islington college
(इस्लिङ्टन कलेज)

LONDON METROPOLITAN UNIVERSITY

Module Code & Module Title

CS5001NA Networks and Operating System

Assessment Weightage & Type

20% Individual Coursework

Year and Semester

2020-21 Autumn

Student Name: Sharnam Dhakhwa

London Met ID: 20049037

College ID: np01cp4s210114

Assignment Due Date: 25th April 2022

Assignment Submission Date: 23rd April, 2022

Title: UNIX Script Programming (Task A)

Process Management (Task B)

Word Count (Task B): 2185

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Table of Contents

Table of Contents

Table of Figures

Abstract

The following coursework compromises two tasks: Task A and Task B. The first task i.e. Task A is to develop a script in the UNIX shell which require us to build a program that acts as a game which includes interactive inputs, validation of the inputs and the errors, sending proper message when an error is seen, and the testing of the entire script. The second task i.e. Task B is to write a technical report that focuses on Process Management which is to be based on different research from different sources like websites, journals, books, and so on.

Task A: UNIX Script Programming

Introduction

Shell is a particular UNIX tool that understands all commands from the terminal and launches the user's program. It allows users to type commands into the terminal manually or to execute commands that have been programmed in "Shell Scripts" automatically. In the first task, we are required to make a simple guessing game that require to be written in a simple script program. We are also required to do a black box testing to check for validation that are done to the program for the inputs provides by the user. The task also requires three files that with content which is necessary for the script to read the contents on the file and display it to the user.

Aims and Objectives

Aims

- To create a script that has simple interactions with the user in UNIX environment that includes simple input and output operations.
- To validate and check the users input.
- To give proper messages when undesired inputs are given

Objectives

- Utilizing various loops, select statements, functions, if statements, arrays to build the program
- Utilizing commands like echo and cat to display different messages and also read the content of the files and display it to the user
- Carrying out Black box testing for validation of the inputs

