REPORT FOR:

BASH SCRIPT THAT EMULATES

SELFISH ROUND ROBIN ALGORITHM

ANTHONY GIBAH
20109235
UNIVERSITY OF WESTMINSTER

TABLE OF CONTENTS

Video Link

REQUIREMENTS

The requirements of this Bash script include the key elements that will enable the script to emulate the behavior of the **Selfish Round Robin** algorithm. These requirements are of various types and various formats ranging from system requirements to input requirements and variable type requirements. Other requirements of the Bash script include its functionalities which include its property of being able to accept any number of processes with various arrival times and emulate the algorithm by running the processes and orienting out their respective running times and waiting times. The output requirement of the script is its ability to correctly output the data to standard output only, a named text file or to both. The key and relevant requirements of this Bash script are listed below in a tabular form. Their importance and relevance to the script are also stated.

Video Link

S/N	Requirement	Requirement Type/Format	Importance	Details
1	Bash Shell	System requirement	Very important	A bash shell needs to be installed on the system that will be used to run the script. Most Unix-based systems come with Bash shell per-installed
2	Operating System	System requirement	Very important	If you are using a Windows OS, you must log in to the University of Westminster's Unix server or download and install a 3rd party tool like Git Bash. If you are using a Mac OS, you do not need to log in to an external server, because Mac OS is built on Linux/Unix based OS.
3	CPU	System Requirement	Important	Due to the very small size of the script and its simplicity, the script does not require much computing power and will therefore run on any basic CPU.
4	Memory	System Requirement	Important	The script does not require much memory to run. It is less than 100KB
5	Data File	Input requirement/ .txt or A file with no extension	Very important	This is the 1st positional parameter. The script requires a data file containing the processes, their required service time and their arrival time. This data will be used as an input to run the script.
6	Data	Input requirement/Strings and Integers	Very important	The data in the data file should be in this format: **Process NUT Arrival Time** A 6 0 B 4 2 C 3 4 Note: The process name comes first, then a space, and the

				service required (NUT) follows, then a space and finally the arrival time.
7	New Queue Priority Increment	Input Requirement/Integer	Very important	This is the 2nd positional parameter. This must be an integer value equal to or greater than 0. Although it can be 0, it is better to set a value greater than 0 for the New Queue priority increment.
8	Accepted Queue Priority Increment	Input Requirement/Integer	Very important	This is the 3rd positional parameter. This must also be an integer value equal to or greater than 0. Although it can be 0, it is better to set a value greater than 0 for the Accepted Queue priority increment.
9	Quanta Number	Input Requirement/Integer	Not important/optional	This is the 4th **optional** positional parameter. The input of a quanta number is optional. If no quanta number is given as input, the default value which is 1, will be used. The quanta number determines how long each process will be allowed to run. It must be an integer value greater than or equal to 1.
10	Output Format	Input Requirement	Important	The user must select how the output of the script will be presented. It will be presented in one of 3 ways. • Output to standard output only. • Output to named text file. • Output to standard output and named text file.
11	Output file	Output requirement	Optional	If the user selects an option to output to text file, the Bash script outputs the data to a file named **output.txt.** Alternatively, the output of the script can be done on the console or standard output.
12	Regular Expressions	Performance requirement	Important	Regular expressions like **=~ ^[0-9]+$** is used to ensure certain user input is an Integer. This prevents the code from failing by reading wrong data formats.
13	Round Robin Algorithm	Performance requirement	Very Important	The script is required to perform the round robin algorithm on the processes on the accepted queue to ensure other processes get time to run too.

DESIGN OF SYSTEM

Video Link

This Bash script is designed to run with positional parameters. The first positional parameter must be a file that contains data of the processes that will run along with their NUT and arrival times respectively. The second positional parameter is the New Queue priority increment value, and the third positional parameter is the Accepted Queue priority increment. The second and third parameters must be integers and this script is designed to use **Regular Expressions** to validate the type of the parameters. The fourth parameter is an optional parameter which is the quanta and must be an integer which will be validated using Regular Expressions. A default Quanta value of 1 will be used if no fourth positional parameter is included. The script also makes use of **"awk"** for processing strings by appending and concatenating strings. Another command line utility used in the design of this script is "grep" which was used to match patterns and find processes in the arrays.

2.1 Design Parameters

The table below is a list of parameters used in the design of the Bash script. It tabulates the type of each parameter, return type and details of the parameter.

S/N	Name	Type	Return Type	Details
1	data	File	-	This is the data containing all the processes, their service required and their arrival time. It will be the 1st positional parameter when the script is run.
2	newQueuePriorityIncrement	Integer	-	This is the value of increment of the priority of processes on the new queue. The new queue processes will increment by this value.
3	acceptedQueuePriorityIncrement			This is the value of increment of the priority of processes on the accepted queue. The accepted queue processes will increment by this value.
4	quanta	Integer	-	This is the quanta number which specifies how long each process is allowed to run.
5	processes	Array	Strings	This is the array of all processes that will run.
6	newQ	Array	Strings	This is an array of processes on the New Queue.
7	acceptedQ	Array	Strings	This is an array of processes on the Accepted Queue.
8	outputType	Integer	-	This is an integer value that the user will input to specify if the output will be to

