1. Banker's Algorithm

```
# Number of processes and resources

processes = 5

resources = 3

# Available resources

available = [3, 3, 2]

# Maximum resources each process may need

maximum = [

    [7, 5, 3],

    [3, 2, 2],

    [9, 0, 2],

    [2, 2, 2],

    [4, 3, 3]

]

# Resources currently allocated to each process

allocated = [

    [0, 1, 0],

    [2, 0, 0],

    [3, 0, 2],

    [2, 1, 1],

    [0, 0, 2]

]


# Calculate the need matrix

need = [[maximum[i][j] - allocated[i][j] for j in range(resources)] for i in range(processes)]
```

```
def is_safe():
    # Initialize work and finish arrays
    work = available[:]
    finish = [False] * processes
    safe_sequence = []

    while len(safe_sequence) < processes:
        found_process = False
        for i in range(processes):
            if not finish[i] and all(need[i][j] <= work[j] for j in range(resources)):
                # Process i can safely execute
                for j in range(resources):
                    work[j] += allocated[i][j]
                safe_sequence.append(i)
                finish[i] = True
                found_process = True
                break
        if not found_process:
            return False, []
    return True, safe_sequence

# Check for system safety
safe, sequence = is_safe()

if safe:
    print("System is in a safe state.")
```

```
    print("Safe sequence:", sequence)
else:
    print("System is not in a safe state.")
```

2. Page replacement algorithm :

a. FIFO

```
def fifo_page_replacement(pages, capacity):
    memory = []  # Memory to store pages
    page_faults = 0  # Count of page faults

    print("Page\tMemory\t\tPage Fault")
    for page in pages:
        # Check if the page is already in memory
        if page not in memory:
            # If memory is full, remove the oldest page (FIFO)
            if len(memory) >= capacity:
                memory.pop(0)
            # Add the new page to memory
            memory.append(page)
            page_faults += 1
            page_fault_status = "Yes"
        else:
            page_fault_status = "No"

        # Print current state
        print(f"{page}\t{memory}\t{' ' * (8 - len(str(memory)))}{page_fault_status}")
```

```python
    return page_faults

# Example usage

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

capacity = 3

print(f"Total Page Faults: {fifo_page_replacement(pages, capacity)}")
```

b. LRU

```python
def lru_page_replacement(pages, capacity):
    memory = []  # Memory to store pages
    page_faults = 0  # Count of page faults
    page_order = []  # Keeps track of the order of pages for LRU


    print("Page\tMemory\t\tPage Fault")
    for page in pages:
        if page not in memory:
            # Page fault occurs
            if len(memory) >= capacity:
                # Remove the least recently used page
                lru_page = page_order.pop(0)
                memory.remove(lru_page)
            memory.append(page)
            page_order.append(page)
            page_faults += 1
            page_fault_status = "Yes"
        else:
            # Update the order since the page is used
            page_order.remove(page)
```

```
            page_order.append(page)
            page_fault_status = "No"

        # Print current state
        print(f"{page}\t{memory}\t{' ' * (8 - len(str(memory)))}{page_fault_status}")
    return page_faults
# Example usage
pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]
capacity = 3

print(f"Total Page Faults: {lru_page_replacement(pages, capacity)}")
```

c. OPT

```
def opt_page_replacement(pages, capacity):
    memory = []  # Memory to store pages
    page_faults = 0  # Count of page faults

    print("Page\tMemory\t\tPage Fault")
    for i in range(len(pages)):
        page = pages[i]

        if page not in memory:
            # Page fault occurs
            if len(memory) >= capacity:
                # Find the page to replace (the one that will not be used for the longest
time)
                farthest_index = -1
```

```
            farthest_page = None

            for m in memory:
                try:
                    # Get the next occurrence of the page in the future reference string
                    index = pages[i + 1:].index(m) + i + 1
                except ValueError:
                    # If the page is not used in the future, treat it as the farthest
                    index = float('inf')

                # Update if this page's future occurrence is farther
                if index > farthest_index:
                    farthest_index = index
                    farthest_page = m

            # Remove the farthest page (the one that will not be used for the longest
time)
            memory.remove(farthest_page)

        # Add the new page to memory
        memory.append(page)
        page_faults += 1
        page_fault_status = "Yes"
    else:
        page_fault_status = "No"

    # Print current state
```

```
        print(f"{page}\t{memory}\t{' ' * (8 - len(str(memory)))}{page_fault_status}")

    return page_faults


# Example usage
pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]
capacity = 3

print(f"Total Page Faults: {opt_page_replacement(pages, capacity)}")
```

3. Scheduling Algorithm :

a. FCFS

```
def fcfs_scheduling(processes):
    n = len(processes)

    # Sort processes based on arrival time
    processes.sort(key=lambda x: x[1])  # Sort by arrival time

    wait_time = 0  # Total wait time
    turnaround_time = 0  # Total turnaround time
    completion_time = []  # Completion times for each process

    # Calculate the completion times and other metrics
    current_time = 0
    for i in range(n):
```

```
    process, arrival, burst = processes[i]

    # Calculate waiting time for each process
    if current_time < arrival:
        current_time = arrival

    wait = current_time - arrival
    wait_time += wait

    # Update current time after the process completes
    current_time += burst

    # Calculate turnaround time for each process
    turnaround = wait + burst
    turnaround_time += turnaround

    completion_time.append((process, current_time))

# Calculate average waiting time and turnaround time
avg_wait_time = wait_time / n
avg_turnaround_time = turnaround_time / n

# Print results
print(f"Process\tArrival Time\tBurst Time\tWait Time\tTurnaround Time")
for i in range(n):
    process, arrival, burst = processes[i]
```

```
        wait = completion_time[i][1] - arrival - burst
        turnaround = completion_time[i][1] - arrival
        print(f"{process}\t{arrival}\t\t{burst}\t\t{wait}\t\t{turnaround}")

    print(f"\nAverage Waiting Time: {avg_wait_time}")
    print(f"Average Turnaround Time: {avg_turnaround_time}")

# Example Usage
processes = [
    ("P1", 0, 4),  # (process name, arrival time, burst time)
    ("P2", 1, 3),
    ("P3", 2, 2),
    ("P4", 3, 1)
]

fcfs_scheduling(processes)
```

b. SJT

```
def sjt_scheduling(processes):
    n = len(processes)

    # Sort processes based on burst time (and arrival time for tie-breaking)
    processes.sort(key=lambda x: (x[2], x[1]))  # Sort by burst time, then arrival time

    wait_time = 0  # Total waiting time
    turnaround_time = 0  # Total turnaround time
```

```
completion_time = []  # Completion times for each process


# Calculate the completion times and other metrics
current_time = 0
for i in range(n):
    process, arrival, burst = processes[i]


    # If the current time is less than the arrival time, move current time to arrival
  time
    if current_time < arrival:
        current_time = arrival


    # Calculate waiting time
    wait = current_time - arrival
    wait_time += wait


    # Update current time after process completes
    current_time += burst


    # Calculate turnaround time
    turnaround = wait + burst
    turnaround_time += turnaround


    # Store the completion time for each process
    completion_time.append((process, current_time))


# Calculate average waiting time and turnaround time
```

```
    avg_wait_time = wait_time / n
    avg_turnaround_time = turnaround_time / n

    # Print results
    print(f"Process\tArrival Time\tBurst Time\tWait Time\tTurnaround Time")
    for i in range(n):
        process, arrival, burst = processes[i]
        wait = completion_time[i][1] - arrival - burst
        turnaround = completion_time[i][1] - arrival
        print(f"{process}\t{arrival}\t\t{burst}\t\t{wait}\t\t{turnaround}")

    print(f"\nAverage Waiting Time: {avg_wait_time}")
    print(f"Average Turnaround Time: {avg_turnaround_time}")

# Example Usage
processes = [
    ("P1", 0, 6),  # (process name, arrival time, burst time)
    ("P2", 1, 8),
    ("P3", 2, 7),
    ("P4", 3, 3),
]

sjt_scheduling(processes)
```

c. Priority Scheduling

```
def priority_scheduling(processes):
    # Sort processes based on priority, then by arrival time
    processes.sort(key=lambda x: (x[3], x[1]))  # Sort by priority, then arrival time

    wait_time = 0  # Total waiting time
    turnaround_time = 0  # Total turnaround time
    completion_time = []  # Completion times for each process

    # Calculate the completion times and other metrics
    current_time = 0
    for i in range(len(processes)):
        process, arrival, burst, priority = processes[i]

        # If the current time is less than the arrival time, move current time to arrival
time
        if current_time < arrival:
            current_time = arrival

        # Calculate waiting time
        wait = current_time - arrival
        wait_time += wait

        # Update current time after process completes
        current_time += burst

        # Calculate turnaround time
        turnaround = wait + burst
```

```
        turnaround_time += turnaround

        # Store the completion time for each process
        completion_time.append((process, current_time))

    # Calculate average waiting time and turnaround time
    avg_wait_time = wait_time / len(processes)
    avg_turnaround_time = turnaround_time / len(processes)

    # Print results
    print(f"Process\tArrival Time\tBurst Time\tPriority\tWait Time\tTurnaround Time")
    for i in range(len(processes)):
        process, arrival, burst, priority = processes[i]
        wait = completion_time[i][1] - arrival - burst
        turnaround = completion_time[i][1] - arrival
        print(f"{process}\t{arrival}\t\t{burst}\t\t{priority}\t\t{wait}\t\t{turnaround}")

    print(f"\nAverage Waiting Time: {avg_wait_time}")
    print(f"Average Turnaround Time: {avg_turnaround_time}")

# Example Usage
processes = [
    ("P1", 0, 4, 2),  # (process name, arrival time, burst time, priority)
    ("P2", 1, 3, 1),
    ("P3", 2, 2, 4),
    ("P4", 3, 1, 3),
```

```
]

priority_scheduling(processes)
```

4. Producer Consumer problem

USING SEMAPHORE:

```
import threading

import time

import random


# Define the size of the buffer

buffer_size = 5

buffer = []

mutex = threading.Semaphore(1)  # Mutex for exclusive access to the buffer

empty = threading.Semaphore(buffer_size)  # Semaphore to track empty slots

full = threading.Semaphore(0)  # Semaphore to track filled slots


# Maximum number of items to produce

num_items_to_produce = 10


# Producer thread

def producer():

   global num_items_to_produce

   for _ in range(num_items_to_produce):

      item = random.randint(1, 100)  # Generate a random item (data)
```

```
        # Wait until there is space in the buffer
        empty.acquire()

        # Critical section: Adding an item to the buffer
        mutex.acquire()
        buffer.append(item)
        print(f"Produced: {item}")
        mutex.release()

        # Signal that there is an item in the buffer
        full.release()

        # Simulate production time
        time.sleep(random.uniform(0.5, 1))

# Consumer thread
def consumer():
    for _ in range(num_items_to_produce):
        # Wait until there is an item in the buffer
        full.acquire()

        # Critical section: Removing an item from the buffer
        mutex.acquire()
        item = buffer.pop(0)
        print(f"Consumed: {item}")
```

```
        mutex.release()

        # Signal that there is space in the buffer
        empty.release()

        # Simulate consumption time
        time.sleep(random.uniform(0.5, 1))

# Create producer and consumer threads
producer_thread = threading.Thread(target=producer)
consumer_thread = threading.Thread(target=consumer)

# Start the threads
producer_thread.start()
consumer_thread.start()

# Wait for the threads to finish
producer_thread.join()
consumer_thread.join()
```

5. Reader Writer Problem

```
import threading
import random
import time
```

```
# Shared resource (for simulation purposes)

resource = 0


# Semaphores and Locks

read_count = 0  # To count the number of readers accessing the resource

mutex = threading.Semaphore(1)  # Mutex to update the read_count

write_lock = threading.Semaphore(1)  # Semaphore to give exclusive access to writers

read_lock = threading.Semaphore(1)  # Semaphore to control access to reading


# Number of readers and writers to simulate

num_readers = 3

num_writers = 2

timeout = 10  # Run for 10 seconds


# Global counters for completed operations

read_operations = 0

write_operations = 0

start_time = time.time()


# Reader thread

def reader(reader_id):

    global read_count, read_operations

    while time.time() - start_time < timeout:

        # Request access to read

        read_lock.acquire()
```

```
# Start reading
mutex.acquire()
read_count += 1  # Increment reader count
if read_count == 1:
    write_lock.acquire()  # First reader locks the writer
mutex.release()

read_lock.release()

# Reading the shared resource (simulated by printing)
print(f"Reader {reader_id} is reading the resource: {resource}")
time.sleep(random.uniform(1, 2))

# Finished reading
mutex.acquire()
read_count -= 1  # Decrement reader count
if read_count == 0:
    write_lock.release()  # Last reader releases the writer lock
mutex.release()

# Update the operation count
read_operations += 1

# Simulate some time between reads
time.sleep(random.uniform(0.5, 1))
```

```
# Writer thread
def writer(writer_id):
    global write_operations
    while time.time() - start_time < timeout:
        # Request access to write
        write_lock.acquire()

        # Start writing to the resource (simulated by updating the value)
        global resource
        resource = random.randint(1, 100)  # Simulate writing a new value
        print(f"Writer {writer_id} is writing the resource: {resource}")
        time.sleep(random.uniform(2, 3))

        # Finished writing
        write_lock.release()

        # Update the operation count
        write_operations += 1

        # Simulate some time between writes
        time.sleep(random.uniform(1, 2))

# Create reader threads
reader_threads = []
for i in range(num_readers):
```

```
    reader_thread = threading.Thread(target=reader, args=(i+1,))
    reader_threads.append(reader_thread)

# Create writer threads
writer_threads = []
for i in range(num_writers):
    writer_thread = threading.Thread(target=writer, args=(i+1,))
    writer_threads.append(writer_thread)

# Start all reader and writer threads
for thread in reader_threads:
    thread.start()

for thread in writer_threads:
    thread.start()

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

for thread in writer_threads:
    thread.join()

print("Time limit reached. All operations stopped.")
```

6. Problems on awk script

a. To Prepare a report

```
#!/usr/bin/awk -f

# Step 3: Start processing each record

{

    # Step 4: Calculate the total marks

    total = $3 ;


    # Step 5: Calculate the percentage

    percentage = total ;


    # Step 6: Determine the result based on percentage

    if (percentage < 40) {

        result = "Fail";

    } else if (percentage >= 60 && percentage <= 65) {

        result = "First Class";

    } else if (percentage > 66) {

        result = "Distinction";

    } else {

        result = "Pass";

    }


    # Step 7: Print the output for each record

    print "Student: " $1 ", Total Marks: " total ", Percentage: " percentage "%, Result: "
result;

}
```

7. Shell program

a. Palindrome

```
#!/bin/bash

# Ask the user to enter a string
echo "Enter a string:"
read str

# Check if the string is empty
if [ -z "$str" ]; then
  echo "Error: String should not be NULL."
  exit 1
fi

# Initialize variables
len=${#str}
ptr=0
flag=true

# Loop through the string and compare characters
while [ $ptr -lt $((len / 2)) ]; do
  if [ "${str:$ptr:1}" != "${str:$(($len - $ptr - 1)):1}" ]; then
    flag=false
    break
  fi
  ptr=$((ptr + 1))
```

```
done

# Check and print if the string is a palindrome
if [ "$flag" = true ]; then
  echo "$str is a palindrome."
else
  echo "$str is not a palindrome."
fi
```

b. Perform arithmetic operations

```
#!/bin/bash

# Arithmetic operations script

echo "Enter first number:"
read num1

echo "Enter second number:"
read num2

echo "Choose operation: "
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
```

```
read choice


case $choice in
    1)
        result=$((num1 + num2))
        echo "Addition: $num1 + $num2 = $result"
        ;;
    2)
        result=$((num1 - num2))
        echo "Subtraction: $num1 - $num2 = $result"
        ;;
    3)
        result=$((num1 * num2))
        echo "Multiplication: $num1 * $num2 = $result"
        ;;
    4)
        if [ $num2 -eq 0 ]; then
            echo "Error: Division by zero is not allowed."
        else
            result=$((num1 / num2))
            echo "Division: $num1 / $num2 = $result"
        fi
        ;;
    *)
        echo "Invalid choice."
        ;;
```

```
esac
```

c. Number of occurrences

```
#!/bin/bash

# Count occurrences of a character script

echo "Enter a string:"
read str

echo "Enter the character to search for:"
read char

# Count occurrences of the character using wc
count=$(echo "$str" | tr -cd "$char" | wc -c)

echo "The character '$char' appears $count times in the string '$str'."
```

d. Insertion sort.

```
#!/bin/bash

# Insertion sort script

echo "Enter numbers separated by space:"
read -a arr

# Insertion Sort Algorithm
```

```
for ((i = 1; i < ${#arr[@]}; i++)); do
    key=${arr[$i]}
    j=$((i-1))

    # Move elements of arr[0..i-1] that are greater than key to one position ahead
    while [ $j -ge 0 ] && [ ${arr[$j]} -gt $key ]; do
        arr[$((j+1))]=${arr[$j]}
        j=$((j-1))
    done
    arr[$((j+1))]=$key
done
# Print sorted array
echo "Sorted array: ${arr[@]}"
```