Script

```
#!/bin/bash
username="$1" # Stores the 1st arguement which is the name
idNumber="$2"  # Stores the 2nd arguement which is the ID
re='^[0-9]+$'  # Helps to check if there is any numbers in name
PASSCT=1       # Password counter to count how many times password is wrong
tries=2        # Keeps track of the amount of tries left for the user entering
the password
STOP="\e[0m"   # Helps to remove any text decorations
CYAN="\e[96m"  # color assignment
BLUE="\e[34m"  # color assignment
BLINK="\e[5m"  # makes the text blink

function print_centered # Function for centering texts and also make row
decorations Expencted Use: (print_centered Text)
{
    [[ $# == 0 ]] && return 1
    declare -i TERM_COLS="$(tput cols)"  #get window size
    declare -i str_len="${#1}"

    [[ $str_len -ge $TERM_COLS ]] && {
        echo "$1";
        return 0;
    }

    declare -i filler_len="$(( (TERM_COLS - str_len) / 2 ))" #deviding the
window to half by calculating
    [[ $# -ge 2 ]] && ch="${2:0:1}" || ch=" "
    filler=""
    for (( i = 0; i < filler_len; i++ )); do
        filler="${filler}${ch}"
    done
    printf "%s%s%s" "$filler" "$1" "$filler"  #adding filler/space to ceneter
the content respectively
    [[ $(( (TERM_COLS - str_len) % 2 )) -ne 0 ]] && printf "%s" "${ch}"
    printf "\n"
    return 0
}


function header()    # Function for decorating the header Expected Use: (header
borderColor textColor title figletDecoration)
{
    printf "$1"
    print_centered " " " " "
    printf "${STOP}"
    printf "$2"
    figlet -f "$4" -t -c "$3"
    printf "${STOP}"
    printf "$1"
    print_centered " " " " "
    printf "${STOP}"
}
```

```
function border()     # Function for border decoration Expected Use:(border
color)
{
    printf "$1"
    print_centered " " " " "
    printf "${STOP}"
}

function check_arguement(){  # Function to check the arguements are typed in
properly
    if [ $# -lt 2 ]              # Check if there is less than 2 arguements
    then
        echo -e "\e[91m(Arguement insufficient)\e[93m\e[1m Usage\e[0m:\e[93m
Please Enter the command line argument as follows:\e[93m\e[5m (./20049037cw2pii
FirstName IdNumber)\e[0m"
    elif [ $# -gt 2 ]            # Check if there is more than 2 arguements
    then
        echo -e "\e[91m(Arguement limit exceeded)\e[93m\e[1m Usage\e[0m:\e[93m
Please Enter the command line argument as follows:\e[93m\e[5m (./20049037cw2pii
FirstName IdNumber)\e[0m"
    elif [[ $username =~ $re ]]  # Checks if there is any numbers in the users
Name
    then
        echo -e "\e[91m(First Name error)\e[93m\e[1m Usage\e[0m:\e[93m Please
Enter the command line argument as follows:\e[93m\e[5m (./20049037cw2pii
FirstName IdNumber)\e[0m"
    elif ! [[ $idNumber =~ $re ]] #  Checks if id conytains only numbers or not
    then
        echo -e "\e[91m(ID error)\e[93m\e[1m Usage\e[0m:\e[93m Please Enter the
command line argument as follows:\e[93m\e[5m (./20049037cw2pii FirstName
IDNumber)\e[0m"
    else
        echo;echo;
        print_centered "-" "-"
        echo;
        echo -e "\e[36mPlease enter the password to continue in the
program\e[0m"
        echo
        password # Executes password function
    fi
}


function password()    # Function for input and checking of the password from
the user
{
    SECRET_KEY="SDhakz"                # Secret KEY
    echo -e -n "Password: "
    read -s MYPWD
    if [ "$MYPWD" = "$SECRET_KEY" ]      # Checks if user inputted password
matches the Secret key
    then
        echo -e "\e[40;38;5;82mPassword accepted\e[0m"
        echo
        print_centered "-" "-"
        PASSCT=1
```

```
        start                           # Starting of the game
    else
        while [ $PASSCT -lt 4 ]         # Loop for number of tries for password
        do
            echo -e "\e[41mWrong password you have \e[7m $tries \e[0m\e[41m
tries left\e[0m "
            (( PASSCT++ ))
            (( tries-- ))
            break
        done

        if [ $PASSCT == 4 ]             # Terminates program if the user exceeds
the amount of password tries
        then
            echo;echo -e "\e[91m\e[5mYou have entered the password wrong 3
times, the program will be terminated\e[0m";echo
        else
            password                    # Re-executes password function
        fi
    fi
}


function start ()      # Function to print the user details along with execution
time and date
{
    now=$(date +%Y-%b-%a)       # Current Date assignment
    time=$(date +%T)            # Current time assignment
    echo;border "\e[34m\e[7m";echo
    printf "${BLINK}"
    printf "${CYAN}"
    figlet -t -c "Welcome to the Band game"
    printf "${STOP}"
    echo -e "\n";border "\e[34m\e[7m";echo

    function details()  # Function to print the details of the user
    {
        echo
        figlet -c -t -f digital "Your details"
        echo
        print_centered "Username: $username    "
        print_centered "User ID: $idNumber    "
        print_centered "Execution Date: $now    "
        print_centered "Execution Time: $time    "
    }
    print_centered "-" "-" | lolcat
    details | boxes -d parchment -a c | lolcat -a -d 1
    echo
    print_centered "-" "-" | lolcat
    sleep 0.5
    menu # Executes menu function
}

function menu() # Function for menu contents
{
    function menu_list() # Function for listing the menu items
```

```
    {
        echo
        print_centered "1) Start Game                    ";echo
        print_centered "2) About Game                    ";echo
        print_centered "3) Exit  Game                    ";echo
    }

    function menu_input()  # Function for taking users menu input and validation
    {
        echo
        printf "\e[38;5;227m"
        print_centered "-" "-"
        printf "${STOP}"
        echo
        echo -e -n "\e[93mPlease enter a number(1-3): \e[0m";read menuItem
        case $menuItem in
            1)
                echo -e "\n\e[92mGame started successfully\e[0m\n"
                border "\e[33m\e[7m"
                guess_band  # Executes guess_band function which is the guessing
band phase
            ;;
            2)
                echo;header   "\e[91m\e[7m"   "\e[38;5;204m"   "About   Info"
"small";echo
                printf "\e[91m";cat ./about.txt;printf "${STOP}"   # Opents the
about info text file
                echo
                border "\e[91m\e[7m"
                check menu exit "Do you want to go back to the menu (yes/no):
"
            ;;
            3)
                quit
            ;;
            *)
                echo -e "\e[91mPlease type valid number between 1 and 3\e[0m"
                menu_input
        esac
    }
    echo;header "\e[33m\e[7m" "\e[93m" "MENU" "small"
    printf "\e[38;5;226m"
    echo;menu_list |boxes -d scroll
    printf "${STOP}"
    menu_input # Calling menu_input function
}

function quit()  # Function for terminating the program
{
    echo;header  "\e[34m\e[7m"  "\e[96m"  "Thank  You  for  playing  the  game"
"standard";echo
    exit
}

function guess_band()  # Function for guessing the band
{
```

```
    echo
    header "\e[36m\e[7m" "\e[96m" "Guess the best Band" "small"

    function band_codes() # Function for displaying the Band names along with
the codes
    {
        echo;
        print_centered   "AC/DC          ------------------------------     AD
";echo
        print_centered   "Beatles      ------------------------------      BE
";echo
        print_centered   "Blondie      ------------------------------      BLO
";echo
        print_centered   "Nirvana      ------------------------------      NIR
";echo
        print_centered   "Queen          ------------------------------     QUE
";echo
    }
    echo;printf   "${CYAN}"   ;band_codes   |   boxes   -d   scroll-akn;printf
"${STOP}";echo       # Calling band_codes function with decorations
    printf "\e[36m";print_centered "-" "-";printf "${STOP}"

    function guess()    # Function for input and validation of the codes entered
by the user
    {
        echo
        echo -e -n "\e[93mEnter the band code: \e[0m";read bandCode
        echo
        case $bandCode in
            "BE")
                echo
                printf "\e[91m\e[5m";figlet -f small -t -c "Wrong Answer" |
boxes -d nuke ;printf "${STOP}"
                header "\e[31m\e[7m" "\e[91m" "Sorry Beatles is not the correct
band" "small"
                check guess_band menu "Do you want to retry (yes/no): "
"\e[31m\e[7m"                    # Executing the check function which executes
guess_band function if yes or executes menu function if no is pressed
            ;;
            "AD")
                echo
                printf "\e[91m\e[5m";figlet -f small -t -c "Wrong Answer" |
boxes -d nuke ;printf "${STOP}"
                header "\e[31m\e[7m" "\e[91m" "Sorry AC/DC is not the correct
band" "small"
                check guess_band menu "Do you want to retry (yes/no): "
"\e[31m\e[7m"                    # Executing the check function which executes
guess_band function if yes or executes menu function if no is pressed
            ;;
            "QUE")
                border "\e[36m\e[7m"
                echo
                header  "\e[32m\e[7m"  "\e[92m"  "Congratulations  you  have
selected the correct band" "small"
                sleep 0.4
                echo
```

```
                printf "\e[95m\e[5m";figlet -f slant -t -c QUEEN;printf
"${STOP}"
                echo
                cat ./QUE/Queen.txt;
                check choose_members menu "Do you want to go to the next stage?
(yes/no): " "\e[32m\e[7m" # Executing the check function which executes
choose_members function if yes or executes menu function if no is pressed
            ;;
            "BLO")
                echo
                printf "\e[91m\e[5m";figlet -f small -t -c "Wrong Answer" |
boxes -d nuke ;printf "${STOP}"
                header "\e[31m\e[7m" "\e[91m" "Sorry Blondie is not the correct
band" "small"
                check guess_band menu "Do you want to retry (yes/no): "
"\e[31m\e[7m"                    # Executing the check function which executes
guess_band function if yes or executes menu function if no is pressed
            ;;
            "NIR")
                echo
                printf "\e[91m\e[5m";figlet -f small -t -c "Wrong Answer" |
boxes -d nuke ;printf "${STOP}"
                header "\e[31m\e[7m" "\e[91m" "Sorry Nirvana is not the correct
band" "small"
                check guess_band menu "Do you want to retry (yes/no): "
"\e[31m\e[7m"                    # Executing the check function which executes
guess_band function if yes or executes menu function if no is pressed
            ;;
            *)
                echo -e "\e[93m\e[1mUsage\e[0m:\e[91m Please enter one of the
codes as shown\e[93m\e[5m (AD/BE/BLO/QUE/NIR)\e[0m"
                guess # Re-executes guess function
        esac
    }
    guess # Calling guess function
}


function check() # Function to check if user wants to go to certaing stage when
there is YES or NO option presented Expected Use: ( check function(case:yes)
function(case:no) message borderColor)
{
    echo -e "\n"
    printf "\e[32m"
    print_centered "-" "-"
    print_centered "-" "-"
    printf "${STOP}"
    printf "\e[92m"
    echo -e -n "$3"
    printf "${STOP}"
    read stage2
    upper=${stage2^^}
    echo
    if [[ $upper == "YES" ]]
    then
        border "$4"
```

```
        $1
    elif [[ $upper == "NO" ]]
    then
        $2
    else
        echo -e "Usage:\e[91m Please enter yes or no\e[0m"
        check "$@"  # Re-executes check function passing the same arguements
    fi
}


function choose_members()  # Function for display input and validation of the
codes of the members inputted by the user
{
    echo
    header "\e[36m\e[7m" "\e[96m" "Choose Any 3 members" "small"

    function member_codes() # Function for displaying the members and their
codes
    {
        echo
        print_centered  "John  Lennon           ------------------------    JL
";echo
        print_centered  "Agnus  Young           ------------------------    AY
";echo
        print_centered   "Freddie   Mercury     ------------------------     FM
";echo
        print_centered  "Debbie  Harry          ------------------------    DH
";echo
        print_centered  "Kurt  Cobain           ------------------------    KC
";echo
    }
    echo;printf   "${CYAN}"   ;member_codes   |   boxes   -d   scroll-akn;printf
"${STOP}";echo # Executes member_codes fuhnction with decorations

    function choice() #function to check if user inputs 3 codes correctly
       {
        printf "\e[36m"
        print_centered "-" "-"
        printf "${STOP}"
        echo -e -n "\e[93mEnter the options from above: \e[0m "
        read M1 M2 M3
                function checking() # Function for checking if the user inputs
correct amount of arguements and correct codes
                {
                in1=$1  # 1st code
                in2=$2  # 2nd code
                in3=$3  # 3rd code
                c=$#    # Number of arguements passed in the checking function
                myArray=("JL" "AY" "FM" "DH" "KC")
                if [[ $c -eq  3 ]] # Checks if there are 3 arguements
                then
                        if [[ " ${myArray[*]} " =~ " ${in1} " && " ${myArray[*]}
" =~ " ${in2} " && " ${myArray[*]} " =~ " ${in3} " ]]  # Checks if the inputted
codes matches the ones in the array
                        then
```

```
                            if [[ $in1 != $in2 && $in1 != $in3 && $in2 != $in3
]] # Checks if the inputted codes are duplicate
                                    then
                                        echo -e "\e[92mSuccess\e[0m"
                                        echo
                                        border "\e[36m\e[7m"
                                        guess_members $in1 $in2 $in3 # Executes
guess_members function passing the inputted arguements
                                else
                                        echo  -e  "\e[93m\e[1mUsage\e[0m:\e[91m
You  have  entered  a  members  code  more  than  once  please  try  again.
Example:\e[93m\e[5m JL AY FM\e[0m"
                                        re_enter choice "Please try again in"
#Calling re_enter function passing parameter choice
                                fi
                        else
                                echo;echo   -e   "\e[93m\e[1mUsage\e[0m:\e[91m
Please  enter  the  codes  that  are  shown  correctly.  Example\e[93m\e[5m  JL  AY
FM\e[0m"
                                re_enter choice "Please try again in" #Calling
re_enter function passing parameter choice
                        fi
                elif [[ $c -gt 3 ]]  # Checks if there are more than 3 arguements
                then
                        echo  -e  "\e[93m\e[1mUsage\e[0m:\e[91m  (More  than  3
arguments  detected)  Please  type  only  3  arguments  as  the  example  given:
Example\e[93m\e[5m JL AY FM\e[0m"
                        re_enter choice "Please try again in" #Calling re_enter
function passing parameter choice
                elif [[ $c -lt 3 ]]  # Checks if there are less than 3 arguements
                then
                        echo  -e  "\e[93m\e[1mUsage\e[0m:\e[91m  (Less  than  3
arguements  detected)  Please  type  only  3  arguements  as  the  example  given:
Example\e[93m\e[5m JL AY FM\e[0m"
                        re_enter choice "Please try again in" #Calling re_enter
function
                fi
                }
        checking $M1 $M2 $M3
        }
    choice #Calling the choice function
}



function re_enter() # Function to countdown for retrial Expectesd Use: (re_enter
function message)
{
    echo
    a=3
    until [ $a -eq 0 ]
    do
        echo -e "\e[94m$2\e[93m $a seconds"
        (( a-- ))
        sleep 0.8
```

```
    done
    $1
}

function guess_members() # Function for input display and vlidation of the
membes of the band
{
    echo
    header "\e[36m\e[7m" "\e[96m" "Choose One of the member out of three"
"small"
    echo -e "\n"
    arrName=()          # Creation of array for storing the display text
    function pushArray()      # Function for checking if the codes match the
cases to push the value in the array
    {
        case $1 in
            "JL")
                arrName+="John_Lennon(JL) "
            ;;
            "AY")
                arrName+="Agnus_Young(AY) "
            ;;
            "FM")
                arrName+="Freddie_Mercury(FM) "
            ;;
            "DH")
                arrName+="Debbie_Harry(DH) "
            ;;
            "KC")
                arrName+="Kurt_Cobain(KC) "
            ;;
        esac
    }
    pushArray $1 # Executing the pushArray function with 1st arguement
    pushArray $2 # Executing the pushArray function with 2nd arguement
    pushArray $3 # Executing the pushArray function with 3rd arguement
    PS3="Enter one of the number from above: "
    select var in $arrName   # select valuess in the array
    do
        case $var in
            "John_Lennon(JL)")
                FILENAME="./JL/JL.txt"
                if [[ ! -f $FILENAME ]] || [[ ! -r $FILENAME ]]
                then
                    echo
                    echo -e "\e[91mNo files found for John Lennon\e[0m"
                    re_enter menu "You will be redirected to the menu screen
in" # Executes re_enter function redirecting to menu
                else
                    cat $FILENAME
                fi
            ;;
            "Freddie_Mercury(FM)")
                FILENAME="./FM/FM.txt"
                if [[ ! -f $FILENAME ]] || [[ ! -r $FILENAME ]]
                then
```

```
                    echo
                    echo -e "\e[91mNo files found for Freddie Mercury\e[0m"
                    re_enter menu "You will be redirected to the menu screen
in" # Executes re_enter function redirecting to menu
                else
                    echo
                    border "\e[36m\e[7m"
                    echo
                    header    "\e[31m\e[7m"    "\e[91m"    "Freddie    Mercury"
"smscript";echo
                    echo -e "\e[93m(FM) \e[41m\e[5mFreddie Mercury\e[0m lead
performer of the band Queen"
                    echo
                    jp2a   --background=dark --size=50x30 --colors --red=0 --
blue=0.5 --green=0.5 ./FM/FM.jpg;echo
                    cat $FILENAME
                    echo -e "\n\n"
                    border "\e[31m\e[7m"
                    check menu quit "Do you want to restart the game? (yes/no):
" # Executes check function
                fi
            ;;
            "Debbie_Harry(DH)")
                FILENAME="./DH/DH.txt"
                if [[ ! -f $FILENAME ]] || [[ ! -r $FILENAME ]]
                then
                    echo
                    echo -e "\e[91mNo files found for Debbie Harry\e[0m"
                    re_enter menu "You will be redirected to the menu screen
in"
                else
                    echo
                    border "\e[36m\e[7m"
                    echo
                    header "\e[33m\e[7m" "\e[93m" "Debbie Harry" "small";echo
                    echo -e "\e[93m(DH) \e[31m\e[43m\e[5mDebbie Harry\e[0m lead
performer of the band Blondie"
                    echo
                    jp2a --size=50x30 --colors ./DH/DH.jpg;echo
                    cat $FILENAME
                    echo -e "\n\n"
                    border "\e[33m\e[7m"
                    check menu quit "Do you want to restart the game? (yes/no):
" # Executes check function
                fi
            ;;
            "Kurt_Cobain(KC)")
                FILENAME="./KC/KC.txt"
                if [[ ! -f $FILENAME ]] || [[ ! -r $FILENAME ]]
                then
                    echo
                    echo -e "\e[91mNo files found for Kurt Cobain\e[0m"
                    re_enter menu "You will be redirected to the menu screen
in" # Executes re_enter function redirecting to menu
                else
                    cat $FILENAME
```

```
                fi
            ;;
            "Agnus_Young(AY)")
                FILENAME="./AY/AY.txt"
                if [[ ! -f $FILENAME ]] || [[ ! -r $FILENAME ]]
                then
                    echo
                    echo -e "\e[91mNo files found for Agnus Young\e[0m"
                    re_enter menu "You will be redirected to the menu screen
in"
                else
                    echo
                    border "\e[36m\e[7m"
                    echo
                    header "\e[36m\e[7m" "\e[92m" "Agnus Young" "smslant";echo
                    echo -e "\e[92m(DH) \e[91m\e[42m\e[5mAgnus Young\e[0m Lead
Guitarist of the band AC/DC"
                    echo
                    jp2a --size=50x30 --colors  ./AY/AY.jpg;echo
                    cat $FILENAME
                    echo -e "\n\n"
                    border "\e[36m\e[7m"
                    check menu quit "Do you want to restart the game? (yes/no):
" # Executes check function
                fi
            ;;
            *)
                echo  -e  "\e[91m\e[1mUsage\e[0m:\e[91m  Please  enter  number
correctly \e[93m\e[5m(1-3)\e[0m"
            ;;
        esac
    done
}

check_arguement "$@" #Calling the check_arguement function
```

