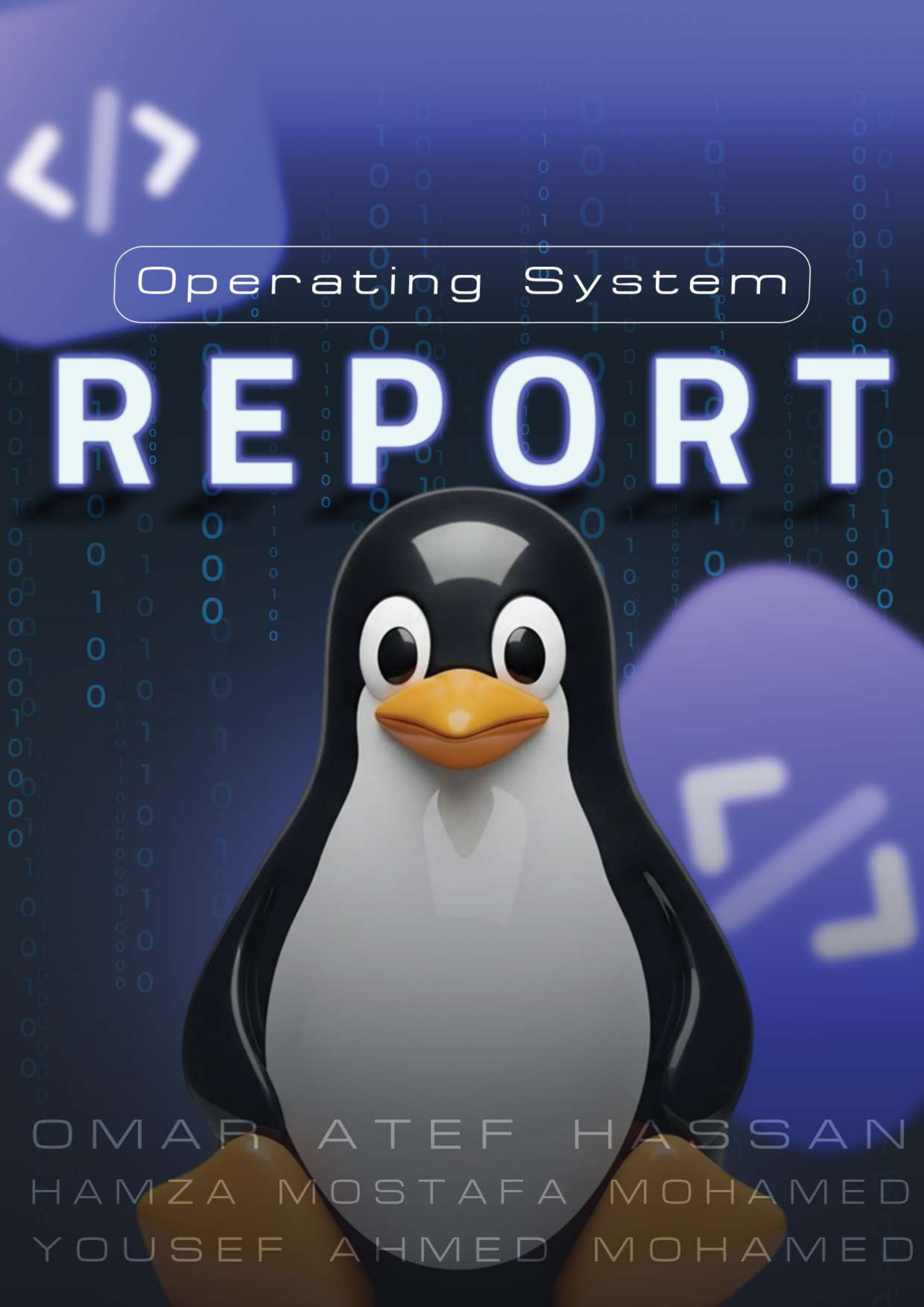
Operating System

REPORT



OMAR ATEF HASSAN
HAMZA MOSTAFA MOHAMED
YOUSEF AHMED MOHAMED

Team Members

عمر عاطف حسن

حمزه مصطفي محمد

يوسف محمد احمد

Multilevel Queue

In [2]:
```
class MultilevelQueueAlgorithm:
    def run(self):
        try:
            n = int(input("Enter Number of Processes: "))
            pid = [] # Process ID
            at = []
            bt = []
            qn = []   # Queue Number
            ct = []
            tat = []
            wt = []

            for i in range(n):
                print(f"\nProcess P{i+1}")
                pid.append(f"P{i+1}")
                at.append(float(input("Enter Arrival Time: ")))
                bt.append(float(input("Enter Burst Time: ")))
                # determent which Queue process belongs to
                qn.append(int(input("Enter Queue Number (1 or 2): ")))

            time = 0

            # Execute Queue 1 (High Priority)
            print("\nExecuting Queue 1 (FCFS)...")
            for i in range(n):
                if qn[i] == 1: # process in Queue1
                    if time < at[i]: # CPU Empty --> waiting for the arrival time
                        time = at[i]
                    #Calc CT :
                    ct.append(time + bt[i])
                    # Update Time :
                    time = ct[-1]
                    tat.append(ct[-1] - at[i])
                    wt.append(tat[-1] - bt[i])

            # Execute Queue 2 (Lower Priority)
```

```
            print("\nExecuting Queue 2 (FCFS)...")
            for i in range(n):
                if qn[i] == 2:# process in Queue2
                    if time < at[i]: # CPU Empty --> waiting for the arrival time
                        time = at[i]
                    # Calc CT :
                    ct.append(time + bt[i])
                    # Update Time :
                    time = ct[-1]
                    tat.append(ct[-1] - at[i])
                    wt.append(tat[-1] - bt[i])

            # Print Table
            print("\nPID\tQ\tAT\tBT\tCT\tTAT\tWT")
            index = 0
            total_tat = 0
            total_wt = 0
            # Note: the execution is not based on PID.
            for i in range(n):
                print(f"{pid[i]}\t{qn[i]}\t{at[i]}\t{bt[i]}\t{ct[index]}\t{tat[index]}\t{wt[index]}")
                total_tat += tat[index]
                total_wt += wt[index]
                index += 1
            # Calc AvgTAT,AvgWT:
            avgTAT = total_tat / n
            avgWT = total_wt / n

            print(f"\nAverage Turnaround Time = {avgTAT}")
            print(f"Average Waiting Time = {avgWT}")

        except ValueError:
            print("Error: Please enter valid numbers")


mlq = MultilevelQueueAlgorithm()
mlq.run()
```

```
Process P1
Process P2
Process P3
```

```
Executing Queue 1 (FCFS)...

Executing Queue 2 (FCFS)...

PID     Q       AT      BT      CT      TAT     WT
P1      2       1.0     2.0     6.0     2.0     0.0
P2      2       5.0     6.0     8.0     7.0     5.0
P3      1       4.0     2.0     14.0    9.0     3.0

Average Turnaround Time = 6.0