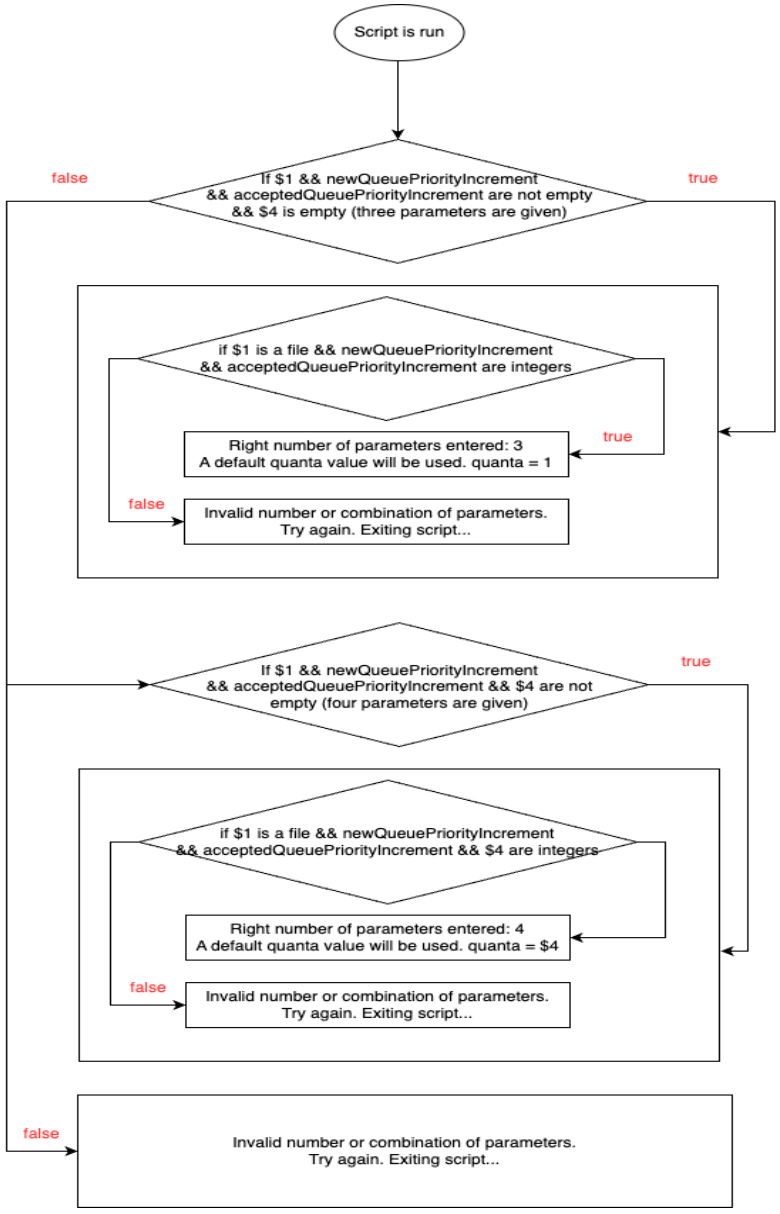
				standard output only, text file or to both.
9	outputFile	String	-	This is the name of the file that the output of the script will be written to.
10	startProcess	Function	Void	This is the function that runs the do-while loop. This is the function that starts and executes the processes.
11	findNewProcess	Function	Void	This function finds processes that are arriving and adds them to their respective queues.
12	performNewQueueOperations	Function	Void	This function checks if any process on the new queue can be moved to the accepted queue and moves the process.
13	performAcceptedQueueOperations	Function	Void	This function checks which process on the accepted queue should be running. It performs the round robin algorithm.
14	removeFinishedProcesses	Function	Void	This removes processes that have finished running. Their service required will be 0.
15	incrementPriorities	Function	Void	This increments the priorities of the new queue and accepted queue.
16	setNewQueueStatusToWaiting	Function	Void	This sets the new queue processes to waiting (W).
17	performMoveToNewQueue	Function	Void	Moves arriving process to new queue.
18	performMoveToAcceptedQueue	Function	Void	Adds a process to accepted queue.
19	getProcessName	Function	String	Returns the name of a particular process
20	getProcessServiceRequired	Function	String	Returns the process service required. This value will be converted to an Integer.
21	getProcessArrivalTime	Function	String	Returns the process arrival time. This value will be converted to an Integer.
22	getProcessPriority	Function	String	Returns the process priority. This value will be converted to an Integer.
23	setAcceptedProcessRunTime	Function	Void	This sets the value of a process in the accepted queue's run time.
24	setNewQueueProcessStatus	Function	Void	Sets the value of the Status of a process in the new queue to W/R/-

2.1 Design Flow Charts

The design flow chart clearly describes the sequence of operations in the bash script. It includes:

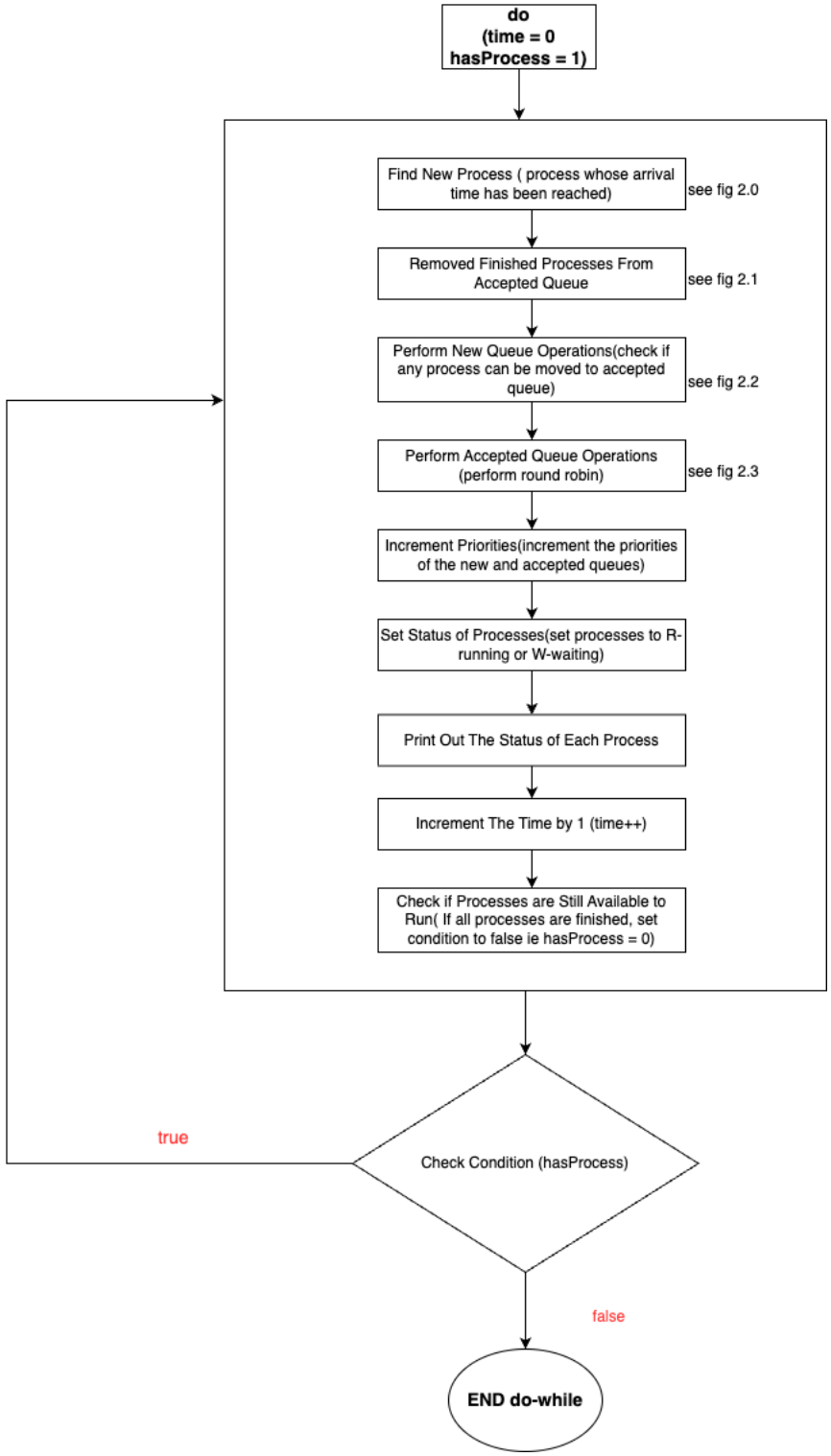
- Flow chart for taking input from a user and validating the input.
- Flow chart for running the processes.
- Flow chart of details of running the processes.