Testing

Test 1: Running the program without username

Test 1	
Objective	Running the program without username
Input	The following argument was passed to the program ➔ ./20049037cw2pii.sh 20049037
Expected Output	➔ A validation message stating: (Argument insufficient) Usage: Please enter the command line argument as follows: (./20049037cw2pii FirstName IdNumber) would be shown
Actual Output	➔ A validation message stating: (Argument insufficient) Usage: Please enter the command line argument as follows: (./20049037cw2pii FirstName IdNumber) was shown
Test Result	The Test was successful.

Table 1 Testing No. 1

Screenshots:

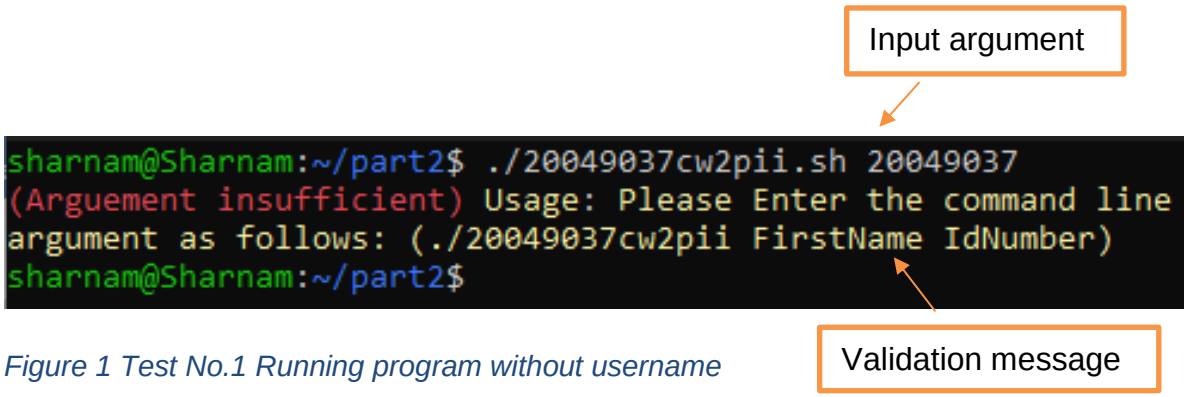

Figure 1 Test No.1 Running program without username

Test 2: Running the program with username and password

Test 2	
Objective	Running the program with both username and id
Input	The following argument was passed to the program ➔ ./20049037cw2pii.sh Sharnam 20049037
Expected Output	➔ The user would be taken to the program where they are supposed to fill in the password for proceeding to further stage of the program.
Actual Output	➔ The user was taken to the program where they are supposed to fill in the password for proceeding to further stage of the program.
Test Result	The Test was successful.

Table 2 Testing No. 2

Screenshots:

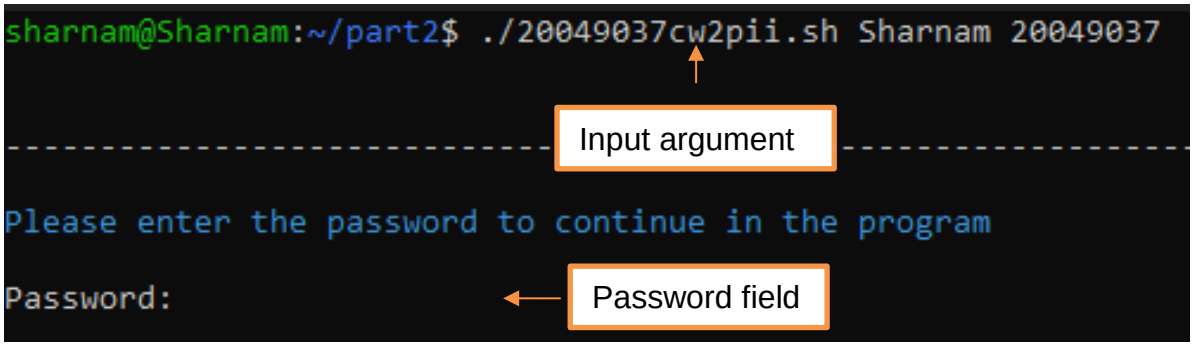

Figure 2 Test No.2 Running program with username and id

Test 3: Typing the password incorrectly 3 times

Test 3	
Objective	Typing the password incorrectly 3 times
Input	An Incorrect password was typed in 3 times ➔ Password: incorrect ➔ Password: incorrect ➔ Password: incorrect
Expected Output	The message would be shown when the user incorrectly types in the password every time stating the number of tries left for the user: ➔ Wrong password you have 2 tries left ➔ Wrong password you have 1 try left ➔ Wrong password you have 0 tries left ➔ Program would be terminated by stating that “You have entered the password wrong 3 times, the program will be terminated”
Actual Output	Message was shown when the user incorrectly typed in the password every time stating the number of tries left for the user: ➔ Wrong password you have 2 tries left ➔ Wrong password you have 1 tries left ➔ Wrong password you have 0 tries left ➔ Program was terminated by stating that “You have entered the password wrong 3 times, the program will be terminated”
Test Result	The Test was successful.

