:
   zona_trabajo = W / num_robots

   Aunque la zona efectiva donde puede trabajar es min(zona_trabajo, Z)

2. ¿Es una simplificación aceptable que la distancia entre cajas y ejes sea la misma?

SÍ, es una simplificación aceptable y realista por las siguientes razones:

a) DISTRIBUCIÓN UNIFORME EN PRODUCCIÓN REAL:
   En líneas de producción automatizadas reales, los robots se instalan con
   separación uniforme para maximizar eficiencia y minimizar costos.

b) OPTIMIZACIÓN DE RECURSOS:
   Tener separación uniforme significa que:
   - Todos los robots tienen la misma carga de trabajo esperada
   - No hay "cuellos de botella" en zonas específicas
   - El mantenimiento es más sencillo (robots idénticos en configuración)

c) SIMPLIFICACIÓN MATEMÁTICA:
   Facilita el cálculo de:
   - Número óptimo de robots
   - Tiempo de procesamiento esperado
   - Análisis de costo-efectividad

d) IMPLEMENTACIÓN EN EL CÓDIGO:
   Ver función calcular_posiciones_robots() en mango_core.c:

```
   void calcular_posiciones_robots(EstadoSistema *estado, float
longitud_banda,int num_robots) {
       if (num_robots <= 0) return;

       float separacion = longitud_banda / num_robots;
       for (int i = 0; i < num_robots; i++) {
           estado->posiciones_robot[i] = separacion * (i + 0.5);
       }
   }
```

CONCLUSIÓN: Esta simplificación es no solo aceptable sino preferible para un sistema real, ya que representa la mejor práctica en diseño de líneas de producción automatizadas.

3. ¿Cuánto tiempo tiene un robot para etiquetar al considerar el número de mangos?

El tiempo disponible para cada robot NO depende directamente del número de mangos, sino de la GEOMETRÍA y VELOCIDAD del sistema:

a) TIEMPO DISPONIBLE POR ROBOT:
   tiempo_disponible = Z / X

Donde:
- Z = tamaño de la caja (cm)
- X = velocidad de la banda (cm/s)

Ejemplo: Si Z=50cm y X=10cm/s:
tiempo_disponible = 50/10 = 5 segundos

b) POR QUÉ NO DEPENDE DEL NÚMERO DE MANGOS:
Cada robot tiene una ventana de tiempo fija que es el tiempo que tarda la caja en pasar por su zona de trabajo. Durante ese tiempo, el robot intenta etiquetar tantos mangos como pueda.

c) RESTRICCIÓN REAL:
El número de mangos que UN robot puede etiquetar está limitado por:

mangos_max_por_robot = tiempo_disponible / tiempo_etiquetado_promedio

Donde tiempo_etiquetado depende de la distancia del mango al centro.

d) IMPLEMENTACIÓN EN EL CÓDIGO:
Ver función proceso_robot() en mango_core.c:

```
float tiempo_disponible = (fin_zona - pos_caja) / config-
>velocidad_banda;

if (tiempo_etiquetado <= tiempo_disponible) {
    // Etiquetar el mango
}
```

e) EJEMPLO NUMÉRICO:
Si Z=50cm, X=10cm/s, velocidad_robot=5cm/s:
- Tiempo disponible = 50/10 = 5s
- Mango a 5cm del centro: tiempo = 2*5/5 = 2s → OK
- Mango a 15cm del centro: tiempo = 2*15/5 = 6s → NO alcanza

Por eso limitamos los mangos a 7cm del centro en el código.

4. ¿Cuánto tiempo como máximo se demoraría un robot en etiquetar un mango?

El tiempo máximo se calcula con la fórmula:

tiempo_max = (2 * distancia_max) / velocidad_robot

Donde:
- velocidad_robot = Z / 10
- distancia_max en nuestro sistema = 7cm (límite impuesto)

a) CÁLCULO PASO A PASO:

Para Z = 50cm:

- velocidad_robot = 50/10 = 5 cm/s
- distancia_max = 7cm
- tiempo_max = (2 * 7) / 5 = 14/5 = 2.8 segundos

b) POR QUÉ FACTOR 2:
El robot debe:
1. Ir desde el centro hasta el mango (distancia)
2. Regresar al centro (distancia)

Total = 2 * distancia

c) POR QUÉ LIMITAMOS A 7CM:
Si permitimos mangos hasta 23cm (mitad de Z=50):
- tiempo_max = (2 * 23) / 5 = 9.2 segundos
- tiempo_disponible = 50/10 = 5 segundos
- 9.2s > 5s → NO ALCANZA

