Context Switching Lab in Python

Dr. Ahmed Awais

April 4, 2025

1 Objective

The objective of this lab is to implement a context switching mechanism in Python, simulating processes that execute different operations based on instructions from specified from list, tuples, list of tuples and or text files.

Introduction

This Lab Manual illustrates how to implement context switching in Python using threading, and further discusses methods to control the execution sequence of threads. Further, we can employ various synchronization mechanisms, such as Locks and Events, to manage thread execution order effectively.

2 Examples

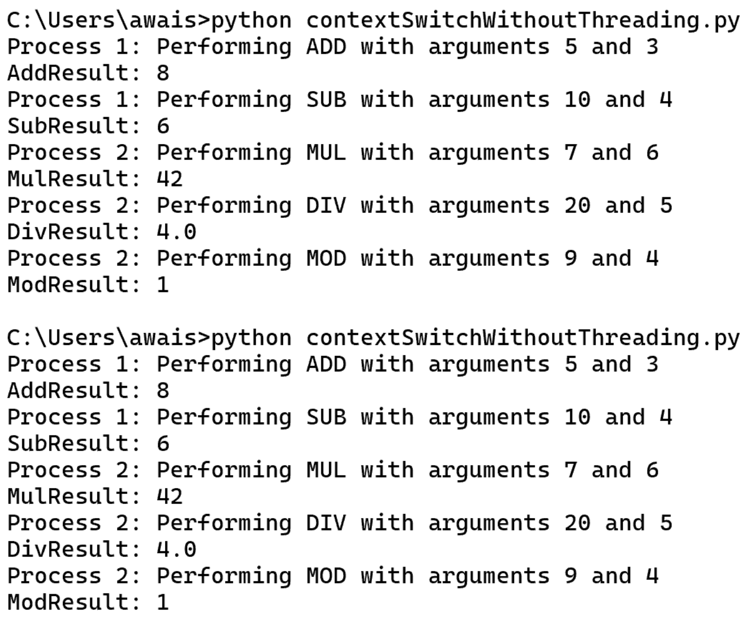
2.1 Basic Context Switching Example without threading

```
class Process:
    def __init__(self, operations, process_id):
        self.operations = operations
        self.process_id = process_id

    def execute(self):
        for operation in self.operations:
            op_name, arg1, arg2 = operation
            print(f"Process {self.process_id}: Performing
            {op_name} with arguments {arg1} and {arg2}")
            self.execute_operation(op_name, arg1, arg2)

    def execute_operation(self, operation, arg1, arg2):
        if operation == "ADD":
            result = arg1 + arg2
            print(f"AddResult: {result}")
        elif operation == "SUB":
            result = arg1 - arg2
            print(f"SubResult: {result}")
        elif operation == "MUL":
            result = arg1 * arg2
            print(f"MulResult: {result}")
        elif operation == "DIV":
            if arg2 != 0:
                result = arg1 / arg2
                print(f"DivResult: {result}")
            else:
                print("Error: Division by zero")
        elif operation == "MOD":
            result = arg1 % arg2
            print(f"ModResult: {result}")

# Define processes and their operations
process_1_operations = [
    ("ADD", 5, 3),  # Process 1 operations
    ("SUB", 10, 4)
]

process_2_operations = [
    ("MUL", 7, 6),  # Process 2 operations
    ("DIV", 20, 5),
    ("MOD", 9, 4)
]

# Create Process instances
process_1 = Process(process_1_operations, "1")
process_2 = Process(process_2_operations, "2")

# Sequentially execute each process
process_1.execute()  # Execute Process 1 first
process_2.execute()  # Then execute Process 2
```

2.2 With Threading

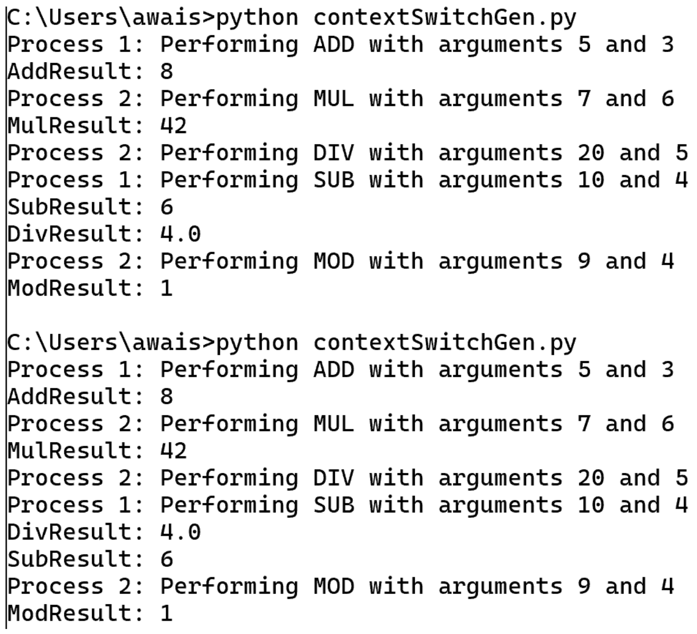
Let”s move to the example of context switching between two threads representing processes with operations defined in code.

```
import threading
import time

process_1_operations = [
    ("ADD", 5, 3),  # Operation format: (operation, operand1, operand2)
    ("SUB", 10, 4)
]

process_2_operations = [
    ("MUL", 7, 6),
    ("DIV", 20, 5),
    ("MOD", 9, 4)
]

def execute_operation(operation, *args):
    if operation == "ADD":
        result = args[0] + args[1]
        print(f"AddResult: {result}")
    elif operation == "SUB":
        result = args[0] - args[1]
        print(f"SubResult: {result}")
    elif operation == "MUL":
        result = args[0] * args[1]
        print(f"MulResult: {result}")
    elif operation == "DIV":
        if args[1] != 0:
            result = args[0] / args[1]
            print(f"DivResult: {result}")
        else:
            print("Error: Division by zero")
    elif operation == "MOD":
        result = args[0] % args[1]
        print(f"ModResult: {result}")

def process_operations(operations, process_id):
    for operation in operations:
        op_name, arg1, arg2 = operation
        print(f"Process {process_id}:
        Performing {op_name} with arguments {arg1} and {arg2}")
        execute_operation(op_name, arg1, arg2)
        time.sleep(1)  # Simulate time taken for the operation

thread1 = threading.Thread(target=process_operations, args=(process_1_operations, "1"))
thread2 = threading.Thread(target=process_operations, args=(process_2_operations, "2"))

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```



