```
import inspect
import os
import re
import threading

from dotenv import load_dotenv
from swarms_memory import DictInternalMemory, DictSharedMemory

from scripts.auto_tests_docs.docs import TEST_WRITER_SOP_PROMPT
from swarm_models import OpenAIChat

load_dotenv()

api_key = os.getenv("OPENAI_API_KEY")

model = OpenAIChat(
    openai_api_key=api_key,
    max_tokens=4000,
)

# agent = Agent(
#     llm=model,
#     agent_name="Unit Testing Agent",
#     agent_description=(
#         "This agent is responsible for generating unit tests for"
#         " the swarms package."
#     ),
#     autosave=True,
#     system_prompt=None,
#     max_loops=1,
# )


def extract_code_from_markdown(markdown_content: str):
    """
    Extracts code blocks from a Markdown string and returns them as a single string.

    Args:
    - markdown_content (str): The Markdown content as a string.

    Returns:
    - str: A single string containing all the code blocks separated by newlines.
    """
    # Regular expression for fenced code blocks
    pattern = r"```(?:\w+\n)?(.*?)```"
    matches = re.findall(pattern, markdown_content, re.DOTALL)

    # Concatenate all code blocks separated by newlines
    return "\n".join(code.strip() for code in matches)


def create_test(cls):
```

```
    """
    Process the documentation for a given class using OpenAI model and save it in a Python file.
    """
    doc = inspect.getdoc(cls)
    source = inspect.getsource(cls)
    input_content = (
        "Class Name:"
        f" {cls.__name__}\n\nDocumentation:\n{doc}\n\nSource"
        f" Code:\n{source}"
    )

    # Process with OpenAI model (assuming the model's __call__ method takes this input and returns processed content)
    processed_content = model(
        TEST_WRITER_SOP_PROMPT(
            input_content, "swarms", "swarms.memory"
        )
    )
    processed_content = extract_code_from_markdown(processed_content)

    doc_content = f"# {cls.__name__}\n\n{processed_content}\n"

    # Create the directory if it doesn't exist
    dir_path = "tests/memory"
    os.makedirs(dir_path, exist_ok=True)
    # Write the processed documentation to a Python file
    file_path = os.path.join(dir_path, f"{cls.__name__.lower()}.py")
    with open(file_path, "w") as file:
        file.write(doc_content)


def main():
    classes = [
        DictInternalMemory,
        DictSharedMemory,
    ]
    threads = []
    for cls in classes:
        thread = threading.Thread(target=create_test, args=(cls,))
        threads.append(thread)
        thread.start()

    # Wait for all threads to complete
    for thread in threads:
        thread.join()

    print("Tests generated in 'tests/memory' directory.")


if __name__ == "__main__":
    main()
```