Con límite de 7cm:
- tiempo_max = 2.8 segundos
- tiempo_disponible = 5 segundos
- 2.8s < 5s → SÍ ALCANZA (con margen)

d) IMPLEMENTACIÓN EN EL CÓDIGO:
Ver función calcular_tiempo_etiquetado() en mango_core.c:

```
float calcular_tiempo_etiquetado(Mango *mango, float tamano_caja) {
    float distancia = sqrt(mango->x * mango->x + mango->y * mango->y);
    float velocidad_robot = tamano_caja / 10.0;
    return (2.0 * distancia) / velocidad_robot;
}
```

e) VELOCIDAD SI EL BRAZO SE MUEVE A LA MISMA VELOCIDAD LINEAL QUE LA CAJA:
Si velocidad_robot = velocidad_banda = X:

Para X=10cm/s, distancia_max=7cm:
- tiempo_max = (2 * 7) / 10 = 1.4 segundos

Esto sería MUCHO MÁS RÁPIDO pero el enunciado especifica que el robot se mueve a Z/10, no a X.

5. ¿Cómo expresar el número de mangos posibles de etiquetar en función de N?

El número de mangos que PUEDEN ser etiquetados depende de varios factores:

a) FÓRMULA TEÓRICA MÁXIMA:

mangos_posibles = num_robots * (tiempo_disponible / tiempo_etiquetado_min)

Donde:
- num_robots = número de robots activos
- tiempo_disponible = Z / X
- tiempo_etiquetado_min = tiempo para el mango más cercano al centro

b) EN LA PRÁCTICA:
No todos los mangos estarán en posición óptima, por lo que:

mangos_esperados ≈ num_robots * (tiempo_disponible / tiempo_etiquetado_promedio)

Con tiempo_etiquetado_promedio calculado experimentalmente.

c) RELACIÓN CON N (número de mangos en la caja):

Si N <= mangos_posibles:
    Todos los N mangos pueden ser etiquetados

Si N > mangos_posibles:
    Solo mangos_posibles serán etiquetados (fallo)

d) NÚMERO ÓPTIMO DE ROBOTS PARA N MANGOS:
De nuestros experimentos (ver curva generada):

num_robots_optimo ≈ ceiling(N / 2)

Para banda de 300cm. La relación es aproximadamente lineal.

e) IMPLEMENTACIÓN PRÁCTICA:
El sistema de análisis (modo 1) encuentra empíricamente el número óptimo de robots ejecutando múltiples simulaciones:

./mango_analysis 1 <N> <simulaciones>

Esto prueba con 1, 2, 3, ... robots hasta lograr 95% de tasa de éxito.

f) EJEMPLO NUMÉRICO:
Para Z=50cm, X=10cm/s, velocidad_robot=5cm/s, distancia_promedio=5cm:

- tiempo_disponible = 50/10 = 5s
- tiempo_etiquetado_promedio = (2*5)/5 = 2s
- mangos_por_robot = 5/2 = 2.5 → ~2 mangos
- Para N=10 mangos: robots_necesarios = 10/2 = 5 robots

Esto coincide con nuestros resultados experimentales.

6. ¿Cómo incluir la variación de N?

La variación de N está incluida en el sistema de varias formas:

a) EN EL SIMULADOR PRINCIPAL:
   El parámetro num_mangos es configurable:

   ./mango_simulator 10 50 200 4 <N>

   Donde N puede ser cualquier valor de 1 a MAX_MANGOS (50)

b) EN EL SISTEMA DE ANÁLISIS - MODO 1:
   Analiza un valor específico de N:

   ./mango_analysis 1 <N> <simulaciones>

   Ejemplo:
   ./mango_analysis 1 20 5

   Esto encuentra cuántos robots se necesitan para N=20 mangos.

c) EN EL SISTEMA DE ANÁLISIS - MODO 2:
   Analiza un RANGO de valores de N:

   ./mango_analysis 2 <N_min> <N_max> <incremento> <sims>

   Ejemplo:
   ./mango_analysis 2 10 30 5 3

   Esto genera una curva para N = 10, 15, 20, 25, 30 mangos
   Cada punto de la curva muestra cuántos robots se necesitan.

d) GENERACIÓN ALEATORIA DENTRO DE CADA SIMULACIÓN:
   Aunque fijamos N (cantidad), las POSICIONES de los N mangos varían:

```
for (int i = 0; i < num_mangos; i++) {
    estado->mangos[i].x = random(-7.0, 7.0);
    estado->mangos[i].y = random(-7.0, 7.0);
}
```

   Esto simula la variabilidad natural del empaque de mangos.

e) RANGO SOLICITADO: N a 1.2N
   El enunciado pide analizar de N a 1.2N mangos.