Table 3 Testing No. 3

Screenshots:

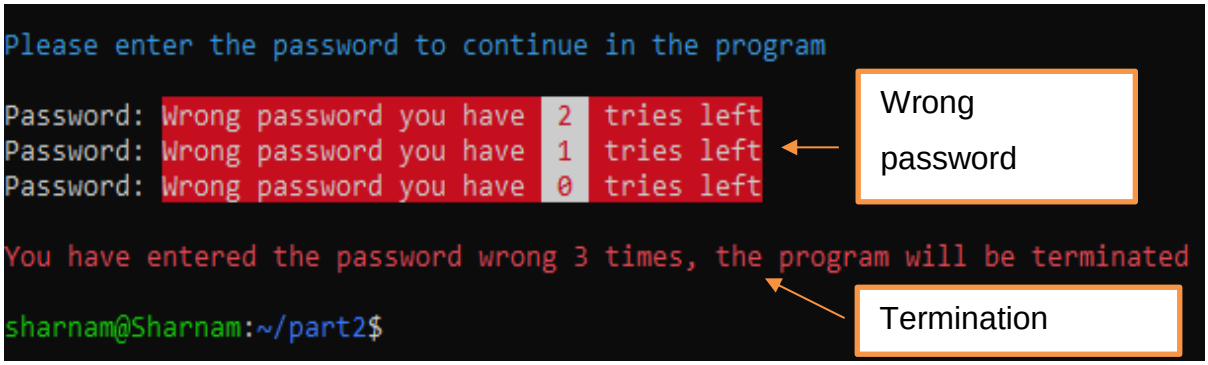

Figure 3 Test No. 3 Typing password incorrectly 3 times

Test 4: Running the program with correct password

Test 4	
Objective	Running the program with correct password
Input	The correct password was entered: ➔ Password: SDhakz
Expected Output	➔ "Password accepted" would be shown ➔ The user would be taken to the program where the program would welcome the user by displaying their username, user id, executed date and time, and the menu screen of the game.
Actual Output	➔ "Password accepted" was shown ➔ The user was taken to the program where the program welcomed the user by displaying their username, user id, executed date and time, and the menu screen of the game.
Test Result	The Test was successful.

Table 4 Testing No. 4

Screenshots:

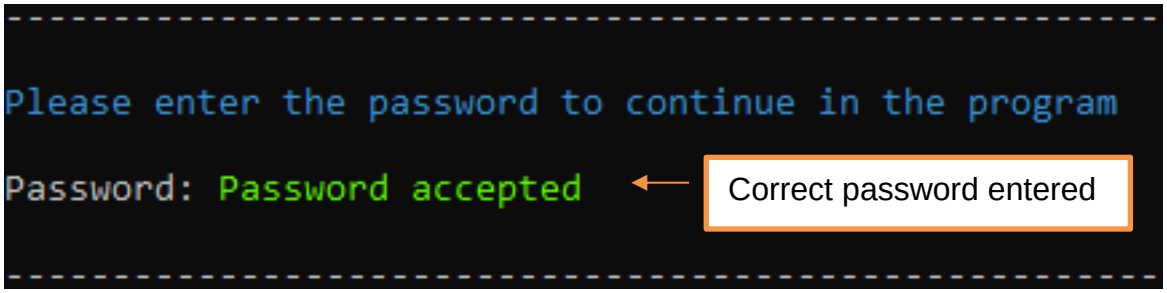

Figure 4 Test No. 4 Inputting correct password

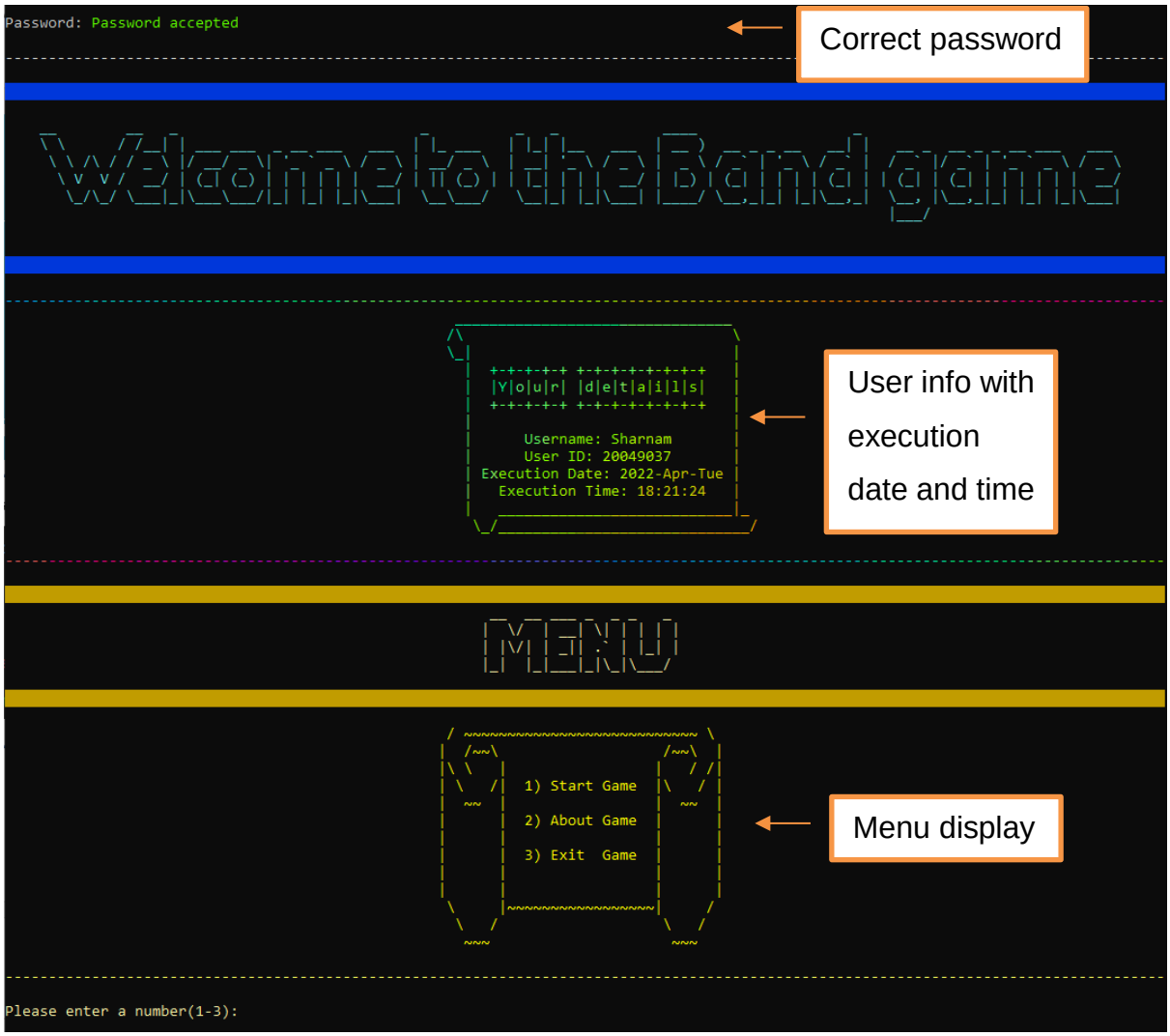

Figure 5 Test No. 4 Inputting correct password to run the game

Test 5: Typing random input instead of a band code

Test 5	
Objective	Typing random input instead of a band code shown on the screen
Input	random input was given: ➔ Enter the band code: ad
Expected Output	➔ The user would be shown a message stating “Usage: Please enter one of the codes as shown (AD/BE/BLO/QUE/NIR)” ➔ The user would be able to type in the band code again
Actual Output	➔ The user was shown a message stating “Usage: Please enter one of the codes as shown (AD/BE/BLO/QUE/NIR)” ➔ The user was able to type in the band code again
Test Result	The Test was successful.

Table 5 Testing No. 5

Screenshots:

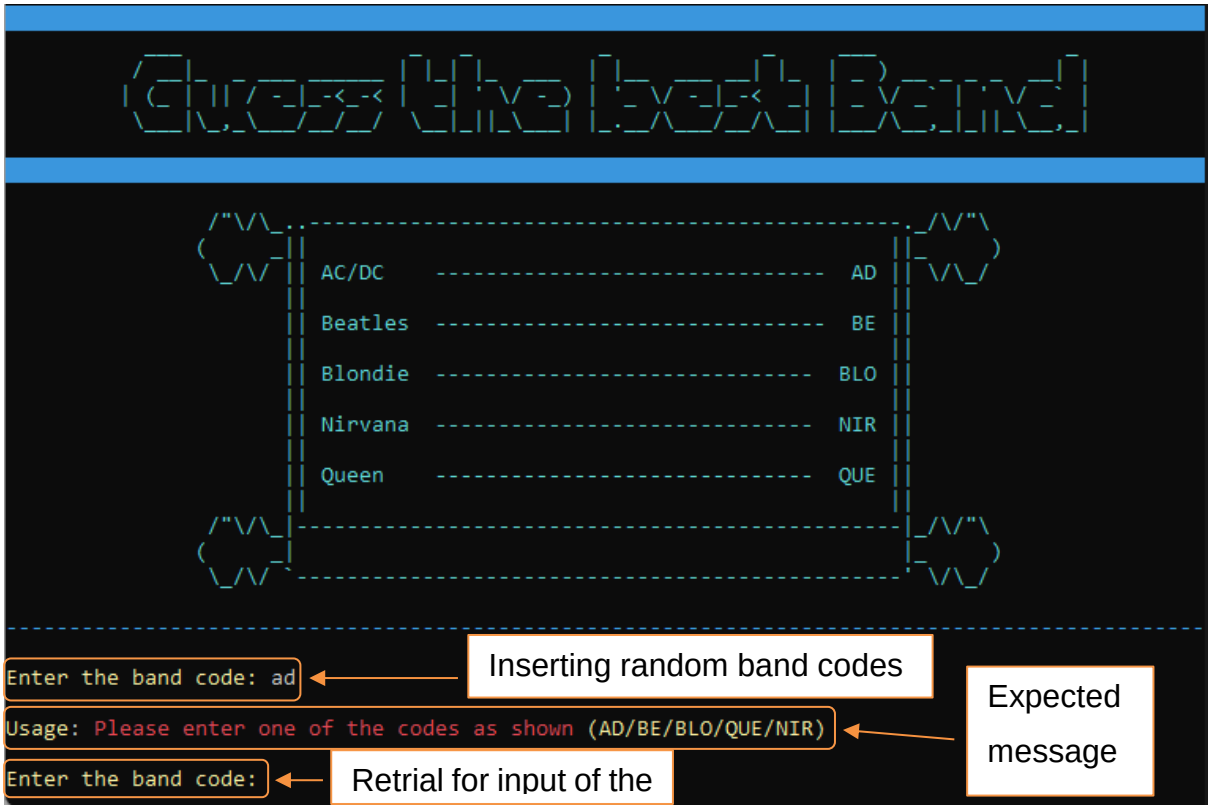

Figure 6 Test No.5 Putting random band codes

Test 6: Inputting the incorrect band code

Test 6	
Objective	Typing incorrect band code
Input	The incorrect band code input was given: ➔ Enter the band code: BE
Expected Output	➔ The user would be given a display with a wrong answer banner stating the band was the incorrect option ➔ The user would be given a choice to retry again.
Actual Output	➔ The user was given a display with a wrong answer banner stating the band was the incorrect option ➔ The user was given a choice to retry again.
Test Result	The Test was successful.

Table 6 Testing No. 6

Screenshots:

Figure 7 Test No. 6 Inputting the wrong band code

Figure 8 Test No. 6 Inputting incorrect band code full

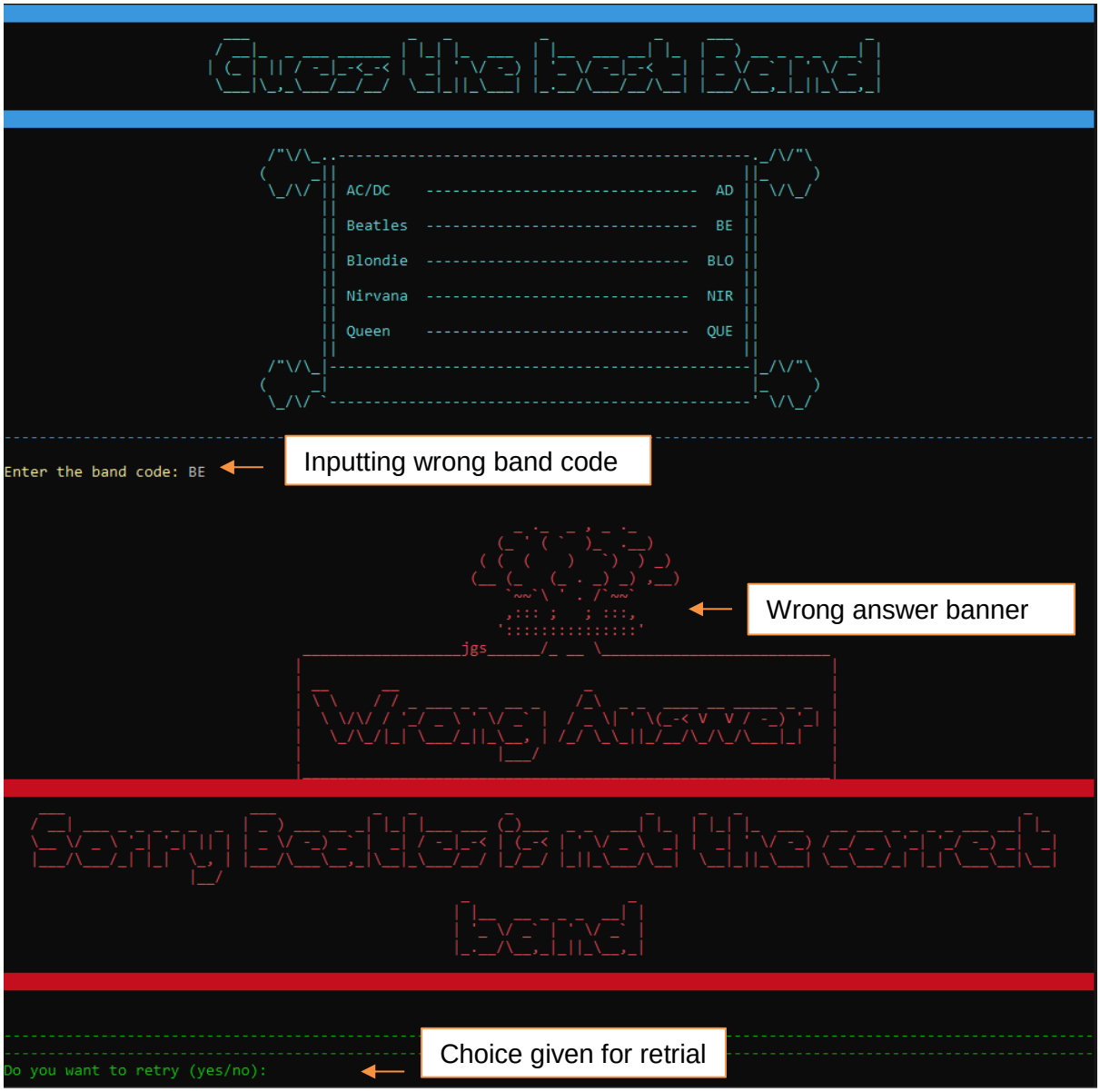

Figure 9 Test No. 6 incorrect band code full test view

Test 7: Inputting correct Band code

Test 7	
Objective	Typing correct band code
Input	Correct band code input was given: ➔ Enter the band code: QUE
Expected Output	➔ The user would be given a display that states the correct answer and opens the text file to display the information about the band
Actual Output	➔ The was given a display which stated correct answer and opened the text file to display the information about the band
Test Result	The Test was successful.

Table 7 Testing No. 7

Screenshots:

Figure 10 Test no. 7 Inputting correct band code full result

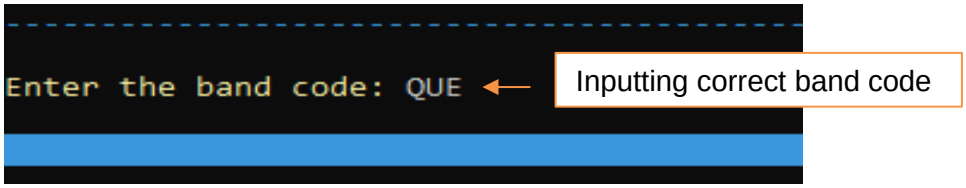

Figure 11 Test No.7 Inputting correct band code

Test 8: Picking 4 member codes

Test 8	
Objective	Choosing 4 member codes
Input	The following member codes were inputted: ➔ Enter the options from above: JL AY FM DH
Expected Output	➔ The user would be given a message stating “Usage: (More than 3 arguments detected) Please type only 3 arguments like the example given: Example JL AY FM” ➔ The user would get another chance to enter it correctly
Actual Output	➔ The user was given a message stating “Usage: (More than 3 arguments detected) Please type only 3 arguments like the example given: Example JL AY FM” ➔ The user got another chance to enter it correctly
Test Result	The Test was successful.

Table 8 Testing No. 8

Screenshots:

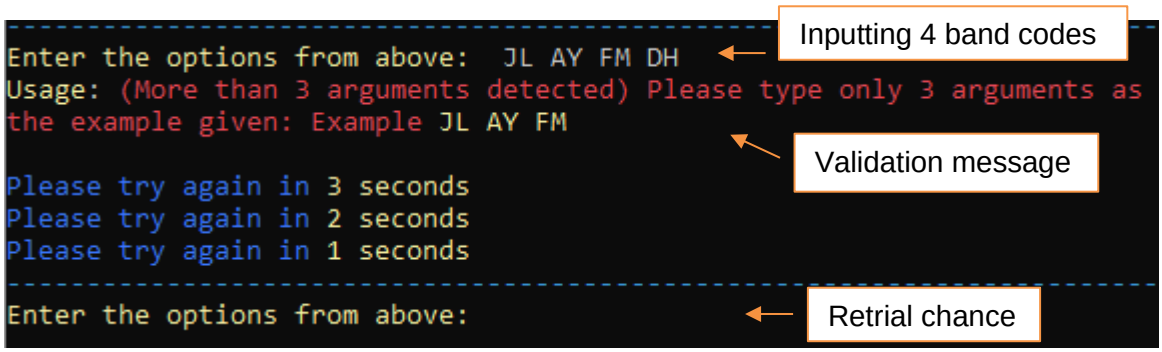

Figure 12 Test No. 8 Inputting 4 band codes

Figure 13 Test No. 8 Inputting 4 band member codes full

Test 9: Picking same member name

Test 9	
Objective	Choosing the same member codes
Input	The following member codes were inputted: ➔ Enter the options from above: JL FM FM
Expected Output	➔ The user would be given a message stating “Usage: You have entered a member’s code more than once please try again. Example: JL AY FM” ➔ The user would get another chance to enter it correctly
Actual Output	➔ The user was given a message stating “Usage: You have entered a member’s code more than once please try again. Example: JL AY FM” ➔ The user got another chance to enter it correctly
Test Result	The Test was successful.

