

Sub-Task 1: Initialize the logging configuration

Objective: Set up the logging system to log messages with timestamps and process names.

```
import logging
```

```
# Setup logger
```

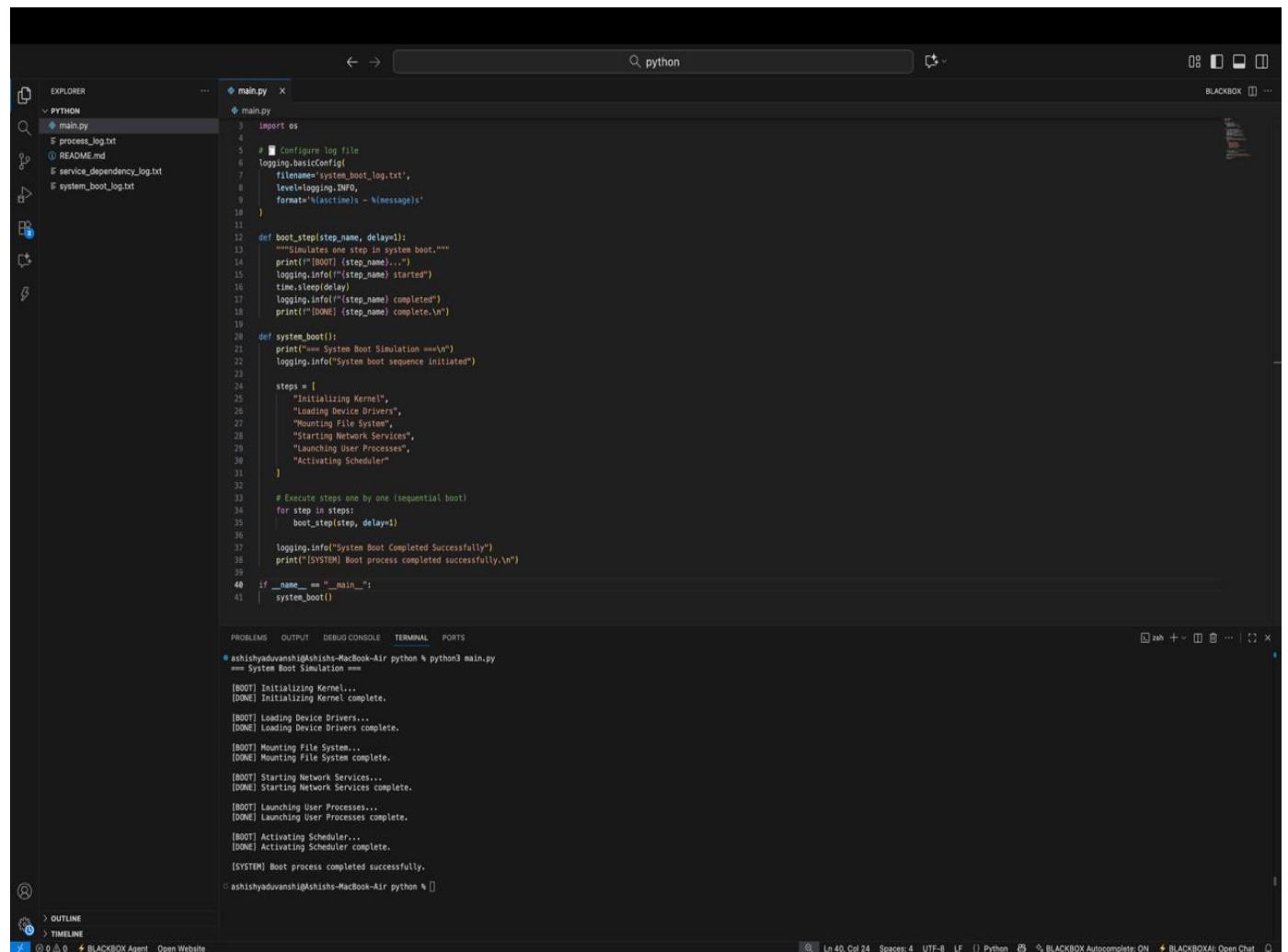
```
logging.basicConfig(
```

```
filename='process_log.txt',
```

```
level=logging.INFO,
```

```
format='%(asctime)s - %(processName)s - %(message)s'
```

```
)
```