   Por ejemplo, si la configuración base es N=20:
   - N_min = 20
   - N_max = 1.2 * 20 = 24
   - incremento = 1 o 2

   Comando:
   ./mango_analysis 2 20 24 1 5

   Esto analizará 20, 21, 22, 23, 24 mangos con 5 simulaciones cada uno.

f) ARCHIVO CSV GENERADO:
   El archivo curva_robots_mangos.csv contiene:

```
   NumMangos,RobotsMínimos,TasaÉxito,TiempoPromedio
   20,4,1.000,0.041
   21,4,0.967,0.043
   22,5,1.000,0.044
   23,5,0.967,0.045
   24,5,1.000,0.047

   De aquí se puede deducir fácilmente el punto costo-efectivo.


================================================================================
=======
EXPLICACIÓN DEL FUNCIONAMIENTO COMPLETO DEL SISTEMA
================================================================================
=======

1. ARQUITECTURA GENERAL
--------------------------------------------------------------------------------
-------

El sistema consta de dos programas principales:

a) MANGO_SIMULATOR: Simula una operación de etiquetado
b) MANGO_ANALYSIS: Ejecuta múltiples simulaciones y genera análisis

Ambos usan el mismo motor de simulación (mango_core.c) que implementa
la lógica de IPC y coordinación de robots.


2. FLUJO DE EJECUCIÓN DEL SIMULADOR
--------------------------------------------------------------------------------
-------

PASO 1: INICIALIZACIÓN
   - Leer parámetros de línea de comandos (X, Z, W, num_robots, N)
   - Validar que sean valores positivos y dentro de límites
   - Configurar handlers de señales (SIGINT, SIGTERM)

PASO 2: CREACIÓN DE IPC
   - Crear memoria compartida con shm_open()
   - Configurar tamaño con ftruncate()
   - Mapear a espacio de direcciones con mmap()
   - Crear semáforo mutex con sem_open()

PASO 3: INICIALIZACIÓN DEL SISTEMA
   - Calcular posiciones de robots (equidistantes en W)
   - Generar N mangos con posiciones aleatorias
   - Marcar todos los mangos como no etiquetados
   - Configurar estado inicial (posicion_caja = 0, simulacion_activa = 1)

PASO 4: CREACIÓN DE PROCESOS ROBOT
   - Para cada robot (0 a num_robots-1):
     * fork() crea proceso hijo
     * Proceso hijo ejecuta proceso_robot()
```

```
     * Proceso padre guarda PID del hijo

PASO 5: BUCLE DE MOVIMIENTO DE BANDA (proceso principal)
   - Calcular tiempo_total = (W + Z) / X
   - Dividir en pasos pequeños (dt = 0.05s)
   - En cada paso:
     * Adquirir mutex
     * Actualizar posicion_caja += X * dt
     * Simular fallas de robots (si redundancia habilitada)
     * Verificar si caja_completada
     * Liberar mutex
     * Dormir dt segundos
   - Continuar hasta que:
     * La caja salga de la banda (posicion > W + Z), O
     * Todos los mangos estén etiquetados

PASO 6: PROCESO DE CADA ROBOT (procesos hijos)
   - Calcular zona de trabajo [inicio_zona, fin_zona]
   - Bucle principal:
     * Esperar a que la caja entre en zona
     * Mientras caja en zona:
       - Adquirir mutex
       - Buscar mango no etiquetado y sin asignar
       - Si encuentra uno:
         · Calcular tiempo_etiquetado
         · Verificar si tiempo_etiquetado <= tiempo_disponible
         · Si sí: marcar como asignado, liberar mutex
         · Dormir durante tiempo_etiquetado (simular trabajo)
         · Adquirir mutex, marcar como etiquetado, liberar mutex
       - Si no encuentra: liberar mutex, esperar un poco
     * Cuando caja sale de zona:
       - Verificar si todos etiquetados
       - Si sí: terminar
   - Imprimir "Finalizando operación"
   - Terminar proceso (exit)

PASO 7: ESPERA Y CONTEO (proceso principal)
   - Marcar simulacion_activa = 0 (señal para robots)
   - Esperar a que todos los procesos robot terminen (waitpid)
   - Contar cuántos mangos fueron etiquetados
   - Imprimir resultado

PASO 8: LIMPIEZA
   - Liberar memoria compartida (munmap)
   - Cerrar descriptores (close)
   - Eliminar recursos IPC (shm_unlink, sem_unlink)
   - Retornar código de éxito/fallo


3. PROTOCOLO DE SINCRONIZACIÓN
-------------------------------------------------------------------------
-------

El protocolo previene condiciones de carrera usando un semáforo mutex:
```

```
SECCIÓN CRÍTICA 1: Asignación de mango
   sem_wait(mutex);              // Entrar a sección crítica
   buscar mango disponible
   if (encontrado) {
       marcar mango.robot_asignado = mi_id
   }
   sem_post(mutex);              // Salir de sección crítica

TRABAJO FUERA DE SECCIÓN CRÍTICA:
   sleep(tiempo_etiquetado);      // No bloquea a otros robots

SECCIÓN CRÍTICA 2: Marcado como etiquetado
   sem_wait(mutex);              // Entrar a sección crítica
   mango.etiquetado = 1
   sem_post(mutex);              // Salir de sección crítica

Este protocolo garantiza:
- Solo un robot puede asignar un mango a la vez
- No hay condiciones de carrera
- El trabajo (sleep) ocurre en paralelo


4. SISTEMA DE ANÁLISIS
-------------------------------------------------------------------------
-------

MODO 1: Encontrar robots óptimos para N mangos
   - Input: N (número de mangos), num_simulaciones
   - Algoritmo:
     for robots = 1 to MAX_ROBOTS:
         ejecutar num_simulaciones
         calcular tasa_exito
         if tasa_exito >= 95%:
             return robots
   - Output: Número mínimo de robots necesarios

MODO 2: Generar curva robots vs mangos
   - Input: N_min, N_max, incremento, num_simulaciones
   - Algoritmo:
     for N = N_min to N_max (paso incremento):
         robots = encontrar_robots_optimos(N)
         guardar en CSV
   - Output: Archivo curva_robots_mangos.csv
   - Bonus: Genera gráfico con gnuplot si está instalado

MODO 3: Análisis de redundancia
   - Input: N, robots_base, prob_fallo, num_simulaciones
   - Algoritmo:
     for robots = robots_base to robots_base+5:
         ejecutar simulaciones con prob_fallo
         calcular tasa_exito
         if tasa_exito >= 95%:
             return robots
```