Fig 1.0 Flow chart for accepting and validating input

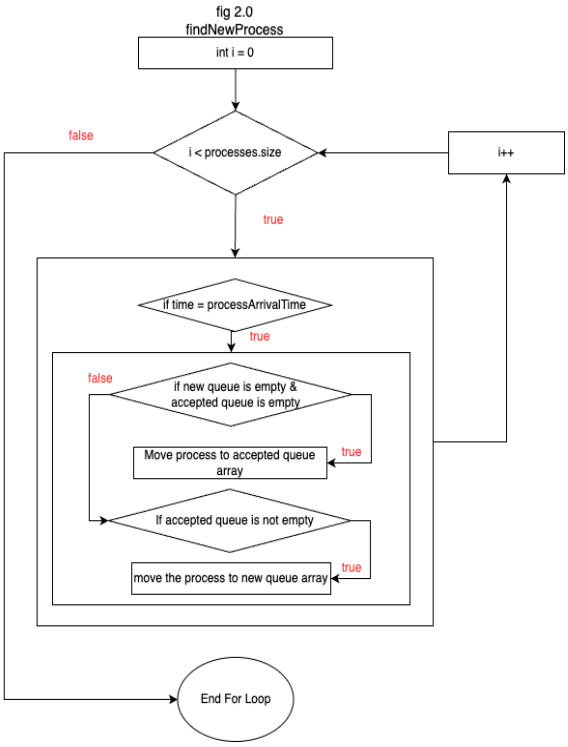


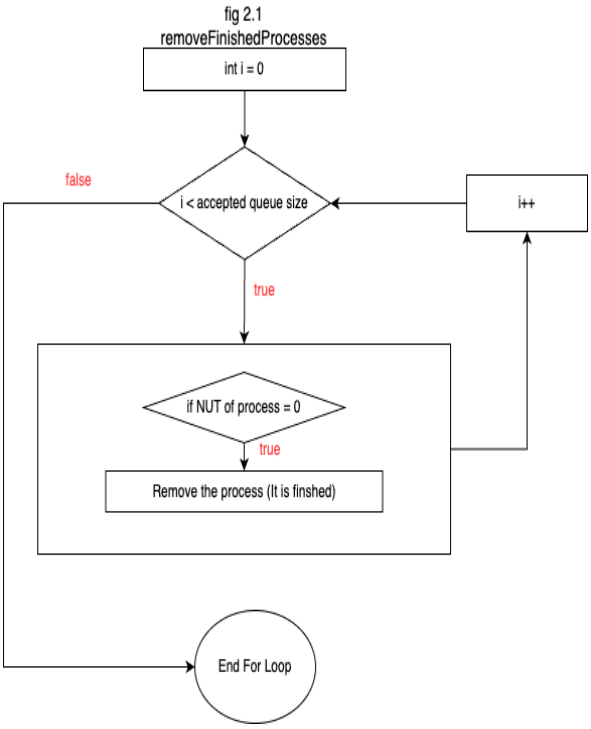
The flow chart above describes how the script is designed to run with positional parameters, validate the parameters to ensure the right number of parameters and types of the parameters are given as input. The flow chart above also shows how a default quanta value of 1 is used if no fourth positional parameter is used to run the script. It makes use of **Regular Expressions** to validate the types of the positional parameters to ensure the inputs are integers and **-f** is used to validate that a file was the first positional parameter.

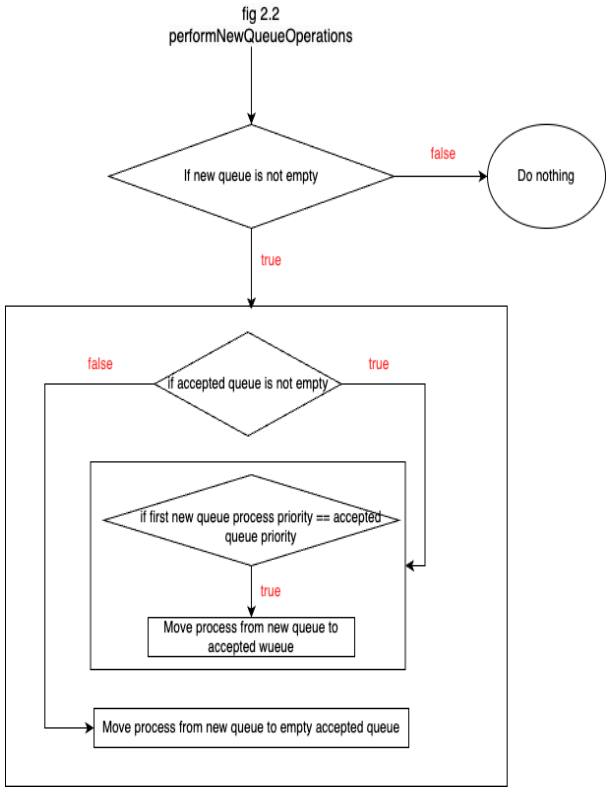
Fig 1.1 Flow chart for running the process.

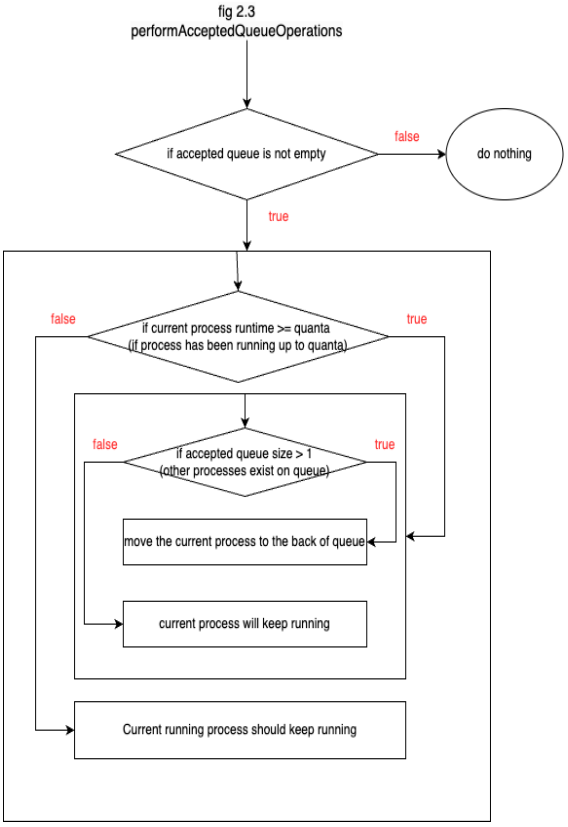


The flow chart above shows a do-while loop that will call a few functions which will implement the Selfish Round Robin algorithm. The loop starts at time = 0 and the time increments by 1 every time the loop is entered. The variable **hasProcess** will be the Boolean that terminates the loop when it becomes 0 (false). The loop will terminate when all processes have arrived, and all the processes have finished running.