The screenshot shows a Python code editor with the following code in the main.py file:

```
import os
import logging
logging.basicConfig(
    filename='process_log.txt',
    level=logging.INFO,
    format='%(asctime)s - %(processName)s - %(message)s'
)

def boot_step(step_name, delay=1):
    """Simulates one step in system boot."""
    print(f'[BOOT] {step_name}...')
    logging.info(f'[step_name] started')
    time.sleep(delay)
    logging.info(f'[step_name] completed')
    print(f'[DONE] {step_name} complete.\n')

def system_boot():
    print("== System Boot Simulation ==\n")
    logging.info("System boot sequence initiated")
    steps = [
        "Initializing Kernel",
        "Loading Device Drivers",
        "Mounting File System",
        "Starting Network Services",
        "Launching User Processes",
        "Activating Scheduler"
    ]
    for step in steps:
        boot_step(step, delay=1)

    logging.info("System Boot Completed Successfully")
    print("[SYSTEM] Boot process completed successfully.\n")

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

The terminal output shows the execution of the main.py script, which logs the boot steps and their completion:

```
ashishyaduvanshi@ashish-MacBook-Air: ~ % python3 main.py
== System Boot Simulation ==
[BOOT] Initializing Kernel...
[DONE] Initializing Kernel complete.
[BOOT] Loading Device Drivers...
[DONE] Loading Device Drivers complete.
[BOOT] Mounting File System...
[DONE] Mounting File System complete.
[BOOT] Starting Network Services...
[DONE] Starting Network Services complete.
[BOOT] Launching User Processes...
[DONE] Launching User Processes complete.
[BOOT] Activating Scheduler...
[DONE] Activating Scheduler complete.
[SYSTEM] Boot process completed successfully.
ashishyaduvanshi@ashish-MacBook-Air: ~ %
```

Sub-Task 2: Define a function that simulates a process task

Objective: Write a function that mimics the work of a system process.

