MultiProcessWorkflow Documentation

The `MultiProcessWorkflow` class extends the `BaseWorkflow` to support parallel processing using multiple workers. This class is designed to efficiently execute tasks concurrently, leveraging the power of multi-processing to enhance performance and scalability.

Key Concepts

- **Parallel Processing**: Utilizing multiple workers to execute tasks concurrently.
- **Workflow Management**: Handling the execution of tasks in a structured workflow.
- **Agents**: Entities responsible for executing tasks.

Attributes

Arguments

Argument	Type	Default	Description
`max_workers`	`int`	`5`	The maximum number of workers to use for parallel processing.
`autosave`	`bool`	`True`	Flag indicating whether to automatically save the workflow.
`agents`	`Sequence[Agent]`	`None`	A list of agents participating in the workflow.
`*args`			Additional positional arguments.
`**kwargs`			Additional keyword arguments.

Attributes

Attribute	Type	Description
`max_workers`	`int`	The maximum number of workers to use for parallel processing.
`autosave`	`bool`	Flag indicating whether to automatically save the workflow.
`agents`	`Sequence[Agent]`	A list of agents participating in the workflow.

Methods

__init__

Initializes the `MultiProcessWorkflow` with the given parameters.

Examples:

```python
from swarms.structs.agent import Agent
from swarms.structs.task import Task
from swarms.structs.multi_process_workflow import MultiProcessWorkflow

agents = [Agent(name="Agent 1"), Agent(name="Agent 2")]
tasks = [Task(name="Task 1", execute=lambda: "result1"), Task(name="Task 2", execute=lambda: "result2")]

workflow = MultiProcessWorkflow(max_workers=3, agents=agents, tasks=tasks)
```

execute_task

Executes a task and handles exceptions.

Arguments:

Parameter	Type	Description
`task`	`str`	The task to execute.
`*args`		Additional positional arguments for the task execution.
`**kwargs`		Additional keyword arguments for the task execution.

Returns:

Return Type	Description
`Any`	The result of the task execution.

Examples:

```python
result = workflow.execute_task(task="Sample Task")
print(result)
```

run

Runs the workflow.

Arguments:

Parameter	Type	Description
`task`	`str`	The task to run.
`*args`		Additional positional arguments for the task execution.
`**kwargs`		Additional keyword arguments for the task execution.

Returns:

Return Type	Description
`List[Any]`	The results of all executed tasks.

Examples:

```python
results = workflow.run(task="Sample Task")
print(results)
```

Additional Examples

Example 1: Simple Task Execution

```python
from swarms import Agent, Task, MultiProcessWorkflow, OpenAIChat
from datetime import datetime
from time import sleep

import os
from dotenv import load_dotenv

# Load the environment variables
load_dotenv()


# Define a function to be used as the action
def my_action():
    print("Action executed")


# Define a function to be used as the condition
def my_condition():
    print("Condition checked")
    return True
```

```
# Create an agent
agent = Agent(
    llm=OpenAIChat(openai_api_key=os.environ["OPENAI_API_KEY"]),
    max_loops=1,
    dashboard=False,
)

# Create a task
task = Task(
    description=(
        "Generate a report on the top 3 biggest expenses for small"
        " businesses and how businesses can save 20%"
    ),
    agent=agent,
)

# Create a workflow with the task
workflow = MultiProcessWorkflow(tasks=[task])

# Run the workflow
results = workflow.run(task)
print(results)
```

Example 2: Workflow with Multiple Agents

```python
from swarms import Agent, Task, MultiProcessWorkflow

# Define tasks
def task1():
    return "Task 1 result"

def task2():
    return "Task 2 result"

# Create agents
agent1 = Agent(name="Agent 1", llm=OpenAIChat())
agent2 = Agent(name="Agent 2", llm=OpenAIChat())

# Create tasks
task_1 = Task(name="Task 1", execute=task1)
task_2 = Task(name="Task 2", execute=task2)

# Create a workflow
workflow = MultiProcessWorkflow(agents=[agent1, agent2], tasks=[task_1, task_2])

# Run the workflow
results = workflow.run(task="Example Task")
print(results)
```

Example 3: Customizing Max Workers

```python
from swarms import Agent, Task, MultiProcessWorkflow, OpenAIChat

# Define a task
def example_task():
    return "Task result"

# Create an agent
agent = Agent(name="Agent 1", llm=OpenAIChat())

# Create a task
task = Task(name="Example Task", execute=example_task)

# Create a workflow with custom max workers
workflow = MultiProcessWorkflow(max_workers=10, agents=[agent], tasks=[task])

# Run the workflow
results = workflow.run(task="Example Task")
print(results)
```

Summary

The `MultiProcessWorkflow` class provides a powerful framework for managing and executing tasks

using multiple workers. With support for parallel processing, customizable workflows, and detailed logging, it is an ideal tool for complex task execution scenarios. This class enhances performance and scalability, making it suitable for a wide range of applications that require efficient task management.