The flow charts above show the breakdown of the functions inside the do-while loop shown in **fig 1.1**. They are **if-else** statements and **for loops** that implement the **Selfish Round Robin Algorithm** by adding arriving processes to the new queue if the accepted queue is not empty, and moving processes from the new queue to the accepted queue if the accepted queue is empty or if the priority of the process in the new queue is the same with the priority of the process in the accepted queue.

IMPLEMENTATION

Video Link

```
#!/bin/bash

#
# The script will take 3 positional parameters
# It will also take an optional 4th parameter
# parameter #1 will be data file
# parameter #2 will be newQueuePriorityIncrement
# parameter #3 will be acceptedQueuePriorityIncrement
# parameter #4 will be quanta value. It will be an optional parameter
# If no 4th parameter is set, default quanta value is 1
#

data=$1
newQueuePriorityIncrement=$2
acceptedQueuePriorityIncrement=$3
quanta=$4
processes=() #Array of the processes
newQ=() #Array of New Queue
acceptedQ=() #Array of Accepted Queue
outputType=1 #This is the output form that the user will select. Default is 1,
output to standard output only
outputFile="output.txt"  # Specify the output file name

#This checks if data entered is empty. If any 2 of 4 are empty, number of
parameters is wrong
if [[ ! -z $1 ]] && [[ ! -z $newQueuePriorityIncrement ]] && [[ ! -z
$acceptedQueuePriorityIncrement ]] && [[ -z $4 ]]; then

    #Now check if first parameter is a file and if the other parameters are
integers
    if [[ -f "$1" && "$newQueuePriorityIncrement" =~ ^[0-9]+$ &&
"$acceptedQueuePriorityIncrement" =~ ^[0-9]+$ ]]; then

    # If the first three parameters are included, but not the fourth, print a
message
    echo ""
    echo "Right number of parameters entered: 3."
    echo "A default quanta value will be used. quanta = 1"
    echo "$1 data file entered"
    echo ""

    #Set quanta value to 1
    quanta=1
```

```
    else
            # If the conditions for either case are not met, print an error message
and exit with code 1
    echo "Invalid number or combination of parameters."
    echo ""
    echo "You need to input at least 3 positional parameters"
    echo "1. The data file"
    echo "2. The new queue priotity increment. It must be an integer."
    echo "3. The accepted queue priority increment. It must be an integer."
    echo "4. (Optional) Quanta. If you don't include quanta, value of 1 will be
used"
    echo ""
    echo "Try again. Exiting script..."
    exit 1
    fi

# This checks if all four parameters are included
elif [[ ! -z $1 ]] && [[ ! -z $newQueuePriorityIncrement ]] && [[ ! -z
$acceptedQueuePriorityIncrement ]] && [[ ! -z $4 ]]; then

    #Now check if first parameter is a file and if the other parameters are
integers
    if [[ -f "$1" && "$newQueuePriorityIncrement" =~ ^[0-9]+$ &&
"$acceptedQueuePriorityIncrement" =~ ^[0-9]+$ &&  $4 =~ ^[0-9]+$  ]]; then
            # If all four parameters are included, print a message
    echo ""
    echo "Right number of parameters entered: 4."
    echo "$1 data file entered"
    echo "quanta = $quanta"
    echo ""
    else
            # If the conditions for either case are not met, print an error message
and exit with code 1
    echo "Invalid number or combination of parameters."
    echo ""
    echo "You need to input at least 3 positional parameters"
    echo "1. The data file"
    echo "2. The new queue priotity increment. It must be an integer."
    echo "3. The accepted queue priority increment. It must be an integer."
    echo "4. (Optional) Quanta. If you don't include quanta, value of 1 will be
used"
    echo ""
    echo "Try again. Exiting script..."
    exit 1
    fi
else
    # If the conditions for either case are not met, print an error message and
exit with code 1
    echo "Invalid number or combination of parameters."
    echo ""
    echo "You need to input at least 3 positional parameters"
```

```
    echo "1. The data file"
    echo "2. The new queue priotity increment"
    echo "3. The accepted queue priority increment"
    echo "4. (Optional) Quanta. If you don't include quanta, value of 1 will be
used"
    echo ""
    echo "Try again. Exiting script..."
    exit 1
fi


echo "Data in data file:"

# Read file line by line into the array
while IFS= read -r line || [ -n "$line" ]; do
    # Check if the line is not empty
    if [ -n "$line" ]; then
        processes+=("$line")
        echo "$line"
    fi
done < "$1"


#Display priority increments
echo "Priority Increment in New_Queue = $newQueuePriorityIncrement and in
Accepted_Queue = $acceptedQueuePriorityIncrement"


#
# Now the script will ask user to select one of three options
# 1. Output to standard output only.
# 2. Output to named text file. If the file exists, then it should be overwritten.
# 3. Output to both standard output and named text file.
#

#We will use a switch statement to achieve this
echo -e "\nSelect an option of how you want your output:"
PS3="Enter a number (1-3): "

select option in "Output to standard output only" "Output to named text file. If
file exists, it will be overwritten" "Output to both standard output and named text
file"; do
    case $REPLY in
        1)
            echo -e "You selected: Output to standard output only.\n"
            outputType=1
            break
            ;;
        2)
            echo "You selected: Output to named text file (overwrite if exists)."
            echo -e "Output text file name is $outputFile \n"
            outputType=2
```

```
            break
            ;;
        3)
            echo "You selected: Output to both standard output and named text
file."
            echo -e "Output text file name is $outputFile \n"
            outputType=3
            break
            ;;
        *)
            echo "Invalid selection. Please enter a number between 1 and 3."
            ;;
    esac
done


#
# Now the script will modify the data that was read and add it to an array named
"processes"
# It will use "awk" to append the following
# 1. The priority. It will have an initial value of 0.
# 2. The status of the process. It will have an initial value of -. But will change
to R or W when running or waiting
# 3. The runtime of the process. This is how long the process has been running.
#    It will always be reset to 0 when a process is moved to the back of the queue
#
#

# Modify the array by appending "0,-,0" to each line
for i in "${!processes[@]}"; do
    processes[$i]="$(echo "${processes[$i]}" | awk '{print $1","$2","$3",0,-,0"}')"
done

################################################################################
##################
#
# The actual sequence for the Selfish Round Robin scheduling algorithm,
# from time 0 until all the listed processes have completed will start here.
# It will make use of a Do-While Loop. Such that, It will start from time=0
# and the condition "hasProcess" will be set to 1. When the processes have finished
# running successfully, the hasProcess will be 0, and the Do-while loop will
terminate.
#
# The loop will carry out the following operations:
# 1. "findNewProcess" - This will check for processes that are arriving at time
"time" and
#  add the processes to their respective queue(array)
# 2. "removeFinishedProcesses" - This will check for processes in the accepted
queue that have finished
#  running and remove them.
```

```
# 3. "performNewQueueOperations" - This will check if any process in the new queue
can be moved to the accepted queue
# 4. "performAcceptedQueueOperations" - This will perform the Round Robin algorithm
on the accepted queue (array)
# 5. "incrementPriorities" - This will increment the priority of the new queue with
$newQueuePriorityIncrement
#    and increment the priority of the accepted queue with
$acceptedQueuePriorityIncrement
# 6. "setNewQueueStatusToWaiting" - This will set the status of the new queue to
waiting.
#
#


# Print header (T A B C ...) based on the output type the person selected
if [ "$outputType" -eq 1 ]; then

    # Output to standard output only

    echo -n "T    "

    for process in "${processes[@]}"; do
        processName=$(echo "$process" | cut -d ',' -f 1)
        echo -n "$processName    "
    done
    echo ""

elif [ "$outputType" -eq 2 ]; then

    # Output to text file

    # First the script must clear the output file
    echo -n > $outputFile

    echo -n "T    "  >> "$outputFile"