```
import time
```

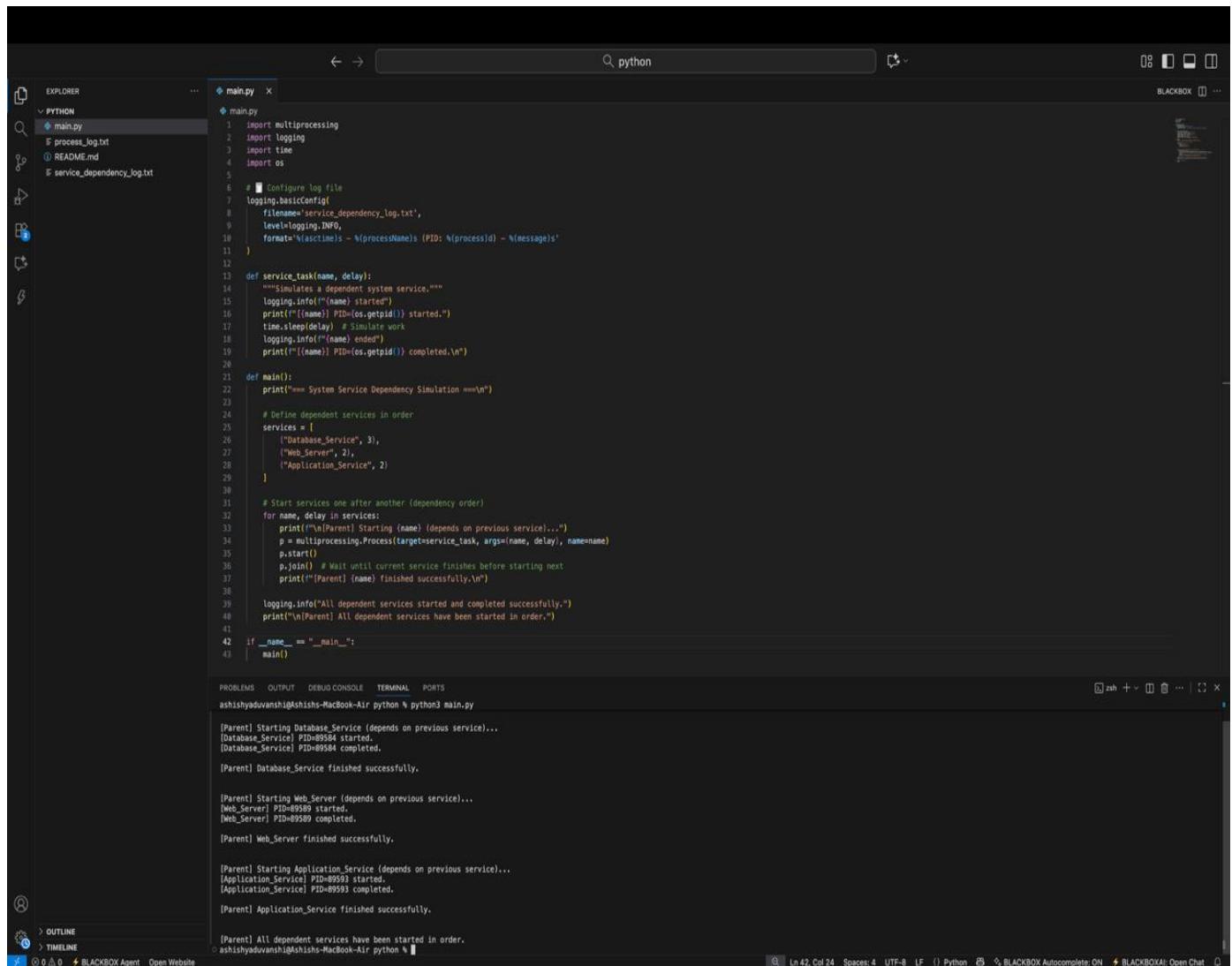
```
# Dummy function to simulate a task
```

```
def system_process(task_name):
```

```
    logging.info(f'{task_name} started')
```

```
    time.sleep(2) # Simulate task delay
```

```
    logging.info(f'{task_name} ended')
```



```
main.py
1 import multiprocessing
2 import logging
3 import time
4 import os
5
6 # Configure log file
7 logging.basicConfig(
8     filename='service_dependency_log.txt',
9     level=logging.INFO,
10    format='[%(asctime)s - %(processName)s (PID: %(processId)d) - %(message)s'
11 )
12
13 def service_task(name, delay):
14     """Simulates a dependent system service."""
15     logging.info(f'{name} started')
16     print(f'[{name}] PID:{os.getpid()} started.')
17     time.sleep(delay) # Simulate work
18     logging.info(f'{name} ended')
19     print(f'[{name}] PID:{os.getpid()} completed.\n')
20
21 def main():
22     print('*** System Service Dependency Simulation ***\n')
23
24     # Define dependent services in order
25     services = [
26         ('Database_Service', 3),
27         ('Web_Server', 2),
28         ('Application_Service', 2)
29     ]
30
31     # Start services one after another (dependency order)
32     for name, delay in services:
33         print(f'[Parent] Starting {name} (depends on previous service)...')
34         p = multiprocessing.Process(target=service_task, args=(name, delay), name=name)
35         p.start()
36         p.join() # Wait until current service finishes before starting next
37         print(f'[Parent] {name} finished successfully.\n')
38
39     logging.info('All dependent services started and completed successfully.')
40     print('All dependent services have been started in order.')
41
42 if __name__ == "__main__":
43     main()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ashishyaduvanshi@ashishs-MacBook-Air python % python3 main.py
[Parent] Starting Database_Service (depends on previous service)...
[Database_Service] PID:89584 started.
[Database_Service] PID:89584 completed.

[Parent] Database_Service finished successfully.

[Parent] Starting Web_Server (depends on previous service)...
[Web_Server] PID:89589 started.
[Web_Server] PID:89589 completed.

[Parent] Web_Server finished successfully.