Table 9 Testing No.9

Screenshots:

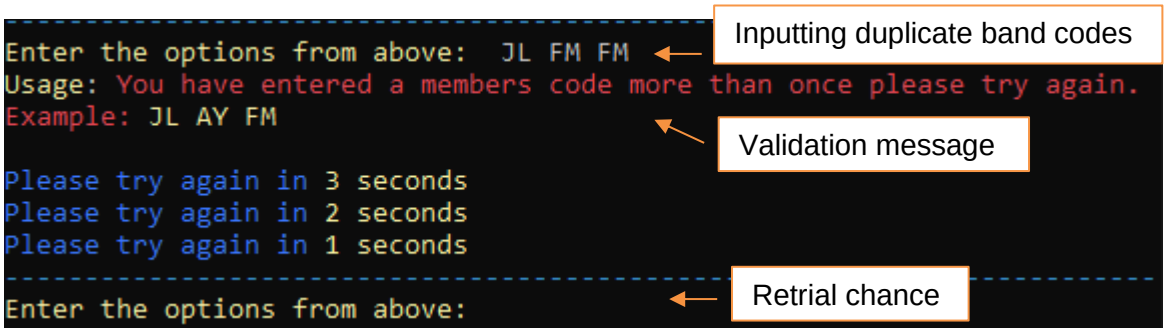

Figure 14 Test No.9 Inputting duplicate band codes

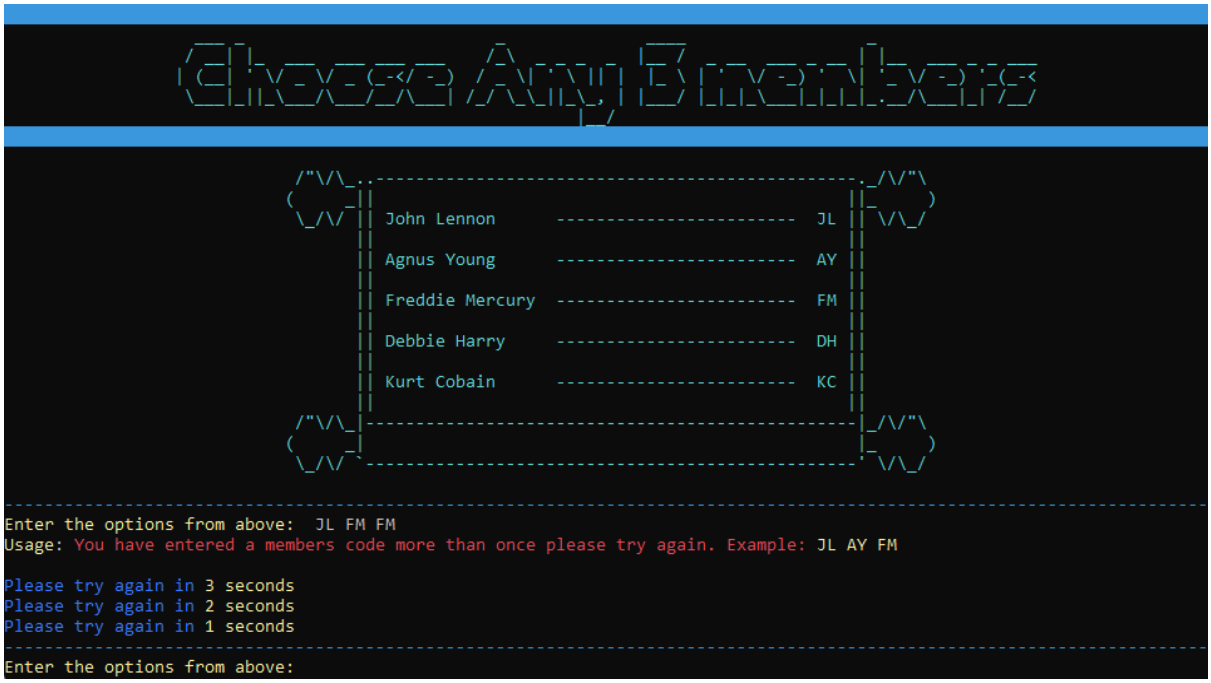

Figure 15 Test No. 9 Inputting duplicate band member codes full

Test 10: Wrong user id was inputted

Test 10	
Objective	Inputting the wrong user-id
Input	The following wrong id was inputted: ➔ Enter one of the numbers from above: 4
Expected Output	➔ The user would be given a message stating “Usage: Please enter the number correctly (1-3)” ➔ The user would get another chance to enter it correctly
Actual Output	➔ The user was given a message stating “Usage: Please enter the number correctly (1-3)” ➔ The user gets another chance to enter it correctly
Test Result	The Test was successful.

Table 10 Testing No. 10

Screenshots:

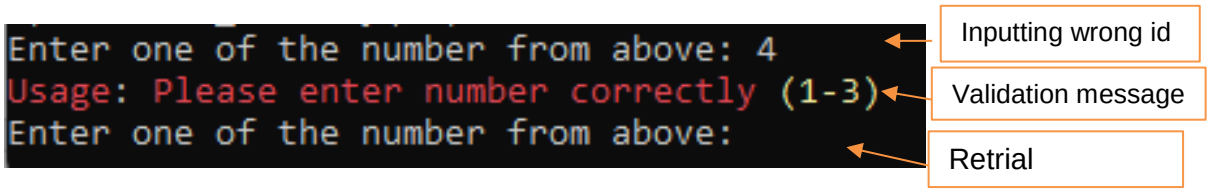

Figure 16 Test No. 10 Inputting wrong id

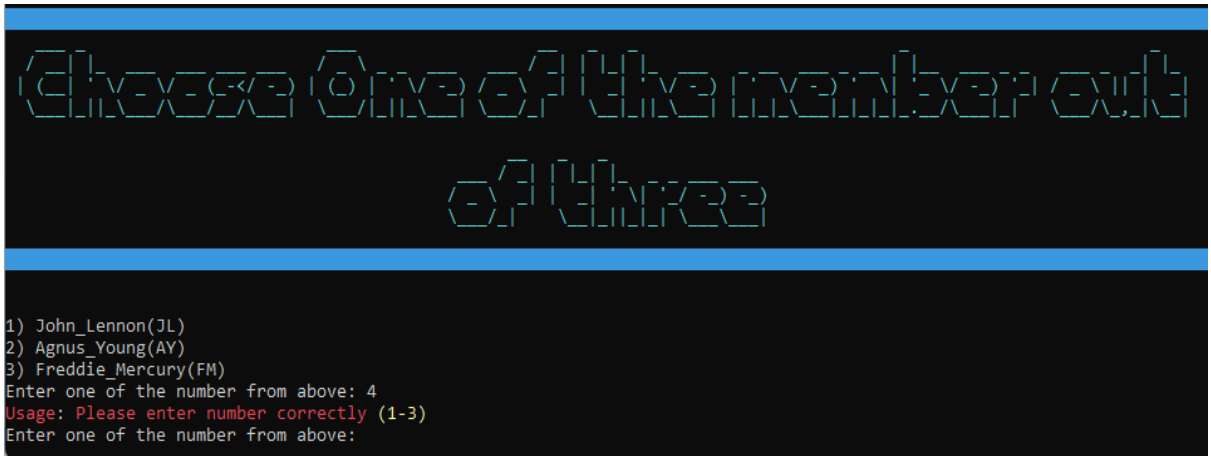

Figure 17 Test No. 10 Inputting wrong id

Test 11: Right user id was inputted

Test 11	
Objective	Inputting right user-id
Input	The following right id was inputted: ➔ Enter one of the numbers from above: 3
Expected Output	➔ The user would be displayed with the chosen member's information ➔ The user would get an option to restart the game or terminate it
Actual Output	➔ The user was not displayed with the chosen member's information ➔ The user did not get an option to restart the game or terminate it
Test Result	The Test was unsuccessful.