    for process in "${processes[@]}"; do
        processName=$(echo "$process" | cut -d ',' -f 1)
        echo -n "$processName    "
    done >> "$outputFile"
    echo >> "$outputFile"

elif [ "$outputType" -eq 3 ]; then

    # Output to both standard output and text file

    # First the script must clear the output file
    echo -n > $outputFile

    # Text file output
    echo -n "T    "  >> "$outputFile"
    for process in "${processes[@]}"; do
```

```
        processName=$(echo "$process" | cut -d ',' -f 1)
        echo -n "$processName    "
    done >> "$outputFile"
    echo >> "$outputFile"

    # Strandard output
    echo -n "T    "
    for process in "${processes[@]}"; do
        processName=$(echo "$process" | cut -d ',' -f 1)
        echo -n "$processName    "
    done
    echo ""
fi



startProcess() {

time=0 # Process time
hasProcess=1 # Condition for Do-While loop

while [ "$hasProcess" -eq 1 ]; do

    # Check if a process is arriving
    findNewProcess "$time"

    # Remove finished processes
    removeFinishedProcesses

    # Move process from new Q to accepted Q
    performNewQueueOperations

    # Perform round robin algorithm
    performAcceptedQueueOperations

    # Increment priorities of both queues
    incrementPriorities

    # Set Status to W
    setNewQueueStatusToWaiting



    # Print the status of the processes (W - R - F) based on the output type the
user selected
    if [ "$outputType" -eq 1 ]; then

        # Output to standard output only

        # Print time and process status
        echo -n "$time"
```

```
    # Print process status for each time $time
    for process in "${processes[@]}"; do
        status=$(getProcessStatus "$process")
        echo -n "    $status"
    done

    echo ""

elif [ "$outputType" -eq 2 ]; then

    # Output to text file

    # Print time
    echo -n "$time" >> "$outputFile"

    # Print process status for each time $time
    for process in "${processes[@]}"; do
        status=$(getProcessStatus "$process")
        echo -n "    $status"
    done >> "$outputFile"

    echo >> "$outputFile"


elif [ "$outputType" -eq 3 ]; then

    # Output to both standard output and text file

    # Print time and process status
    echo -n "$time"

    # Print process status for each time $time
    for process in "${processes[@]}"; do
        status=$(getProcessStatus "$process")
        echo -n "    $status"
    done

    echo ""

    # Print time
    echo -n "$time" >> "$outputFile"

    # Print process status for each time $time
    for process in "${processes[@]}"; do
        status=$(getProcessStatus "$process")
        echo -n "    $status"
    done >> "$outputFile"

    echo >> "$outputFile"
```

```
    fi


    # This code is used to check if all processes have arrived by checking for
    #  the process with the highest arrival time and comparing ot to the current
    #  time $time
    highestArrivalTime=0
    for process in "${processes[@]}"; do
        arrivalTime=$(getProcessArrivalTime "$process")
        if [ "$highestArrivalTime" -lt "$arrivalTime" ]; then
            highestArrivalTime=$arrivalTime
        fi
    done

    # Increment time
    ((time++))

    # If all processes have arrived and all processes have finished running, set
hasProcess=0
    if [ "$time" -gt "$highestArrivalTime" ] && [ "${#acceptedQ[@]}" -eq 0 ] && [
"${#newQ[@]}" -eq 0 ]; then
        hasProcess=0
    fi

done
}

# Function to find new processes
findNewProcess() {
    local time=$1

    for ((i = 0; i < ${#processes[@]}; i++)); do
        # Check if a process is arriving and add to the appropriate queue
        newOrAccepted "$time" "$i"
    done
}

# Function to perform new queue operations
performNewQueueOperations() {
    # Check if newQ is not empty
    if [ ${#newQ[@]} -gt 0 ]; then
        processesToRemove=()
        highestPriority=-2147483648  # minimum integer value
        processesWithHighestPriority=()

        # Iterate through processes in newQ
        for currentProcess in "${newQ[@]}"; do
            currentPriority=$(getProcessPriority "$currentProcess")

            if [ "$currentPriority" -eq "$highestPriority" ]; then
                # Add to the list if priority is the same
```

```
                processesWithHighestPriority+=("$currentProcess")
            elif [ "$currentPriority" -gt "$highestPriority" ]; then
                # Update the list and remove previous processes if priority is
higher
                highestPriority="$currentPriority"
                processesWithHighestPriority=("$currentProcess")
                processesToRemove=("$currentProcess")
            fi
        done

        # Check if priority is equal and move to accepted queue
        if [ ${#acceptedQ[@]} -ne 0 ]; then
            if [ "$(getProcessPriority "${processesWithHighestPriority[0]}")" -eq
"$(getProcessPriority "${acceptedQ[0]}")" ]; then
                for process in "${processesWithHighestPriority[@]}"; do
                    performMoveToAcceptedQueue "$process"
                newQ=($(echo "${newQ[@]}" | tr ' ' '\n' | grep -v "$process"))
                done

            fi
        else
            for process in "${processesWithHighestPriority[@]}"; do
                performMoveToAcceptedQueue "$process"
            done

            # Move to accepted queue and remove from newQ
            newQ=($(echo "${newQ[@]}" | tr ' ' '\n' | grep -v
"${processesToRemove[@]}"))

        fi
    fi
}

# Function to perform accepted queue operations (round robin operations)
performAcceptedQueueOperations() {
    if [ "${#acceptedQ[@]}" -gt 0 ]; then

        # Check and set which process should be running
        if [ "${#acceptedQ[@]}" -gt 0 ]; then
        firstProcess=${acceptedQ[0]}

        # check if the process has been running for a period up to the quanta
        if [ "$(getProcessRunTime "$firstProcess")" -ge "$quanta" ]; then

            # If it has, and if it is the only process on the accepted queue, let
it keep running
            if [ "${#acceptedQ[@]}" -eq 1 ]; then
                runTime=$(getProcessRunTime "$firstProcess")
                requiredServiceTime=$(getProcessServiceRequired "$firstProcess")

                ((runTime++)) # Increment the runtime
```

```
                ((requiredServiceTime--)) # decrement service time

                for ((i = 0; i < ${#processes[@]}; i++)); do
                    if [ "$(getProcessName "${processes[i]}")" == "$(getProcessName
"$firstProcess")" ]; then
                        setMainProcessStatus "R" "$i"
                        setMainProcessRunTime "$runTime" "$i"
                        setMainProcessServiceRequired "$requiredServiceTime" "$i"
                    fi
                done

                setAcceptedProcessStatus "R" 0
                setAcceptedProcessRunTime "$runTime" 0
                setAcceptedProcessServiceRequired "$requiredServiceTime" 0
            else
                # It is not the only one on the queue, move it to the back of the
queue

                runTime=0

                for ((i = 0; i < ${#processes[@]}; i++)); do
                    if [ "$(getProcessName "${processes[i]}")" == "$(getProcessName
"$firstProcess")" ]; then
                        setMainProcessStatus "W" "$i"
                        setMainProcessRunTime "$runTime" "$i"
                    fi
                done

                setAcceptedProcessStatus "W" 0
                setAcceptedProcessRunTime "$runTime" 0

                processToMove="${acceptedQ[0]}"

                acceptedQ=("${acceptedQ[@]:1}" "$processToMove")

                newRunTime=$(getProcessRunTime "${acceptedQ[0]}")
                newRequiredServiceTime=$(getProcessServiceRequired
"${acceptedQ[0]}")

                ((newRunTime++))
                ((newRequiredServiceTime--))

                for ((i = 0; i < ${#processes[@]}; i++)); do
                    if [ "$(getProcessName "${processes[i]}")" == "$(getProcessName
"${acceptedQ[0]}")" ]; then
                        setMainProcessStatus "R" "$i"
                        setMainProcessRunTime "$newRunTime" "$i"
                        setMainProcessServiceRequired "$newRequiredServiceTime"
"$i"
                    fi
                done
```

```
                setAcceptedProcessStatus "R" 0
                setAcceptedProcessRunTime "$newRunTime" 0
                setAcceptedProcessServiceRequired "$newRequiredServiceTime" 0

            fi
        else