- Output: Archivo analisis_redundancia.csv

5. CÁLCULOS MATEMÁTICOS CLAVE

POSICIÓN DE ROBOTS:
```
separacion = W / num_robots
posicion[i] = separacion * (i + 0.5)
```

ZONA DE TRABAJO DE ROBOT i:
```
inicio_zona = posicion[i] - Z/2
fin_zona = posicion[i] + Z/2
```

TIEMPO DISPONIBLE:
```
tiempo_disponible = (fin_zona - posicion_caja) / X
```

TIEMPO DE ETIQUETADO:
```
distancia = sqrt(x² + y²)
velocidad_robot = Z / 10
tiempo_etiquetado = (2 * distancia) / velocidad_robot
```

CONDICIÓN PARA ETIQUETAR:
```
tiempo_etiquetado <= tiempo_disponible
```

OTRAS PREGUNTAS POSIBLES

Q1: ¿Por qué usar procesos (fork) en lugar de threads (pthread)?

RESPUESTA:
Los procesos son más apropiados para esta simulación porque:

a) AISLAMIENTO: Cada robot es una entidad independiente en la realidad
b) ROBUSTEZ: Si un proceso falla, no afecta a los demás
c) REALISMO: Simula mejor robots físicos independientes
d) SIMPLICIDAD: IPC es explícito y claro
e) APRENDIZAJE: Demuestra uso de fork, IPC POSIX, sincronización de procesos

Los threads serían más eficientes en memoria pero menos realistas para simular robots físicos independientes.

Q2: ¿Cómo se garantiza que no haya condiciones de carrera?

RESPUESTA:
Se usan tres mecanismos:

a) SEMÁFORO MUTEX: Protege todas las operaciones sobre estado compartido
b) PROTOCOLO DE DOS FASES:
   - Fase 1: Asignar mango (dentro de mutex)
   - Fase 2: Trabajar (fuera de mutex)
   - Fase 3: Marcar completado (dentro de mutex)
c) CAMPO robot_asignado: Previene que dos robots tomen el mismo mango

Verificado mediante:
- Análisis del código
- Múltiples ejecuciones sin duplicados
- Nunca se observó un mango etiquetado dos veces

Q3: ¿Qué pasa si un robot falla durante la operación?