[Parent] Starting Application_Service (depends on previous service)...
[Application_Service] PID:89593 started.
[Application_Service] PID:89593 completed.

[Parent] Application_Service finished successfully.

[Parent] All dependent services have been started in order.
ashishyaduvanshi@ashishs-MacBook-Air python %
```

Sub-Task 3: Create at least two processes and start them concurrently

Objective: Use the multiprocessing module to initiate parallel tasks.

```
import multiprocessing
```

```
if __name__ == '__main__':
```

```
    print("System Starting...")
```

```
# Create processes
```

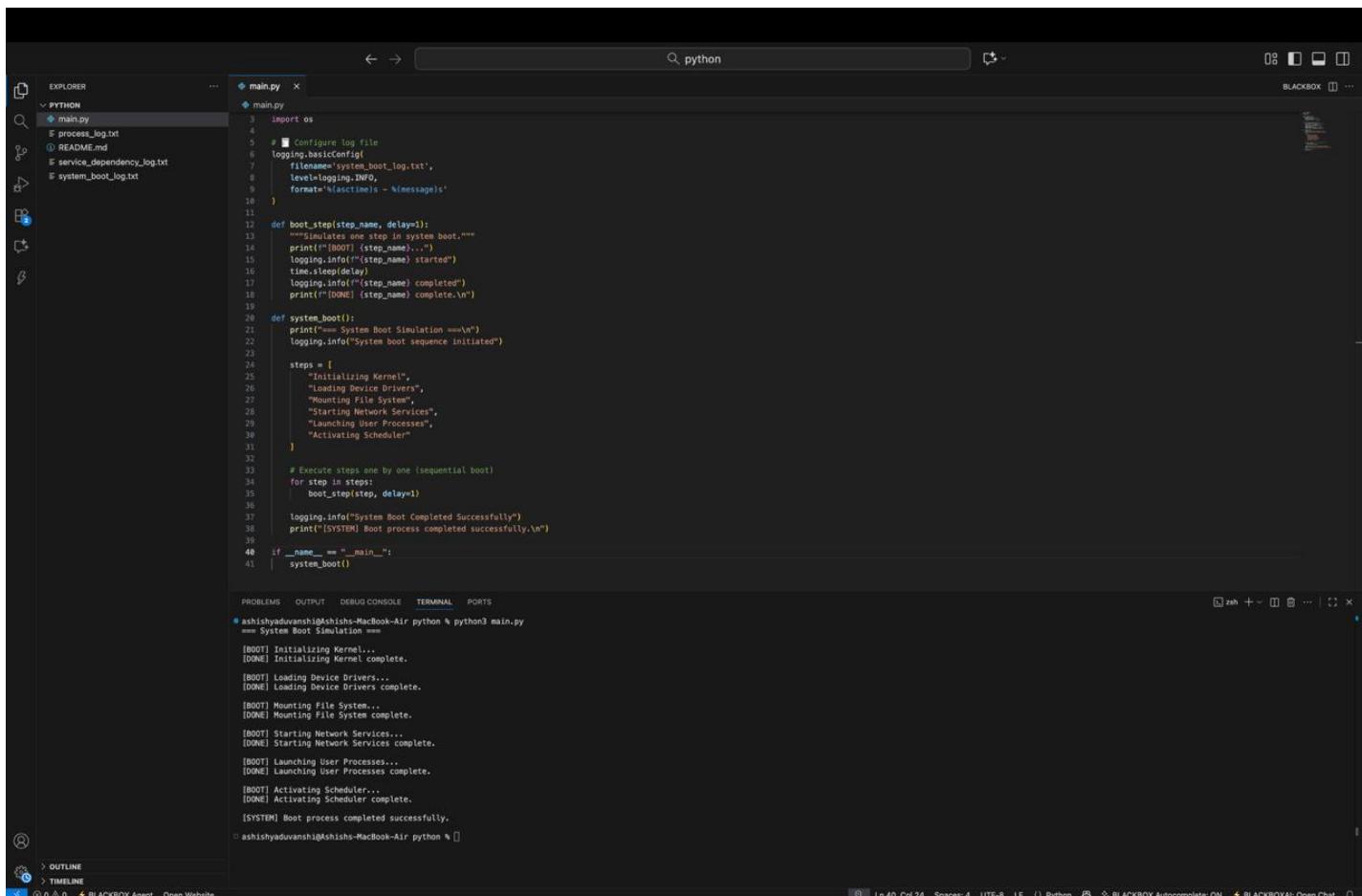
```
p1 = multiprocessing.Process(target=system_process, args=('Process-1',))
```

```
p2 = multiprocessing.Process(target=system_process, args=('Process-2',))
```

```
# Start processes
```

```
p1.start()
```

```
p2.start()
```



The screenshot shows a Python development environment with the following details:

- File Explorer:** Shows a folder named "PYTHON" containing "main.py", "process_log.txt", "README.md", and "service_dependency_log.txt".
- Code Editor:** The "main.py" file is open, displaying Python code for bootstrapping a system. It includes imports for os, logging, and multiprocessing. It defines functions for boot steps, system boot, and a main loop. It uses logging.info for status messages and prints for completed steps.
- Terminal:** The terminal window shows the command "ashishyadu@ashish-MacBook-Air: ~ python3 main.py" and the output of the script. The output shows the system boot process, including kernel initialization, device driver loading, file system mounting, network service starting, user process launching, and scheduler activating. It concludes with the message "[SYSTEM] Boot process completed successfully."
- Status Bar:** Shows the line number (Ln 40), column (Col 24), and spacing (Space: 4, LF). It also indicates "Python" is the active language and "BLACKBOX Autocomplete: ON".

Sub-Task 4: Ensure proper termination and verify logs

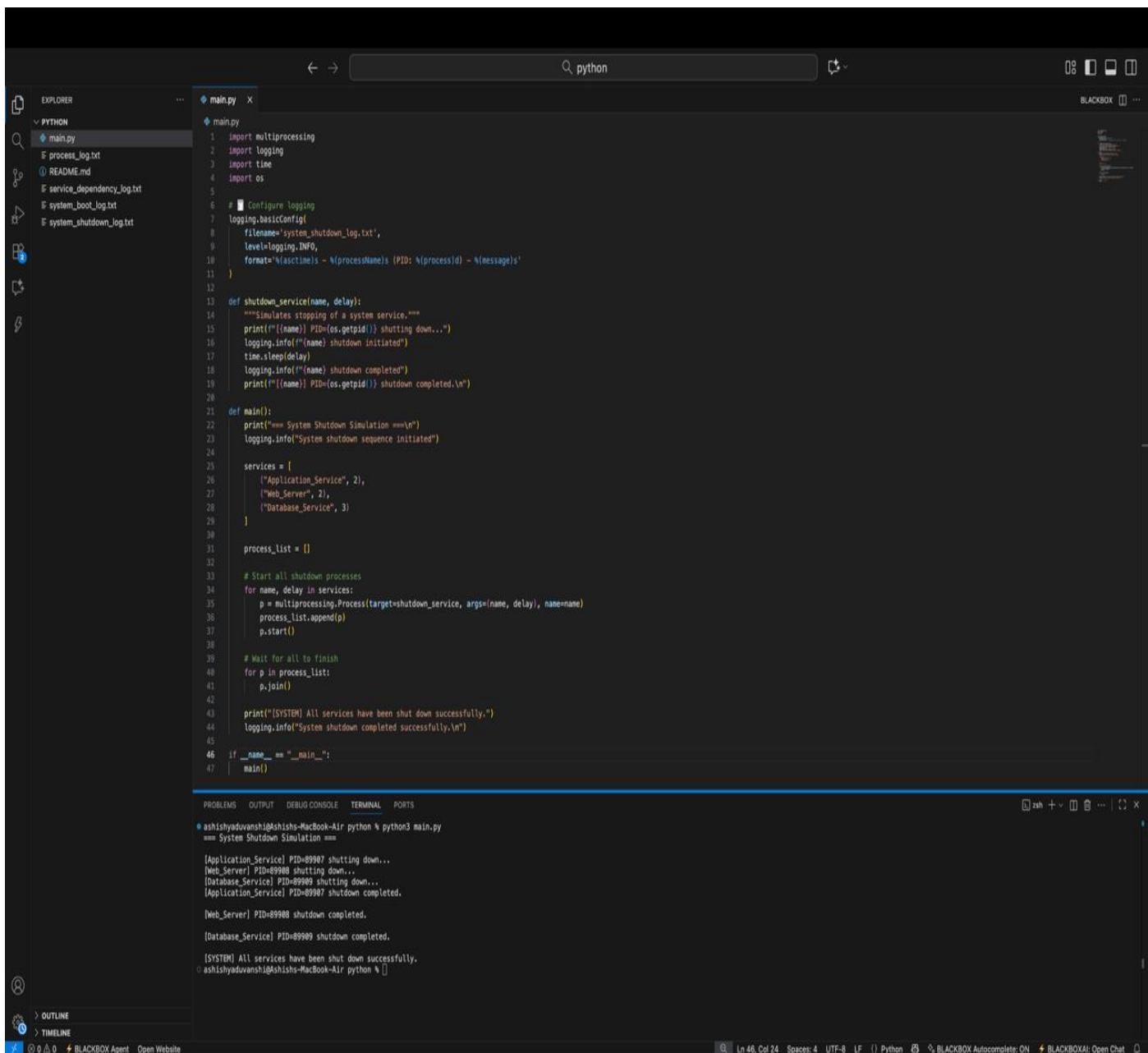
Objective: Wait for processes to complete and confirm the shutdown.

Wait for processes to complete