            # It has not run for a period up to the quanta, let it keep running
            runTime=$(getProcessRunTime "$firstProcess")
            requiredServiceTime=$(getProcessServiceRequired "$firstProcess")

            ((runTime++))
            ((requiredServiceTime--))

            for ((i = 0; i < ${#processes[@]}; i++)); do
                if [ "$(getProcessName "${processes[i]}")" == "$(getProcessName
"$firstProcess")" ]; then
                    setMainProcessStatus "R" "$i"
                    setMainProcessRunTime "$runTime" "$i"
                    setMainProcessServiceRequired "$requiredServiceTime" "$i"
                fi
            done

            setAcceptedProcessStatus "R" 0
            setAcceptedProcessRunTime "$runTime" 0
            setAcceptedProcessServiceRequired "$requiredServiceTime" 0

        fi
    fi
    fi
}

# This function increments the priorities of the new queue and accepted queue
incrementPriorities() {
    if [ "${#newQ[@]}" -gt 0 ]; then
        for ((i = 0; i < ${#newQ[@]}; i++)); do
            priority=$(getProcessPriority "${newQ[i]}")
            ((priority += newQueuePriorityIncrement))
            setNewQueueProcessPriority "$priority" "$i"
        done
    fi

    if [ "${#acceptedQ[@]}" -gt 0 ]; then
        for ((i = 0; i < ${#acceptedQ[@]}; i++)); do
            priority=$(getProcessPriority "${acceptedQ[i]}")
            ((priority += acceptedQueuePriorityIncrement))
            setAcceptedProcessPriority "$priority" "$i"
        done
    fi
}
```

```
# This sets the new queue status to Waiting - W
setNewQueueStatusToWaiting() {
    for ((i = 0; i < ${#processes[@]}; i++)); do
        for ((j = 0; j < ${#newQ[@]}; j++)); do
            setNewQueueProcessStatus "W" "$j"

            if [ "$(getProcessName "${processes[i]}")" == "$(getProcessName
"${newQ[j]}")" ]; then
                setMainProcessStatus "W" "$i"
            fi
        done
    done

}

# Function to check if a process is arriving and print information
newOrAccepted() {
    local time=$1
    local i=$2


    if [ $time -eq $(getProcessArrivalTime "${processes[$i]}") ]; then
        if [ ${#newQ[@]} -eq 0 ] && [ ${#acceptedQ[@]} -eq 0 ]; then
                performMoveToAcceptedQueue "${processes[$i]}"
        else
            # Add logic for newQ and acceptedQ
            performMoveToNewQueue "${processes[$i]}"
        fi
    fi
}

# Moves an arriving process to new queue
performMoveToNewQueue() {
    local process=$1
    newQ+=("$process")
}



# Function to move a process to the accepted queue
performMoveToAcceptedQueue() {
    local process=$1
    acceptedQ+=("$process")
}

removeFinishedProcesses() {
    # These are the processes to be deleted from the Accepted Q
    local processesToRemove=()

    for ((i = 0; i < ${#acceptedQ[@]}; i++)); do
        process=${acceptedQ[i]}
```

```
        if [ "$(getProcessServiceRequired "$process")" -eq 0 ]; then
            processesToRemove+=("$process")
        fi
    done

    # Remove processes from the Accepted Q
    for processToRemove in "${processesToRemove[@]}"; do
        # acceptedQ=("${acceptedQ[@]/$processToRemove}")
        acceptedQ=($(echo "${acceptedQ[@]}" | tr ' ' '\n' | grep -v
"$processToRemove"))

    done

    # Set the status of the process in the main Process List
    for ((i = 0; i < ${#processes[@]}; i++)); do
        for ((i1 = 0; i1 < ${#processesToRemove[@]}; i1++)); do
            if [ "$(getProcessName "${processes[i]}")" == "$(getProcessName
"${processesToRemove[i1]}")" ]; then
                setMainProcessStatus "F" "$i"
            fi
        done
    done

}

# Returns the name of a process
getProcessName() {
    local process=$1
    IFS=',' read -ra processDetails <<< "$process"
    echo "${processDetails[0]}"
}

# Returns the NUT or service required by a process
getProcessServiceRequired() {
    local process=$1
    IFS=',' read -ra processDetails <<< "$process"
    echo "${processDetails[1]}"
}

# Function to extract arrival time from process string
getProcessArrivalTime() {
    local process=$1
    IFS=',' read -ra processDetails <<< "$process"
    echo "${processDetails[2]}"
}

# Returns the priority of a process
getProcessPriority() {
    process="$1"
    IFS=',' read -ra processDetails <<< "$process"
```

```
    echo "${processDetails[3]}"
}

# Returns the status of a process
getProcessStatus() {
    local process=$1
    IFS=',' read -ra processDetails <<< "$process"
    echo "${processDetails[4]}"
}

# Returns the runtime of a process (The runtime is how long the process has been
running)
getProcessRunTime() {
    local process=$1
    IFS=',' read -ra processDetails <<< "$process"
    echo "${processDetails[5]}"
}

# Sets the value of a process service required in the main processes array
setMainProcessServiceRequired() {
    local serviceRequiredTime=$1
    local i=$2

    local currentProcess=${processes[i]}
    local processName=$(getProcessName "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local status=$(getProcessStatus "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    processes[i]=$(IFS=','; echo
"$processName,$serviceRequiredTime,$arrivalTime,$priority,$status,$runTime")

}

# Sets the value of a process status in the main processes array
setMainProcessStatus() {
    local status=$1
    local i=$2

    local currentProcess=${processes[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    processes[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runTime")
}
```

```
# Sets the value of a process runtime in the main processes array
setMainProcessRunTime() {
    local runtime=$1
    local i=$2

    local currentProcess=${processes[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local status=$(getProcessStatus "$currentProcess")

    processes[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runtime")

}

# Sets the value of a process priority in the new queue array
setNewQueueProcessPriority() {
    local priority=$1
    local i=$2

    local currentProcess=${newQ[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local status=$(getProcessStatus "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    newQ[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runTime")
}

# Sets the value of a process status in the new queue array
setNewQueueProcessStatus() {
    local status=$1
    local i=$2

    local currentProcess=${newQ[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    newQ[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runTime")
}

# Sets the value of a process service required in the accepted queue array
setAcceptedProcessServiceRequired() {
```

```
    local serviceRequiredTime=$1
    local i=$2

    local currentProcess=${acceptedQ[i]}
    local processName=$(getProcessName "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local status=$(getProcessStatus "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    acceptedQ[i]=$(IFS=','; echo
"$processName,$serviceRequiredTime,$arrivalTime,$priority,$status,$runTime")
}

# Sets the value of a process priority in the accepted queue array
setAcceptedProcessPriority() {
    local priority=$1
    local i=$2

    local currentProcess=${acceptedQ[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local status=$(getProcessStatus "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    acceptedQ[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runTime")
}

# Sets the value of a process status in the accepted queue array
setAcceptedProcessStatus() {
    local status=$1
    local i=$2

    local currentProcess=${acceptedQ[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local runTime=$(getProcessRunTime "$currentProcess")

    acceptedQ[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runTime")
}

# Sets the value of a process runtime in the accepted queue array
setAcceptedProcessRunTime() {
    local runtime=$1
    local i=$2
```

```
    local currentProcess=${acceptedQ[i]}
    local processName=$(getProcessName "$currentProcess")
    local serviceRequired=$(getProcessServiceRequired "$currentProcess")
    local arrivalTime=$(getProcessArrivalTime "$currentProcess")
    local priority=$(getProcessPriority "$currentProcess")
    local status=$(getProcessStatus "$currentProcess")

    acceptedQ[i]=$(IFS=','; echo
"$processName,$serviceRequired,$arrivalTime,$priority,$status,$runtime")
}

# This is the function that runs the Selfish Round Robin Algorithm
startProcess
```