Table 11 Testing No. 11

Screenshots:

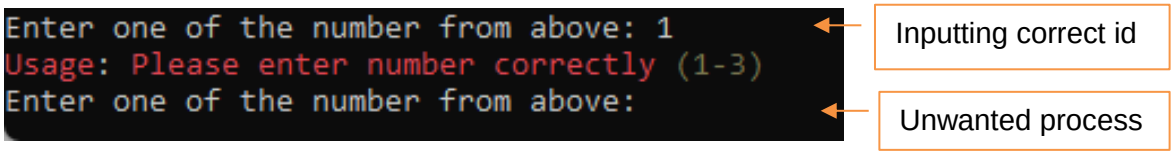

Figure 18 Test No. 11 Inputting right id but error occurs

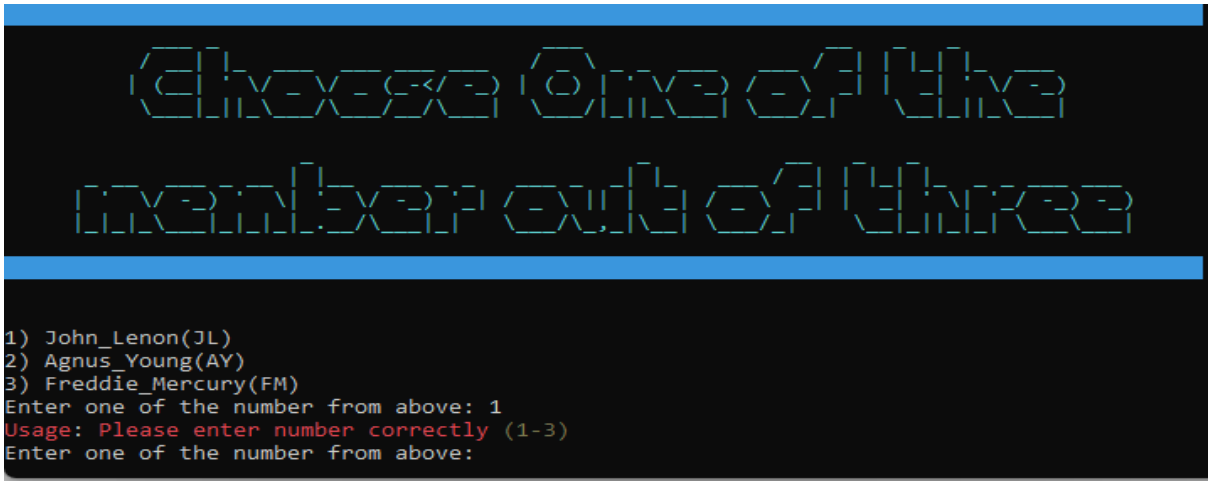

Figure 19 Test no.11 Unsuccessful test when correct user id was given (error) full

Test 11: Right user id was inputted (Debugged)

Test 11	
Objective	Inputting right user-id
Input	The following right id was inputted: ➔ Enter one of the numbers from above: 3
Expected Output	➔ The user would be displayed the chosen member's information ➔ The user would get an option to restart the game or terminate it
Actual Output	➔ The user was displayed with the chosen member's information ➔ The user gets an option to restart the game or terminate it
Test Result	The Test was successful.

Table 12 Testing No. 11 Debugged

Screenshots:

```
1) John_Lennon(JL)
2) Agnus_Young(AY)
3) Freddie_Mercury(FM)
Enter one of the number from above: 3
```