```
C:\Users\awais>python contextSwitchWithoutThreading.py
Process 1: Performing ADD with arguments 5 and 3
AddResult: 8
Process 1: Performing SUB with arguments 10 and 4
SubResult: 6
Process 2: Performing MUL with arguments 7 and 6
MulResult: 42
Process 2: Performing DIV with arguments 20 and 5
DivResult: 4.0
Process 2: Performing MOD with arguments 9 and 4
ModResult: 1

C:\Users\awais>python contextSwitchWithoutThreading.py
Process 1: Performing ADD with arguments 5 and 3
AddResult: 8
Process 1: Performing SUB with arguments 10 and 4
SubResult: 6
Process 2: Performing MUL with arguments 7 and 6
MulResult: 42
Process 2: Performing DIV with arguments 20 and 5
DivResult: 4.0
Process 2: Performing MOD with arguments 9 and 4
ModResult: 1
```

Figure 1: Context Switch Without Threading



```
C:\Users\awais>python contextSwitchGen.py
Process 1: Performing ADD with arguments 5 and 3
AddResult: 8
Process 2: Performing MUL with arguments 7 and 6
MulResult: 42
Process 2: Performing DIV with arguments 20 and 5
Process 1: Performing SUB with arguments 10 and 4
SubResult: 6
DivResult: 4.0
Process 2: Performing MOD with arguments 9 and 4
ModResult: 1

C:\Users\awais>python contextSwitchGen.py
Process 1: Performing ADD with arguments 5 and 3
AddResult: 8
Process 2: Performing MUL with arguments 7 and 6
MulResult: 42
Process 2: Performing DIV with arguments 20 and 5
Process 1: Performing SUB with arguments 10 and 4
DivResult: 4.0
SubResult: 6
Process 2: Performing MOD with arguments 9 and 4
ModResult: 1
```

Figure 2: Context Switch With Threading

2.3 With Threading along with Filing

```
Create a file named as tasks.txt with the following data
process 1
ADD 5 3 AddResult
SUB 10 4 SubResult
process 2
MUL 7 6 MulResult
DIV 20 5 DivResult
MOD 9 4 ModResult
```

```
import threading
import time

def execute_operation(operation, *args):
    if operation == "ADD":
        result = int(args[0]) + int(args[1])
        print(f"AddResult: {result}")
    elif operation == "SUB":
        result = int(args[0]) - int(args[1])
        print(f"SubResult: {result}")
    elif operation == "MUL":
        result = int(args[0]) * int(args[1])
        print(f"MulResult: {result}")
    elif operation == "DIV":
        if int(args[1]) != 0:
            result = int(args[0]) / int(args[1])
            print(f"DivResult: {result}")
        else:
            print("Error: Division by zero")
    elif operation == "MOD":
        result = int(args[0]) % int(args[1])
        print(f"ModResult: {result}")

def process_from_file(filename, process_id):
    with open(filename, "r") as file:
        tasks = file.readlines()

    for line in tasks:
        line = line.strip()
        if line.startswith(f"process {process_id}"):
            continue

        if line.startswith("process"):
            break  # Stop processing if we reach the next process

        operation_parts = line.split()
        operation = operation_parts[0]  # ADD, SUB, etc.
        args = operation_parts[1:3]  # operands a, b
        execute_operation(operation, *args)
        time.sleep(1)  # Simulate time taken for the operation

# Define the threads for each process
thread1 = threading.Thread(target=process_from_file, args=("tasks.txt", "1"))
thread2 = threading.Thread(target=process_from_file, args=("tasks.txt", "2"))

# Start the threads
thread1.start()
thread2.start()

# Wait for both threads to complete
thread1.join()
thread2.join()
```