TESTING

Video Link

Test Case A:

Process Name	Service Required	Arrival Time	Expected Finish Time	Observed Finish Time
A	4	0	4	4
B	5	3	15	15
C	4	2	10	10
D	6	5	19	19

New Queue Priority Increment = 2
Accepted Queue Priority Increment = 1
Quanta = 2

Expected Output					Observed Output				
T	**A**	**B**	**C**	**D**	**T**	**A**	**B**	**C**	**D**
0	R	-	-	-	0	R	-	-	-
1	R	-	-	-	1	R	-	-	-
2	R	-	W	-	2	R	-	W	-
3	R	W	W	-	3	R	W	W	-
4	F	W	R	-	4	F	W	R	-
5	F	W	R	W	5	F	W	R	W
6	F	R	W	W	6	F	R	W	W
7	F	R	W	W	7	F	R	W	W
8	F	W	R	W	8	F	W	R	W
9	F	W	R	W	9	F	W	R	W
10	F	R	F	W	10	F	R	F	W
11	F	R	F	W	11	F	R	F	W
12	F	W	F	R	12	F	W	F	R
13	F	W	F	R	13	F	W	F	R
14	F	R	F	W	14	F	R	F	W
15	F	F	F	R	15	F	F	F	R
16	F	F	F	R	16	F	F	F	R
17	F	F	F	R	17	F	F	F	R
18	F	F	F	R	18	F	F	F	R
19	F	F	F	F	19	F	F	F	F

```
tony@eroam-120-21 Bash % ./srr.sh data 2 1 2

Right number of parameters entered: 4.
data data file entered
quanta = 2

Data in data file:
A       4       0
B       5       3
C       4       2
D       6       5
Priority Increment in New_Queue = 2 and in Accepted_Queue = 1

Select an option of how you want your output:
1) Output to standard output only
2) Output to named text file. If file exists, it will be overwritten
3) Output to both standard output and named text file
Enter a number (1-3): 3
You selected: Output to both standard output and named text file.
Output text file name is output.txt

T    A    B    C    D
0    R    -    -    -
1    R    -    -    -
2    R    -    W    -
3    R    W    W    -
4    F    W    R    -
5    F    W    R    W
6    F    R    W    W
7    F    R    W    W
8    F    W    R    W
9    F    W    R    W
10    F    R    F    W
11    F    R    F    W
12    F    W    F    R
13    F    W    F    R
14    F    R    F    W
15    F    F    F    R
16    F    F    F    R
17    F    F    F    R
18    F    F    F    R
19    F    F    F    F
tony@eroam-120-21 Bash %
```

Test Case A: Screenshot

Test Case B:

Process Name	Service Required	Arrival Time	Expected Finish Time	Observed Finish Time
A	7	0	10	10
B	3	2	9	9
C	8	4	24	24
D	4	5	22	22
E	2	6	21	21

New Queue Priority Increment = 3
Accepted Queue Priority Increment = 2
Quanta = 3

