PRACTICAL 7

def optimal\_page\_replacement(pages, capacity):

    page\_faults = 0

    page\_frames = [-1] \* capacity

    next\_use = [-1] \* capacity

    for i, page in enumerate(pages):

        if page not in page\_frames:

            page\_faults += 1

            if -1 in page\_frames:

                # If there is an empty slot, insert the page

                empty\_slot = page\_frames.index(-1)

                page\_frames[empty\_slot] = page

                next\_use[empty\_slot] = find\_next\_use(pages, i)

            else:

                # Find the page in page\_frames with the farthest next use

                idx = next\_use.index(max(next\_use))

                page\_frames[idx] = page

                next\_use[idx] = find\_next\_use(pages, i)

    print("Page Faults:", page\_faults)

def find\_next\_use(pages, current\_index):

    for i in range(current\_index + 1, len(pages)):

        if pages[i] in pages[current\_index + 1:]:

            return i

    return float('inf')  # If the page is not used anymore, set next use to infinity

if \_\_name\_\_ == "\_\_main\_\_":

    # Example pages reference string

    pages\_reference = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1]

    # Set the number of page frames (capacity)

    page\_frame\_capacity = 3

    optimal\_page\_replacement(pages\_reference, page\_frame\_capacity)

PRACTICAL 8

class Process:

    def \_\_init\_\_(self, process\_id, arrival\_time, burst\_time):

        self.process\_id = process\_id

        self.arrival\_time = arrival\_time

        self.burst\_time = burst\_time

def sjf\_scheduler(processes):

    processes.sort(key=lambda x: (x.arrival\_time, x.burst\_time))  # Sort processes based on arrival time and burst time

    completion\_time = [0] \* len(processes)

    turnaround\_time = [0] \* len(processes)

    waiting\_time = [0] \* len(processes)

    completion\_time[0] = processes[0].burst\_time

    turnaround\_time[0] = completion\_time[0] - processes[0].arrival\_time

    waiting\_time[0] = turnaround\_time[0] - processes[0].burst\_time

    for i in range(1, len(processes)):

        completion\_time[i] = completion\_time[i - 1] + processes[i].burst\_time

        turnaround\_time[i] = completion\_time[i] - processes[i].arrival\_time

        waiting\_time[i] = turnaround\_time[i] - processes[i].burst\_time

    print("Process\t Arrival Time\t Burst Time\t Completion Time\t Turnaround Time\t Waiting Time")

    for i in range(len(processes)):

        print(f"{processes[i].process\_id}\t\t{processes[i].arrival\_time}\t\t{processes[i].burst\_time}\t\t"

              f"{completion\_time[i]}\t\t\t{turnaround\_time[i]}\t\t\t{waiting\_time[i]}")

if \_\_name\_\_ == "\_\_main\_\_":

    # Example processes

    processes = [

        Process(1, 0, 6),

        Process(2, 2, 8),

        Process(3, 3, 4),

        Process(4, 5, 3),

    ]

    sjf\_scheduler(processes)