RESPUESTA:
El sistema maneja fallas mediante:

a) DETECCIÓN: Campo robots_fallados[i] marca robots que fallaron
b) SIMULACIÓN: Con prob_fallo > 0, robots pueden fallar aleatoriamente
c) CONTINUACIÓN: Robots restantes continúan trabajando
d) REDUNDANCIA: Modo 3 del análisis determina cuántos robots extra necesitas

Ejemplo: Si tienes 5 robots y 1 falla:
- 4 robots continúan operando
- Algunos mangos pueden no ser etiquetados
- Análisis muestra cuántos robots extra necesitas para tolerar fallas

Q4: ¿Cómo se limpia la memoria compartida si el programa se interrumpe?

RESPUESTA:
Mediante signal handlers:

```
void signal_handler(int signo) {
    estado_compartido->simulacion_activa = 0;  // Señal a robots
    cleanup_recursos();                         // Limpia IPC
    exit(0);
}
```

Registrados para:
- SIGINT (Ctrl+C)
- SIGTERM (kill)

Además, el Makefile tiene un target:

make clean-ipc

Que elimina recursos huérfanos manualmente.

Q5: ¿El sistema es escalable a más robots o más mangos?

RESPUESTA:
Sí, es escalable con limitaciones:

a) LÍMITES CONFIGURABLES:
   #define MAX_ROBOTS 20
   #define MAX_MANGOS 50

   Pueden aumentarse recompilando.

b) ESCALABILIDAD DE PROCESOS:
   Limitada por:
   - Número de procesos del sistema (ulimit -u)
   - Memoria disponible
   - Overhead de context switching

   En práctica, hasta ~100 robots es manejable.

c) ESCALABILIDAD DE IPC:
   Memoria compartida es eficiente incluso con muchos procesos.

d) MEJORAS PARA ESCALABILIDAD EXTREMA:
   - Usar threads en lugar de procesos
   - Implementar pool de workers
   - Usar memoria compartida con múltiples segmentos

Q6: ¿Cómo se validó la correctitud del sistema?

RESPUESTA:
Mediante varias técnicas:

a) PRUEBAS UNITARIAS:
   - make test ejecuta configuración conocida
   - Verifica que todos los mangos sean etiquetados

b) PRUEBAS DE VALIDACIÓN:
   - Parámetros inválidos son rechazados
   - Límites son respetados

c) ANÁLISIS ESTADÍSTICO:
   - Múltiples simulaciones con diferentes configuraciones
   - Tasa de éxito medida y verificada

d) INSPECCIÓN MANUAL:
   - Output muestra qué robot etiquetó cada mango
   - Verificación visual de que no hay duplicados

e) VERIFICACIÓN DE IPC:
   - make clean-ipc verifica que recursos se liberan
   - No se observaron leaks de memoria

Q7: ¿Qué mejoras se podrían implementar?

RESPUESTAS:

a) OPTIMIZACIÓN DE ASIGNACIÓN:
   - Algoritmo de asignación óptima (Hungarian algorithm)
   - Predecir qué robot debe tomar qué mango

b) VISUALIZACIÓN:
   - Interfaz gráfica en tiempo real
   - Animación de la banda y robots
   - Gráficos de utilización

c) FEATURES AVANZADAS:
   - Múltiples cajas simultáneas
   - Priorización de mangos (más grandes primero)
   - Balanceo dinámico de carga

d) ROBUSTEZ:
   - Logs persistentes
   - Reconexión automática de robots
   - Modo de recuperación de fallas

e) ANÁLISIS:
   - Machine learning para predecir configuración óptima
   - Simulación de Monte Carlo más sofisticada
   - Optimización multi-objetivo (costo vs throughput)

Q8: ¿Cómo se calculó el límite de 7cm para la distancia de mangos?

RESPUESTA:
Mediante análisis matemático y experimental:

a) RESTRICCIÓN TEMPORAL:
   tiempo_disponible = Z / X = 50 / 10 = 5 segundos

b) TIEMPO DE ETIQUETADO:
   tiempo = (2 * distancia) / (Z/10) = 20 * distancia / Z

   Para Z=50:

tiempo = 20 * distancia / 50 = 0.4 * distancia

c) DISTANCIA MÁXIMA TEÓRICA:
tiempo <= tiempo_disponible
0.4 * distancia <= 5
distancia <= 12.5 cm

d) MARGEN DE SEGURIDAD:
En práctica, el robot necesita tiempo para:
- Detectar el mango
- Iniciar movimiento
- Decelerar al llegar

Por eso usamos 7cm (56% del máximo teórico), dejando ~44% de margen.

e) VERIFICACIÓN EXPERIMENTAL:
Con 7cm, observamos:
- 100% éxito con número adecuado de robots
- Tiempo promedio ~2-3s (< 5s disponibles)
- Margin suficiente para variabilidad

==
FIN DEL DOCUMENTO
==