Expected Output						Observed Output					
T	**A**	**B**	**C**	**D**	**E**	**T**	**A**	**B**	**C**	**D**	**E**
0	R	-	-	-	-	0	R	-	-	-	-
1	R	-	-	-	-	1	R	-	-	-	-
2	R	W	-	-	-	2	R	W	-	-	-
3	R	W	-	-	-	3	R	W	-	-	-
4	R	W	W	-	-	4	R	W	W	-	-
5	R	W	W	W	-	5	R	W	W	W	-
6	W	R	W	W	W	6	W	R	W	W	W
7	W	R	W	W	W	7	W	R	W	W	W
8	W	R	W	W	W	8	W	R	W	W	W
9	R	F	W	W	W	9	R	F	W	W	W
10	F	F	R	W	W	10	F	F	R	W	W
11	F	F	R	W	W	11	F	F	R	W	W
12	F	F	R	W	W	12	F	F	R	W	W
13	F	F	W	R	W	13	F	F	W	R	W
14	F	F	W	R	W	14	F	F	W	R	W
15	F	F	W	R	W	15	F	F	W	R	W
16	F	F	R	W	W	16	F	F	R	W	W
17	F	F	R	W	W	17	F	F	R	W	W
18	F	F	R	W	W	18	F	F	R	W	W
19	F	F	W	W	R	19	F	F	W	W	R
20	F	F	W	W	R	20	F	F	W	W	R
21	F	F	W	R	F	21	F	F	W	R	F
22	F	F	R	F	F	22	F	F	R	F	F
23	F	F	R	F	F	23	F	F	R	F	F
24	F	F	F	F	F	24	F	F	F	F	F

```
tony@eroam-120-21 Bash % ./srr.sh data 3 2 3

Right number of parameters entered: 4.
data data file entered
quanta = 3

Data in data file:
A       7       0
B       3       2
C       8       4
D       4       5
E       2       6
Priority Increment in New_Queue = 3 and in Accepted_Queue = 2

Select an option of how you want your output:
1) Output to standard output only
2) Output to named text file. If file exists, it will be overwritten
3) Output to both standard output and named text file
Enter a number (1-3): 3
You selected: Output to both standard output and named text file.
Output text file name is output.txt

T    A    B    C    D    E
0    R    -    -    -    -
1    R    -    -    -    -
2    R    W    -    -    -
3    R    W    -    -    -
4    R    W    W    -    -
5    R    W    W    W    -
6    W    R    W    W    W
7    W    R    W    W    W
8    W    R    W    W    W
9    R    F    W    W    W
10    F    F    R    W    W
11    F    F    R    W    W
12    F    F    R    W    W
13    F    F    W    R    W
14    F    F    W    R    W
15    F    F    W    R    W
16    F    F    R    W    W
17    F    F    R    W    W
18    F    F    R    W    W
19    F    F    W    W    R
20    F    F    W    W    R
21    F    F    W    R    F
22    F    F    R    F    F
23    F    F    R    F    F
24    F    F    F    F    F
tony@eroam-120-21 Bash %
```

Test Case B: Screenshot

Test Case C:

This test case shows how the bash script handles an input of positional parameters that are not the right type.

```
tony@Anthonys-MacBook-Pro Bash % ./srr.sh data e e e
Invalid number or combination of parameters.

You need to input at least 3 positional parameters
1. The data file
2. The new queue priotity increment. It must be an integer.
3. The accepted queue priority increment. It must be an integer.
4. (Optional) Quanta. If you don't include quanta, value of 1 will be used

Try again. Exiting script...
tony@Anthonys-MacBook-Pro Bash %
```

Shown in the image below is the output produced when non-integer values are entered as the 2nd, 3rd, and 4th positional parameters. The Script uses **Regular Expressions** to check if the values are integers.

```
tony@Anthonys-MacBook-Pro Bash % ./srr.sh 3 2 1 2
Invalid number or combination of parameters.

You need to input at least 3 positional parameters
1. The data file
2. The new queue priotity increment. It must be an integer.
3. The accepted queue priority increment. It must be an integer.
4. (Optional) Quanta. If you don't include quanta, value of 1 will be used

Try again. Exiting script...
tony@Anthonys-MacBook-Pro Bash %
```

The image above shows the output of the script when an integer (not a file) is entered as the 1st positional parameter instead of a file. The script uses **-f** to check if input is. Regular file.

Video Link

EVALUATION

Algorithm Implementation: Upon critical analysis and after carrying out various tests on the Bash script, it is certain that the script successfully emulates the selfish round robin algorithm. The algorithm was successfully implemented through a do-while loop and a series of if-else conditions and for loops. The success of this script in emulating the selfish round robin algorithm is proof that the algorithm can be used in successfully scheduling processes to optimize the use of the CPU time.

Implementation Limitations: Although the Selfish Round Robin algorithm was successfully implemented in this Bash script, it is noteworthy that Bash has a lot of limitations that make it difficult to implement this algorithm. Some of its limitations can be found in the way Bash manipulates arrays. Another limitation can be found in the way Bash handles and manipulates different data types. These limitations were somewhat overcome with the use of certain commands like “grep” and “sed” and “awk”. Regular Expressions were also used to handle data types quite successfully.

Input Data Inconsistency: Data entered as the first positional parameter may lead to incorrect output if the data is not arranged in the manner required by the Bash script. This makes the script prone to error if the author of the script fails to provide the format in which the data must be provided. For instance, in a process is provided such that:
A – 0 – 6 = Process name – Arrival time – Service Required
If the script expects the data in another format like:
Process name – Service Required – Arrival time, this will lead to a different output from the output expected. This makes the script susceptible to errors.

Portability: Although the Bash script has its limitations, the algorithm can easily be implemented by any other programming language like Java by following the same design patterns used in writing the code in Bash. This makes this implementation very portable.

System Requirement Limitation: The Bash script runs on Linux/Unix based systems; therefore, it is limited to certain Operating systems. For instance, a Windows OS will not run the bash script directly unless a tool like Git Bash is installed on the system or if the Windows system logs into a Unix Server. Mac OS however runs it directly because it is a Unix/Linux based Operating System. This therefore means Windows OS without access to a Linux/Unix server and a Bash tool installed cannot run this Bash Script, making it limited.

Code Readability: The code in this Bash Script is relatively easy to read because it was grouped into functions that implement different aspects and functionalities of the algorithm.

In conclusion, the Bash script successfully emulates the Selfish Round Robin Algorithm and has been successfully tested with various quanta, processes, NUT and arrival times. A video clearly showing the Bash Script running and displaying the right output has been created and a link to the video can be found here Video Link.

References

1. Charalambous, G. (2023). *L3 Bash Part 1.* London: University of Westminster.
2. Charalambous, G. (2023). *L4 Bash Part 2.* London: University of Westminster.
3. Charalambous, G. (2023). *File & Regular Expressions.* London: University of Westminster.
4. Charalambous, G. (2023). *Grep sed.* London: University of Westminster.
5. Charalambous, G. (2023). *awk.* London: University of Westminster.