Average Waiting Time = 2.6666666666666665
```

SJFPreemptive

In [3]:
```python
class SJFPreemptive:
    def run(self):
        try:
            process = int(input("Enter Number of Processes: "))

            at = []  # Arrival Time
            bt = []  # Burst Time
            rt = []  # Remaining Time الوقت الباقي
            for i in range(process): # input values AT,BT
                arrival_time = float(input(f"Enter Arrival Time for P{i}: "))
                burst_time = float(input(f"Enter Burst Time for P{i}: "))
                at.append(arrival_time)
                bt.append(burst_time)
            rt = bt.copy()  # Remaining time initially = Burst Time

            # Book a place in memory --> EX:ct[shortest] = current_time
            ct = [0] * process  # Completion Time
            tat = [0] * process  # Turnaround Time
            wt = [0] * process  # Waiting Time

            completed = 0
            current_time = 0
            min_rt = float('inf') # lesest current time
            shortest = -1 # num select propcess
            check = False # If procc ready or no
```

```
            while completed != process:

                min_rt = float('inf')
                shortest = -1
                check = False
                for i in range(process):
                    if at[i] <= current_time and rt[i] > 0: #procc arrived but not complete
                        if rt[i] < min_rt:
                            min_rt = rt[i] # select that procc
                            shortest = i
                            check = True
                        elif rt[i] == min_rt:
                            if at[i] < at[shortest]: #select that arrived early
                                shortest = i
                if not check:
                    current_time += 1 # go next time
                    continue


                rt[shortest] -= 1 # Execute only one time unit.
                current_time += 1


                if rt[shortest] == 0: # That procc end
                    completed += 1 # record the end time
                    ct[shortest] = current_time
                    # calc TAT , WT
                    tat[shortest] = ct[shortest] - at[shortest]
                    wt[shortest] = tat[shortest] - bt[shortest]

            print("\nCompletion Time calculated...")
            print("\nPID\tAT\tBT\tCT\tTAT\tWT")
            for i in range(process):
                print(f"{i+1}\t{at[i]}\t{bt[i]}\t{ct[i]}\t{tat[i]}\t{wt[i]}")

            avgTAT = sum(tat) / process
            avgWT = sum(wt) / process

            print(f"\nAverage Turn around Time = {avgTAT}")
            print(f"Average Waiting Time = {avgWT}")

        except ValueError:
```

```
            print("Error: Please enter a valid number")
sjf = SJFPreemptive()
sjf.run()
```

```
Completion Time calculated...

PID     AT      BT      CT      TAT     WT
1       1.0     2.0     3       2.0     0.0
2       3.0     5.0     10      7.0     2.0
3       4.0     2.0     6       2.0     0.0

Average Turn around Time = 3.6666666666666665
Average Waiting Time = 0.6666666666666666
```

