Reto #1: Implementación de una Shell Básica en Go

1. **Contexto del problema:**

   Eres un entusiasta de los sistemas operativos y quieres entender a fondo cómo funcionan las shells como Bash o Zsh. Tu objetivo es construir una shell simplificada en Go. Esta shell debería ser capaz de leer comandos del usuario, ejecutarlos como procesos separados y mostrar su salida. Además, debe soportar algunas características básicas como comandos internos y la ejecución de comandos en segundo plano.

2. **Requerimientos técnicos:**

   a) **Bucle de Lectura-Evaluación-Impresión (REPL):**

      - La shell debe entrar en un bucle continuo:

        1. **Lectura (Read):** Muestra un prompt (ej. goshell>) y lee una línea de entrada del usuario.
        2. **Evaluación (Eval):** Parsea la línea de entrada para identificar el comando y sus argumentos.
        3. **Impresión (Print):** Ejecuta el comando y muestra su salida (stdout y stderr) al usuario.

   b) **Ejecución de Comandos Externos:**

      - Debe ser capaz de ejecutar programas externos disponibles en el PATH del sistema (ej. ls, cat, echo, grep).
      - Utiliza el paquete estándar os/exec para ejecutar estos comandos.
      - Redirige correctamente la **entrada estándar (stdin)**, la **salida estándar (stdout)** y la **salida de error estándar (stderr)** del proceso hijo a la shell.

   c) **Comandos Internos (Built-ins):**

      - Implementa al menos dos comandos internos que no ejecuten programas externos, sino que manejen la lógica dentro de la propia shell:
        - cd <directorio>: Cambia el directorio de trabajo actual de la shell. Utiliza os.Chdir.
        - exit: Termina la shell.

   d) **Ejecución en Segundo Plano (Background Processes):**

      - Permite al usuario ejecutar un comando en segundo plano usando el sufijo & (ej. sleep 5 &).

- Cuando un comando se ejecuta en segundo plano, la shell debe mostrar el PID del proceso hijo y volver al prompt inmediatamente, sin esperar a que el comando termine.
- Los procesos en segundo plano deben ejecutar su salida en la consola de la shell, sin interferir con el prompt.
- Considera cómo manejar la concurrencia para que la shell no se bloquee mientras espera la terminación del proceso en segundo plano.

e) **Manejo de Errores:**

- Maneja comandos no encontrados (ej. command not found).
- Maneja errores de ejecución de procesos.
- Maneja errores de parsing de la entrada.

f) **Paso de Argumentos:**

- Los comandos deben poder recibir múltiples argumentos (ej. ls -l /tmp).

g) **Gestión de Señales (Opcional, Avanzado):**

- Considera cómo la shell manejaría señales como SIGINT (Ctrl+C) para terminar el proceso en primer plano, pero no la propia shell.

3. Restricciones:

- **No se permiten librerías de parsing de comandos de terceros.** El parsing de la línea de entrada (separar comando de argumentos, identificar &) debe ser implementado manualmente.
- La funcionalidad es básica. No se requieren pipes (|), redirecciones (>, <), variables de entorno personalizadas (más allá del PATH del sistema), o autocompletado.
- El almacenamiento de procesos en segundo plano para su posterior gestión (ej. comando jobs) no es un requisito, solo su ejecución y no bloqueo de la shell.

4. Criterios de evaluación:

- **Dominio de os/exec:** Correcta utilización del paquete para la ejecución de procesos y la gestión de su E/S.
- **Manejo de E/S:** Redirección apropiada de stdin, stdout y stderr.
- **Lógica de Parsing:** Implementación robusta del parsing básico de la línea de comandos, incluyendo el sufijo &.
- **Implementación de Built-ins:** Correcta implementación de comandos internos que modifican el estado de la shell.

- **Concurrencia para Background:** Uso efectivo de goroutines para permitir la ejecución en segundo plano sin bloquear la shell principal.
- **Manejo de Errores:** Robustez en la gestión de errores de comandos y procesos.

5. **Preguntas de reflexión previas a codificar:**
    1. **Parsing de la Línea:** ¿Cómo separarías la cadena de entrada en el comando y sus argumentos? ¿Cómo detectarías la presencia del & al final de la línea?
    2. **Redirección de E/S:** Al ejecutar un comando externo, ¿cómo conectarías la salida de ese comando a la salida estándar de tu shell? ¿Qué métodos de os/exec.Cmd usarías?
    3. **Ejecución en Segundo Plano:** Si un comando termina con &, ¿cómo lanzarías el proceso para que no bloquee el hilo principal de la shell? ¿Qué goroutine se encargaría de esperar su finalización y cómo informaría de ello (si es que lo hace) sin interferir con el prompt?
    4. **Flujo de Control:** ¿Cómo diseñarías la estructura principal del bucle REPL para que sea clara y maneje tanto los built-ins como los comandos externos y los procesos en segundo plano?
    5. **Robustez:** ¿Qué sucede si el usuario ingresa una línea vacía o solo espacios? ¿Cómo manejarías esos casos?
6. **Entrega esperada:**
    - Un archivo main.go que contenga el bucle REPL principal.
    - Funciones o módulos adicionales (ej. parser.go para la lógica de parsing, executor.go para la ejecución de comandos) para organizar el código de manera modular.
    - Un archivo README.md explicando:
        - Tu enfoque para el parsing de la línea de comandos.
        - Cómo manejaste la ejecución de comandos externos y la redirección de E/S.
        - La implementación de los comandos internos.
        - La estrategia para la ejecución de comandos en segundo plano y cómo se maneja la concurrencia.
    - Un archivo shell_test.go con pruebas unitarias para la lógica de parsing de comandos y, si es posible, pruebas de integración que ejecuten comandos reales (como ls o sleep) y verifiquen su salida y comportamiento (incluyendo los comandos en segundo plano).