# Lab Exercise – 15

AIM :: Implement `Banker’s Algorithm` for Deadlock avoidance using Shell Scripting.

## Theory ::

### Banker’s Algorithm

The **Banker’s Algorithm** is a deadlock avoidance mechanism that helps in safe resource allocation among multiple processes. Here's how it works:

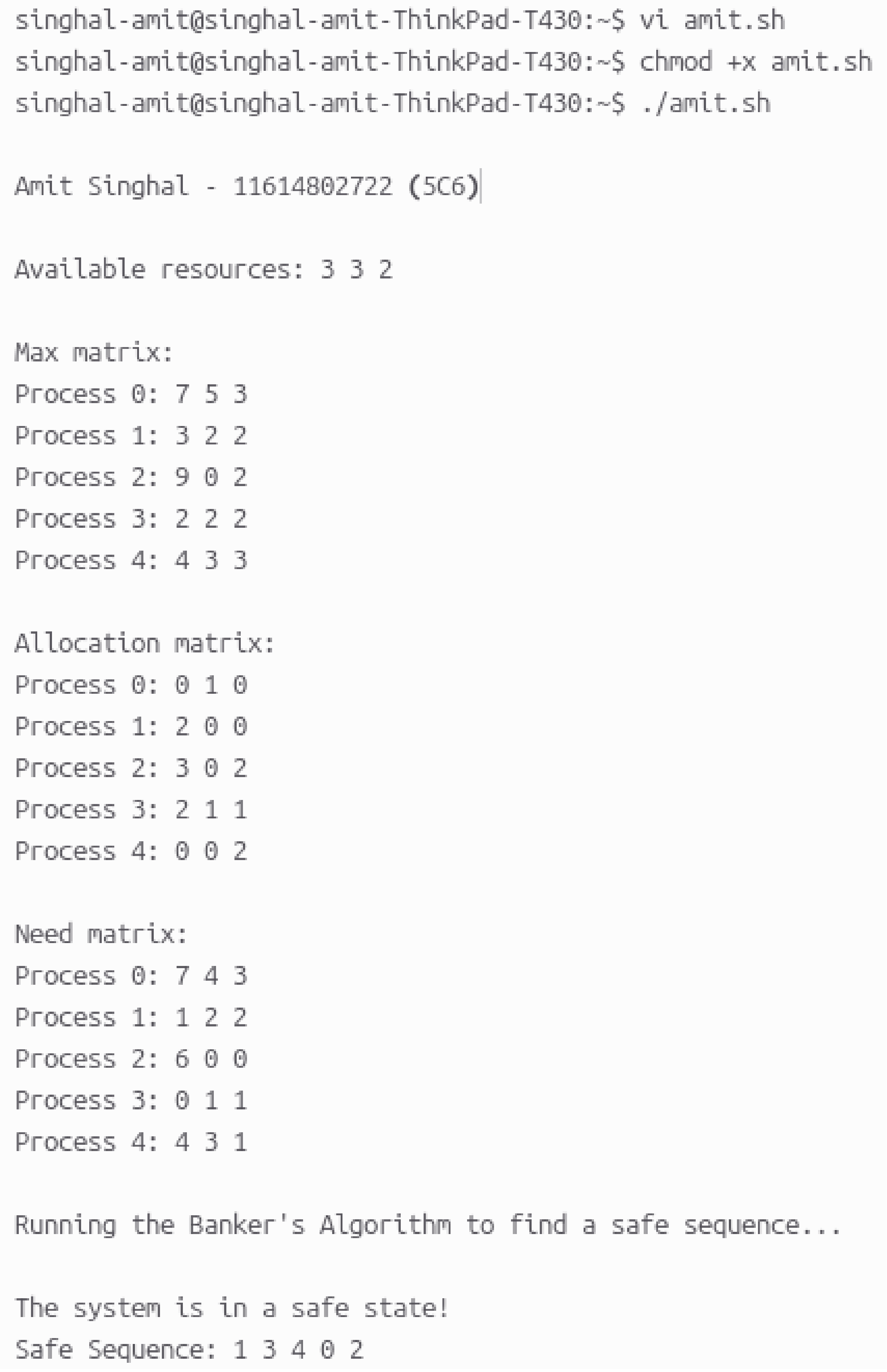
1. **Processes:** These represent programs needing resources like memory or CPU.
2. **Resources:** Finite units requested by processes (e.g., files, memory, etc.).
3. **Safe State:** A state where at least one process sequence can be completed without causing a deadlock.
4. **Unsafe State:** A state where no safe sequence exists, potentially leading to a deadlock.

The algorithm checks requests and ensures that the system always stays in a safe state by calculating safe sequences and allowing or denying resource requests accordingly. Here's a code example to implement this logic, and it will generate a safe or unsafe sequence based on the given resources and allocations.

## Source_Code ::

```
echo "Amit Singhal - 11614802722 (5C6)"


P=5
R=3


available=(3 3 2)


max=(
"7 5 3"
"3 2 2"
"9 0 2"
"2 2 2"
"4 3 3"
)


allocation=(
"0 1 0"
"2 0 0"
"3 0 2"
"2 1 1"
"0 0 2"
)

declare -A need

# Calculate the Need matrix
for ((i=0; i<$P; i++)); do
  for ((j=0; j<$R; j++)); do
    max_value=(${max[i]})
    allocation_value=(${allocation[i]})
    need[$i,$j]=$(( ${max_value[$j]} - ${allocation_value[$j]} ))
  done
done

# Function to print matrices
function print_matrices {
  echo "Available resources: ${available[@]}"

  echo -e "\nMax matrix:"
  for ((i=0; i<$P; i++)); do
    echo "Process $i: ${max[i]}"
  done

  echo -e "\nAllocation matrix:"
  for ((i=0; i<$P; i++)); do
    echo "Process $i: ${allocation[i]}"
  done

  echo -e "\nNeed matrix:"
  for ((i=0; i<$P; i++)); do
    echo -n "Process $i: "
    for ((j=0; j<$R; j++)); do
      echo -n "${need[$i,$j]} "
    done
    echo ""
  done
}

# Function to check if the request is less than or equal to available resources
function is_less_or_equal {
  local process=$1
  for ((i=0; i<$R; i++)); do
    if [ ${need[$process,$i]} -gt ${available[$i]} ]; then
      return 1
    fi
  done
  return 0
}

# Safety algorithm to find if there exists a safe sequence
function safety_algorithm {
  local work=("${available[@]}")
  local finish=()
  local safe_sequence=()

  # Initialize finish array to false for all processes
  for ((i=0; i<$P; i++)); do
    finish[$i]=0
  done

  echo -e "\nRunning the Banker's Algorithm to find a safe sequence..."

  while true; do
    local found=false
    for ((i=0; i<$P; i++)); do
      if [ ${finish[$i]} -eq 0 ]; then
        is_less_or_equal $i
        if [ $? -eq 0 ]; then
          for ((j=0; j<$R; j++)); do
            work[$j]=$(( ${work[$j]} + ${allocation[$i,$j]} ))
          done
          safe_sequence+=($i)
          finish[$i]=1
          found=true
        fi
      fi
    done
    if [ "$found" == false ]; then
      break
    fi
  done

  # Check if all processes are finished
  for ((i=0; i<$P; i++)); do
    if [ ${finish[$i]} -eq 0 ]; then
      echo "The system is in an unsafe state!"
      return 1
    fi
  done

  echo "The system is in a safe state!"
  echo "Safe Sequence: ${safe_sequence[@]}"
  return 0
}

# Print matrices
print_matrices

# Run the safety algorithm
safety_algorithm
```

Output ::



```
singhal-amit@singhal-amit-ThinkPad-T430:~$ vi amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ chmod +x amit.sh
singhal-amit@singhal-amit-ThinkPad-T430:~$ ./amit.sh

Amit Singhal - 11614802722 (5C6)

Available resources: 3 3 2

Max matrix:
Process 0: 7 5 3
Process 1: 3 2 2
Process 2: 9 0 2
Process 3: 2 2 2
Process 4: 4 3 3

Allocation matrix:
Process 0: 0 1 0
Process 1: 2 0 0
Process 2: 3 0 2
Process 3: 2 1 1
Process 4: 0 0 2

Need matrix:
Process 0: 7 4 3
Process 1: 1 2 2
Process 2: 6 0 0
Process 3: 0 1 1
Process 4: 4 3 1

Running the Banker's Algorithm to find a safe sequence...

The system is in a safe state!
Safe Sequence: 1 3 4 0 2
```