Priority Non-Preemptive

In [4]:
```
def priority_non_preemptive():
    try:
        process = int(input("Enter Number of Processes: "))

        at = []
        bt = []
        priority = []
        ct = [0] * process
        tat = [0] * process
        wt = [0] * process
        finished = [False] * process

        # Read Process
        for i in range(process):
            at.append(float(input(f"Enter Arrival Time for P{i}: ")))
            bt.append(float(input(f"Enter Burst Time for P{i}: ")))
            priority.append(float(input(f"Enter Priority for P{i}: ")))

        current_time = 0
        completed = 0
        # Select the highest-priority process that has already arrived
        while completed < process:
            idx = -1
            best_priority = float("inf")
```

```
            # Find best priority
            for i in range(process):
                if not finished[i] and at[i] <= current_time:
                    if priority[i] < best_priority:
                        best_priority = priority[i]
                        idx = i

            # nothing arrived yet → go to the next arrival time
            if idx == -1:
                current_time += 1
                continue

            # Run the selected process
            current_time += bt[idx]
            ct[idx] = current_time
            tat[idx] = ct[idx] - at[idx]
            wt[idx] = tat[idx] - bt[idx]

            finished[idx] = True
            completed += 1

        # Print Results
        print("\nPID\tAT\tBT\tPr\tCT\tTAT\tWT")
        for i in range(process):
            print(f"{i+1}\t{at[i]}\t{bt[i]}\t{priority[i]}\t{ct[i]}\t{tat[i]}\t{wt[i]}")

        avgTAT = sum(tat) / process
        avgWT = sum(wt) / process

        print(f"\nAverage Turnaround Time = {avgTAT}")
        print(f"Average Waiting Time = {avgWT}")

    except ValueError:
        print("Error: Please enter a valid number")
priority_non_preemptive()
```

```
PID     AT      BT      Pr      CT      TAT     WT
1       1.0     2.0     5.0     3.0     2.0     0.0
2       2.0     6.0     10.0    9.0     7.0     1.0
3       5.0     4.0     9.0     13.0    8.0     4.0

Average Turnaround Time = 5.666666666666667
Average Waiting Time = 1.6666666666666667
```

FIFO

In [5]:
```python
def FIFOMemory(self):
        try:
            frames_count = int(input("Enter Number of Frames: "))
            n = int(input("Enter Number of Page References: "))

            pages = []
            for i in range(n):
                pages.append(int(input(f"Enter Page {i}: ")))

            frames = []             # Memory frames
            fifo_index = 0          # Points to oldest page
            page_faults = 0         # number error/faile
            page_hits = 0           # number show pages

            print("\nPage\tFrames\t\tStatus")
            print("--------------------------------")

            for page in pages:
                # Page HIT
                if page in frames: # found in page
                    page_hits += 1
                    print(f"{page}\t{frames}\tHIT")

                # Page FAULT
                else:
                    page_faults += 1

                    # If memory not full → add page
                    if len(frames) < frames_count:
                        frames.append(page)
```

```
                    else:
                        # Replace the OLDEST page (FIFO)
                        frames[fifo_index] = page
                        # % frames_count --> go First Frame
                        fifo_index = (fifo_index + 1) % frames_count

                    print(f"{page}\t{frames}\tFAULT")

            print("\nTotal Page Faults =", page_faults)
            print("Total Page Hits =", page_hits)

        except ValueError:
            print("Error: Please enter valid numbers")
n = int(input("Enter You Select Number: "))
FIFOMemory(n)
```

```
Page    Frames          Status
------------------------------
1       [1]     FAULT
7       [1, 7]  FAULT
5       [1, 7, 5]       FAULT
6       [6, 7, 5]       FAULT

Total Page Faults = 4
Total Page Hits = 0
```

LRU :

In [8]:
```
def LRUPageReplacement(self):
        try:
            frames_count = int(input("Enter number of frames: "))
            pages = list(map(int, input("Enter page reference string (space separated): ").split()))

            frames = []          # Memory frames
            page_faults = 0

            print("\nPage\tFrames\t\tStatus")
            print("-" * 40)

            for page in pages:
```

```
                # Page Hit
                if page in frames: # found --> Delete From old add in last list
                    frames.remove(page)
                    frames.append(page)
                    status = "Hit"
                else:
                    # Page Fault
                    page_faults += 1
                    status = "Fault"

                    # If frames full → remove least recently used
                    if len(frames) == frames_count:
                        frames.pop(0)
                    # add page that --> Latest page
                    frames.append(page)
                # print Stage after each page
                print(f"{page}\t{frames}\t{status}")

            print("\nTotal Page Faults =", page_faults)

        except ValueError:
            print("Error: Please enter valid numbers")
n = int(input("Enter You Select Number: "))
LRUPageReplacement(n)
```

```
Page    Frames          Status
---------------------------------------
2       [2]     Fault

Total Page Faults = 1
```

In [ ]:

In [ ]: