

Generar un programa que sea capaz de restaurar el estado de ejecución.



- **Nombre:** Jesus Eduardo Jimenez Covarrubias
- **Materia:** Computación Tolerante a Fallas
- **Profesor:** Michel Emanuel Lopez Franco
- **NRC:** 179961
- **Sección:** D06
- **Ciclo:** 2025-A

Instrucciones:

(Application checkpointing)

Checkpointing is a technique that provides fault tolerance for computing systems.

<https://docs.python.org/3/library/pickle.html>

Desarrollo:

Código (Python):

```
import pickle
import os
import time

class ApplicationState:
    def __init__(self):
        self.counter = 0
        self.data = []

    def __str__(self):
        return f"Estado actual: Contador = {self.counter}, Datos = {self.data}"

def save_checkpoint(state,
filename="checkpoint.pkl"):
    """Guarda el estado actual de la aplicación"""
    with open(filename, "wb") as f:
        pickle.dump(state, f)
    print("\nCheckpoint guardado exitosamente")

def load_checkpoint(filename="checkpoint.pkl"):
    """Carga el último estado guardado"""
    if os.path.exists(filename):
        with open(filename, "rb") as f:
            state = pickle.load(f)
        print("\nCheckpoint cargado exitosamente")
    return state
```

```

    return None

def main():
    # Intentar cargar checkpoint existente
    state = load_checkpoint()

    # Si no existe checkpoint, crear nuevo estado
    if not state:
        state = ApplicationState()
        print("Iniciando nueva ejecución")
    else:
        print("Reanudando ejecución desde
checkpoint")

    print(state)

    # Simular procesamiento con checkpointing
    automático
    try:
        while state.counter < 100:
            # Realizar algún cálculo
            state.data.append(state.counter ** 2)

            # Mostrar progreso
            print(f"\nProcesando iteración
{state.counter}...")
            time.sleep(0.5) # Simular trabajo

            # Guardar checkpoint cada 5 iteraciones
            if state.counter % 5 == 0 and
state.counter > 0:
                save_checkpoint(state)

            state.counter += 1

        # Eliminar checkpoint al completar
        if os.path.exists("checkpoint.pkl"):
            os.remove("checkpoint.pkl")
        print("\nProceso completado exitosamente")

```

```

except KeyboardInterrupt:
    print("\nInterrupción recibida. Guardando
checkpoint final...")
    save_checkpoint(state)

if __name__ == "__main__":
    main()

```

Explicación del Código (Python):

Vamos a desglosar el código paso a paso y entender cada componente:

1. Importaciones necesarias

```

import pickle # Para serialización de objetos

import os    # Manejo de archivos

import time  # Simular retrasos en el procesamiento

```

2. Clase ApplicationState

```
class ApplicationState:
```

```

    def __init__(self):

        self.counter = 0 # Contador de progreso

        self.data = []   # Datos generados

    def __str__(self):

        return f"Estado actual: Contador = {self.counter}, Datos = {self.data}"

```

Propósito: Almacena todo el estado de la aplicación que queremos preservar

-Atributos:

- `counter` : Lleva registro del progreso
- `data` : Almacena resultados intermedios

- `__str__`: Método para mostrar el estado de forma legible

3. Funciones de Checkpointing**

```
def save_checkpoint(state, filename="checkpoint.pkl"):
```

```
    with open(filename, "wb") as f:
```

```
        pickle.dump(state, f)
```

- Serializa el estado usando ``pickle.dump()``

- `wb`: Abre el archivo en modo escritura binaria

```
def load_checkpoint(filename="checkpoint.pkl"):
```

```
    if os.path.exists(filename):
```

```
        with open(filename, "rb") as f:
```

```
            return pickle.load(f)
```

```
    return None
```

- Deserializa el estado con ``pickle.load()``

- `rb`: Abre el archivo en modo lectura binaria

- Verifica si existe el archivo antes de cargar

4. Función Principal (main)

Flujo lógico:

```
state = load_checkpoint() # Intenta cargar estado previo
```

```
if not state:
```

```
    state = ApplicationState() # Crea nuevo estado si no hay checkpoint
```

- Primero intenta recuperar un estado guardado

- Si no existe, inicia desde cero

Bucle principal:

```
while state.counter < 100:
```

```
    # Actualiza el estado
```

```
    state.data.append(state.counter ** 2)
```

```
    state.counter += 1
```

```
    # Checkpoint cada 5 iteraciones
```

```
    if state.counter % 5 == 0:
```

```
        save_checkpoint(state)
```

- Simula un proceso de cálculo largo (100 iteraciones)

- Guarda checkpoint cada 5 iteraciones

- Almacena cuadrados del contador en `data`

Manejo de interrupciones:**

```
try:
```

```
    # Bucle principal
```

```
except KeyboardInterrupt:
```

```
    save_checkpoint(state) # Guarda estado final al interrumpir
```

- Captura `Ctrl+C` para guardar último estado antes de salir

Limpieza final:

```
if os.path.exists("checkpoint.pkl"):
```

```
    os.remove("checkpoint.pkl")
```

- Elimina el archivo de checkpoint si el proceso completa todas las iteraciones

5. Ejecución del Programa

Comportamientos clave:

1. Primera ejecución:

- Crea nuevo estado
- Guarda checkpoints cada 5 iteraciones

Iniciando nueva ejecución

Estado actual: Contador = 0, Datos = []

Procesando iteración 0...

Procesando iteración 1...

...

Checkpoint guardado en iteración 5

2. Si se interrumpe (Ctrl+C):

- Guarda el estado actual inmediatamente

Procesando iteración 23...

^C

Interrupción recibida. Guardando checkpoint final...

3. Re-ejecución:

- Recupera el último estado guardado

Reanudando ejecución desde checkpoint

Estado actual: Contador = 23, Datos = [0, 1, 4, ..., 484]

6. Consideraciones Importantes

1. Serialización:

- Solo objetos serializables pueden guardarse
- No funciona con:

- Conexiones de red
- Archivos abiertos
- Objetos complejos personalizados (sin implementar `__reduce__`)

2. Seguridad:

- ¡Nunca cargar checkpoints de fuentes no confiables!
- Pickle puede ejecutar código arbitrario

3. Persistencia:

- Los checkpoints son permanentes en disco
- Deben limpiarse manualmente cuando ya no son necesarios

4. Optimización:

- Frecuencia de guardado afecta el rendimiento
- Balance entre seguridad de datos y velocidad

Este patrón es útil para:

- Procesos largos de cálculo
- Entornos con riesgo de fallos
- Aplicaciones distribuidas
- Entornos donde se necesita pausar/reanudar procesos

Pantallazos del Código (Python):

Checkpoint cargado exitosamente
Reanudando ejecución desde checkpoint
Estado actual: Contador = 57, Datos = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444]

Procesando iteración 57...

Procesando iteración 58...

Procesando iteración 59...

Procesando iteración 60...

Checkpoint guardado exitosamente

Procesando iteración 61...

Procesando iteración 62...

Procesando iteración 63...

Procesando iteración 64...

Procesando iteración 73...

Procesando iteración 74...

Procesando iteración 75...

Checkpoint guardado exitosamente

Procesando iteración 76...

Procesando iteración 77...

Procesando iteración 78...

Procesando iteración 79...

Procesando iteración 80...

Checkpoint guardado exitosamente

Procesando iteración 81...

Procesando iteración 82...

Procesando iteración 83...

Procesando iteración 84...

Procesando iteración 85...

Checkpoint guardado exitosamente

Interrupción Ctrl +C:

Iniciando nueva ejecución

Estado actual: Contador = 0, Datos = []

Procesando iteración 0...

Procesando iteración 1...

Procesando iteración 2...

Procesando iteración 3...

Procesando iteración 4...

Procesando iteración 5...

Checkpoint guardado exitosamente





Procesando iteración 6...

Procesando iteración 7...




Procesando iteración 8...

Procesando iteración 9...

Mientras se enjuta el programa se Genera un Archivo.pkl:

| | | | | |
|---|------------------------------|------------------------|---------------------|------|
|  | .idea | 15/02/2025 05:17 p. m. | Carpeta de archivos | |
|  | checkpoints | 13/02/2025 09:04 p. m. | Carpeta de archivos | |
|  | Application_checkpointing.py | 15/02/2025 04:56 p. m. | JetBrains PyCharm | 2 KB |
|  | checkpoint.pkl | 15/02/2025 05:17 p. m. | Archivo PKL | 1 KB |

Cuando se acaba la ejecución se elimina:

| | | | | |
|---|------------------------------|------------------------|---------------------|------|
|  | .idea | 15/02/2025 05:17 p. m. | Carpeta de archivos | |
|  | checkpoints | 13/02/2025 09:04 p. m. | Carpeta de archivos | |
|  | Application_checkpointing.py | 15/02/2025 04:56 p. m. | JetBrains PyCharm | 2 KB |

Conclusión:

Este código es como un botón de "guardar progreso" para tus programas. Imagina que estás haciendo una tarea larga y de repente se apaga la computadora. Con este código, cada ratito guarda dónde vas, para que al prenderla otra vez, no tengas que empezar desde cero.

Así funciona:

Primera vez que lo usas: Empieza de cero, como un juego nuevo.

Si se apaga o lo detienes: Recuerda dónde te quedaste (guarda en un archivo).

Cuando lo abres de nuevo: Te pregunta si quieres seguir desde donde estabas o empezar de nuevo.

Ventajas:

Ahorras tiempo: No repites lo que ya hiciste.

Seguro: Si falla algo, no pierdes todo.

Pausa cuando quieras: Usa Ctrl+C para parar y guardar.

Ejemplo:

Si estás calculando 100 números y se apaga a la mitad, al volver solo terminas lo faltante, ¡no todo de nuevo!

Para usarlo bien:

Déjalo que guarde solo cada cierto tiempo (como cada 5 pasos).

Si ya terminaste, borra el archivo "checkpoint.pkl" para empezar fresco.