```
p1.join()
```

```
p2.join()
```

```
print("System Shutdown.")
```



The screenshot shows a code editor interface with a Python file named 'main.py' open. The code implements a system shutdown simulation using the multiprocessing module. It defines a 'shutdown_service' function that prints a shutdown message for a given service and PID. The 'main' function starts processes for three services: Application_Service, Web_Server, and Database_Service, each with a unique PID. It then waits for all processes to join, indicating successful shutdown. The code is annotated with comments explaining its purpose. The code editor includes an Explorer sidebar with files like 'process_log.txt', 'service_dependency_log.txt', and 'system_shutdown_log.txt'. The bottom status bar shows the command 'ashishyaduvanshi@ashishs-MacBook-Air python main.py' and the output of the program's execution.

```
import multiprocessing
import logging
import time
import os

# Configure logging
logging.basicConfig(
    filename='system_shutdown_log.txt',
    level=logging.INFO,
    format='%(asctime)s - %(processName)s (PID: %(processId)d) - %(message)s'
)

def shutdown_service(name, delay):
    """Simulates stopping of a system service."""
    print(f"({name}) PID={os.getpid()} shutting down...")
    logging.info(f"({name}) shutdown initiated")
    time.sleep(delay)
    logging.info(f"({name}) shutdown completed")
    print(f"({name}) PID={os.getpid()} shutdown completed.\n")

def main():
    print("== System Shutdown Simulation ==\n")
    logging.info("System shutdown sequence initiated")

    services = [
        ("Application_Service", 2),
        ("Web_Server", 2),
        ("Database_Service", 3)
    ]

    process_list = []

    # Start all shutdown processes
    for name, delay in services:
        p = multiprocessing.Process(target=shutdown_service, args=(name, delay), name=name)
        process_list.append(p)
        p.start()

    # Wait for all to finish
    for p in process_list:
        p.join()

    print("[SYSTEM] All services have been shut down successfully.")
    logging.info("System shutdown completed successfully.\n")

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

ashishyaduvanshi@ashishs-MacBook-Air python main.py
== System Shutdown Simulation ==
[Application_Service] PID=89987 shutting down...
[Web_Server] PID=89988 shutting down...
[Database_Service] PID=89989 shutting down...
[Application_Service] PID=89997 shutdown completed.
[Web_Server] PID=89998 shutdown completed.
[Database_Service] PID=89999 shutdown completed.
[SYSTEM] All services have been shut down successfully.