Figure 20 Test No. 11 Inputting id which has existing file

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Do you want to restart the game? (yes/no):
```

Figure 21 Test No. 11 user getting option to restart the game

Figure 22 Test No. 11 Full outcome when correct id is inputted

Test 12: No External File of member (except 3 profile players that you have made)

Test 12	
Objective	Putting code of the member with no external file
Input	The following id was inputted: ➔ Enter one of the numbers from above: 1
Expected Output	➔ The user would be shown a message stating “No files found for John Lennon” ➔ The user would be redirected to the menu
Actual Output	➔ The user was shown a message stating “No files found for John Lennon” ➔ The user was redirected to the menu
Test Result	The Test was successful.

Table 13 Testing No. 12

Screenshots:

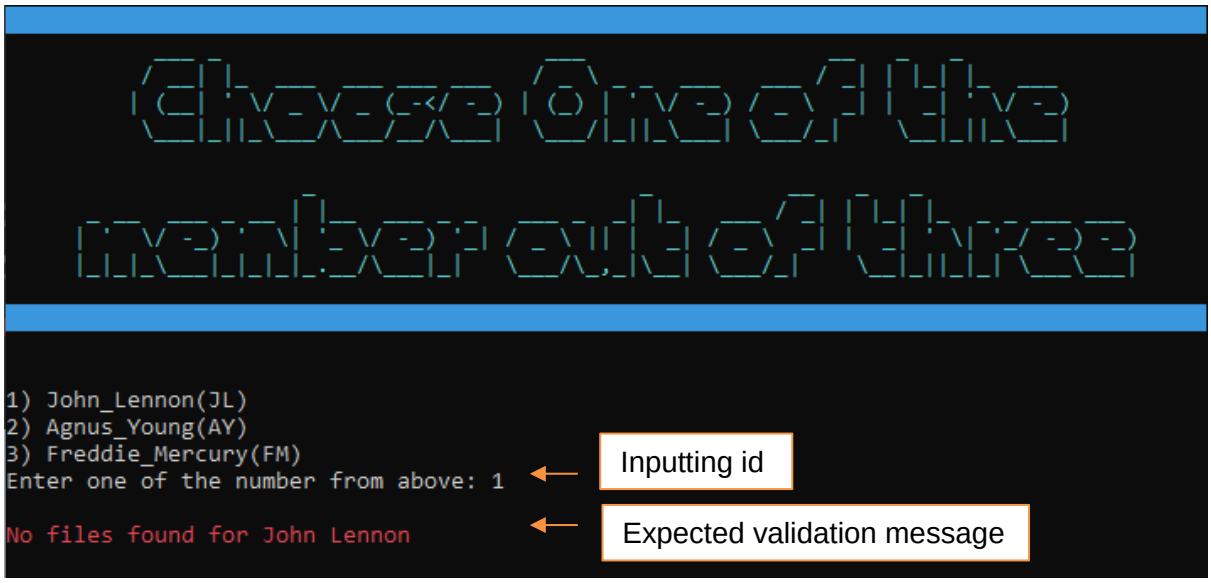

Figure 23 Test 12 No files found for the chosen id

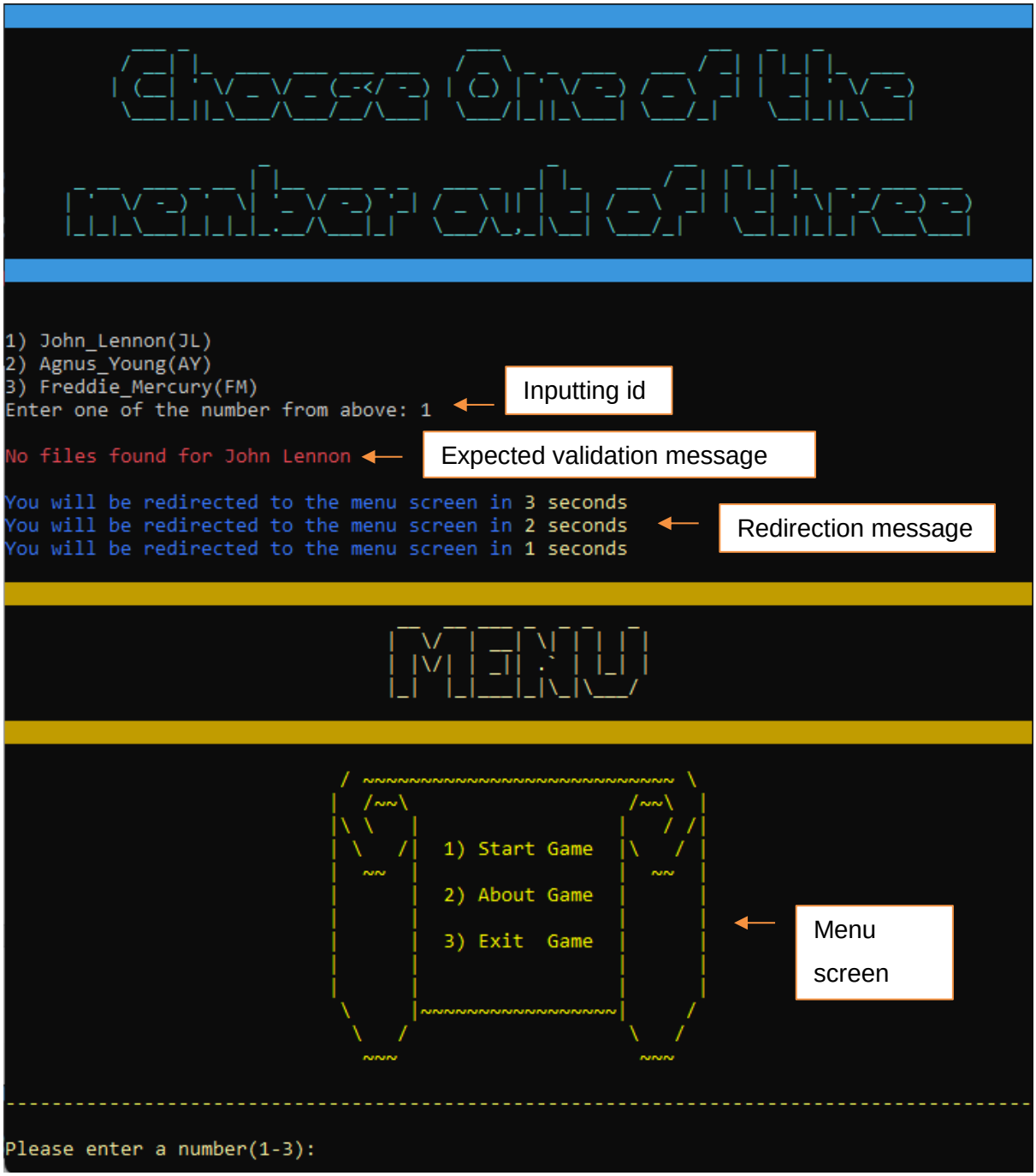

Figure 24 Test No. 12 Redirection to menu screen after no files found

Test 13: Repeating the game after saying yes at the end of the game

Test 13	
Objective	To see if the game restarts after pressing yes at the ending phase
Input	The following was inputted: ➔ Do you want to restart the game? (yes/no): yes
Expected Output	➔ The user would be taken to the menu screen
Actual Output	➔ The user was taken to the menu
Test Result	The Test was successful.

Table 14 Testing No. 13

Screenshots:

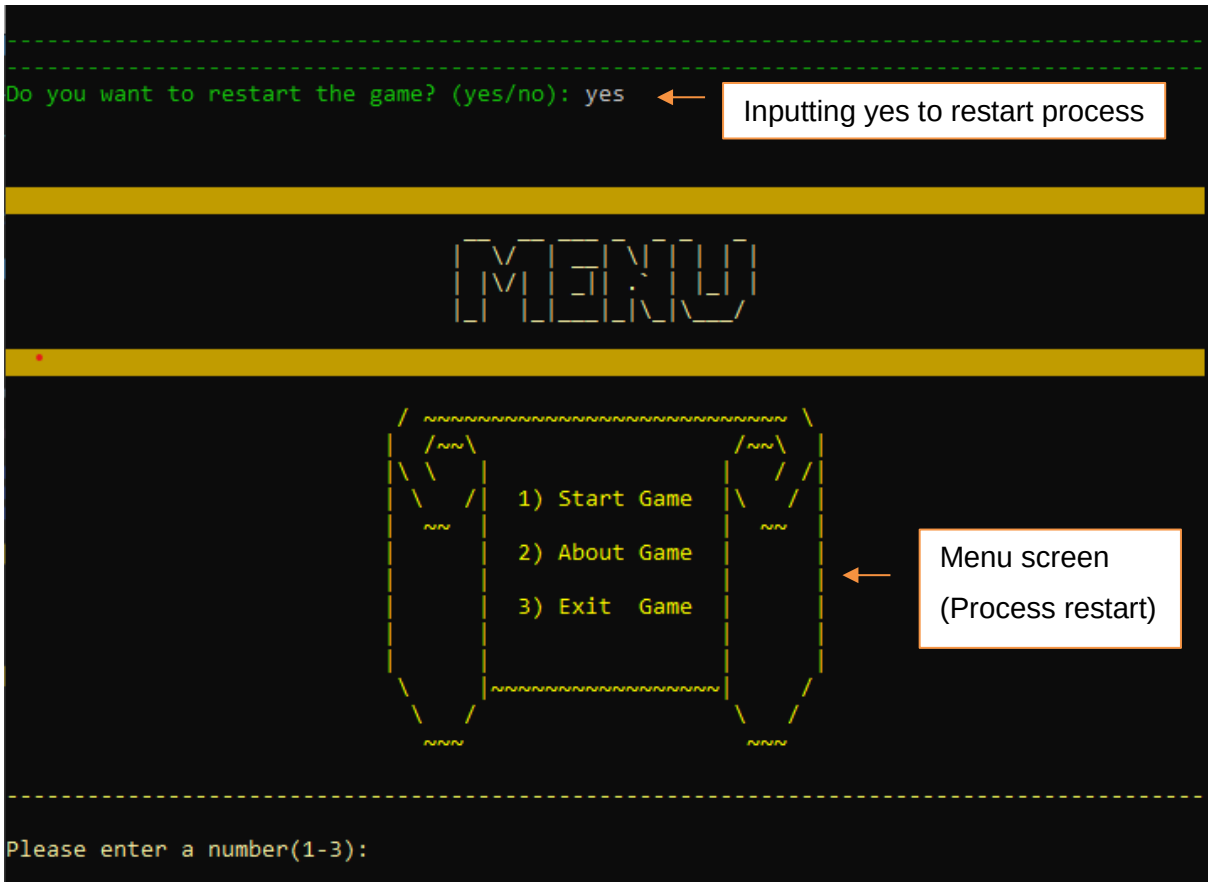

Figure 25 User taken to menu to restart the game when pressed yes

Test 14: Exiting the game after saying no at the end of the game

Test 14	
Objective	To see if the game terminates after pressing no at the ending phase
Input	The following was inputted: ➔ Do you want to restart the game? (yes/no): no
Expected Output	➔ The program would terminate by thanking the user
Actual Output	➔ The program was terminate by thanking the user
Test Result	The Test was successful.

Table 15 Testing No.14

Screenshots:

Figure 26 Test No. 14 terminating the game when inputting no

Contents of three files: (TEXTS)

FM

Freddie Mercury, the frontman for Queen, was a prolific writer, an introverted empath, and a legendary musician. His band is one of the most successful in history, with more than 300 million albums sold worldwide.

Queen's two-night stand at London's Wembley Stadium in1986 is still widely regarded as one of the most memorable live rock performances ever.

AY

Angus McKinnon Young (born March 31, 1955) is an Australian musician and best known as the lead guitarist and songwriter co-founder of the Australian hard rock band AC / DC.

The only permanent former member. He is known for his energetic performances, uniform-inspired stage outfits, and his own view of Chuck Berry's runway. Young and other AC / DC members were inducted into the Rock and Roll Hall of Fame in 2003

DH

Deborah "Debbie" Ann Harry (born July 1, 1945) is an American singer-songwriter and actress best known as the lead singer of the band Blondie. Her band's recordings topped the US and UK charts between 1979 and 2017.

In 1981, Harry released his debut solo album, KooKoo, and began his acting career during a break from Blondie, starring in the Neo-Noir Union City (1980) and David Cronenberg's body horror film Videodrome (1983).

In 1986 she released her second solo album, Rockbird, and she starred in John Waters' cult dance film Hairspray (1988).

Conclusion

The program of task A is built and tested successfully. The bash shell which is a compatible shell that includes several useful features from the C shell and Korn shell is also used for the completion of the coursework. Microsoft word and snippet tools were used for the creation of the proper documentation of the coursework itself.

Moreover, various concepts and the utilization of select statements, loops, functions, if-else statements, switch-case statements and many more were grasped. Validation techniques, diagnosing input errors, and providing appropriate messages and testing's were also learned during the progress of this coursework. The coursework also helped to grasp the concept of functions passing arguments in functions.

There were many problems during the completion of this task like errors due to mistakes in the casing of the variables, mistakes in closing the loops, and calling the function in the wrong area but all of the errors were solved easily as debugging the program wasn't hard due to the experience in debugging. There were also challenges like trying to align the program at the centre which was solved by researching stack overflow and making a function for it. The module teachers also helped to clarify the functionality of the program which was a lot of help in making the program smoothly as the blueprints were laid.

Finally, this task helped to point out important notes about using a Linux shell-like knowing about commands, files and directory being case sensitive, knowing that the file extension doesn't matter as it is determined automatically and nearly every command supports **--help** argument which will guide the user how the command is used. It also helped to hone bash scripting skills.

Task B: Process Management

Introduction

Rapid improvements in microcomputer technology have resulted in devices that outperform mainframe computers in terms of operation and pricing. There has been a strong tendency for large-scale computer systems to shift from mainframes to distributed systems made up of smaller computers. A small-computer distributed system, executes processes on numerous computers, resulting in processes being managed independently on individual computers, as opposed to a mainframe system, which performs many processes on one computer and manages all of them collectively. This results in user to keep the network in mind when managing one's own processes (DBNSTJ, 2022).

So to counter this situation Network Operating System was proposed in 1986 to manage network-wide processes in a distributed processing system. A network OS is a computer OS that allows multiple independent computers to connect and communicate through a network. This OS would allow users to execute distributed processing without having to worry about the particular locations of their running processes (DBNSTJ, 2022).

The execution of a program that accomplishes the actions defined in that program is called a process. Process management is the management of numerous tasks such as process creation, scheduling, termination, and a deadlock. A process is an 'active' entity, instead of a program, which is considered a 'passive' entity. A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created) (GeeksforGeeks, 2015).

Aims and Objectives

Aims

- To understand process management in the operating system
- To grasp the concept of process architecture
- To get comfortable with the backgrounds of process management such as the process control blocks, process states, process hierarchy
- To gain an idea about how the process is implemented

Objectives

- Research various sources on the web about the network operating system and the process management
- Research on various books and e-books relating to process management
- Research on journals and articles

Background:

Process Architecture

Process architecture is generally divided into four sections for increasing the efficiency:

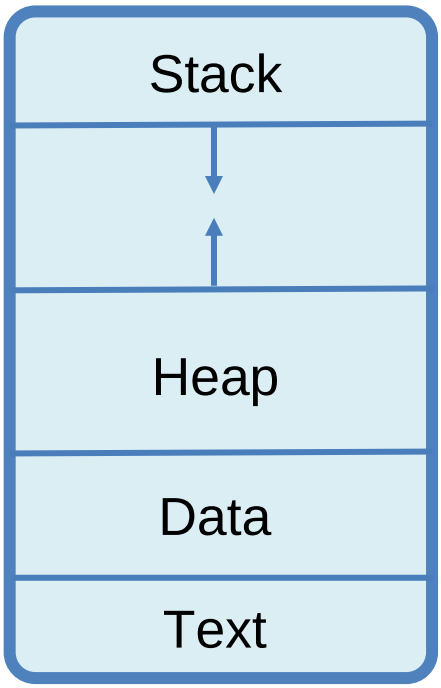

Figure 28 Process architecture

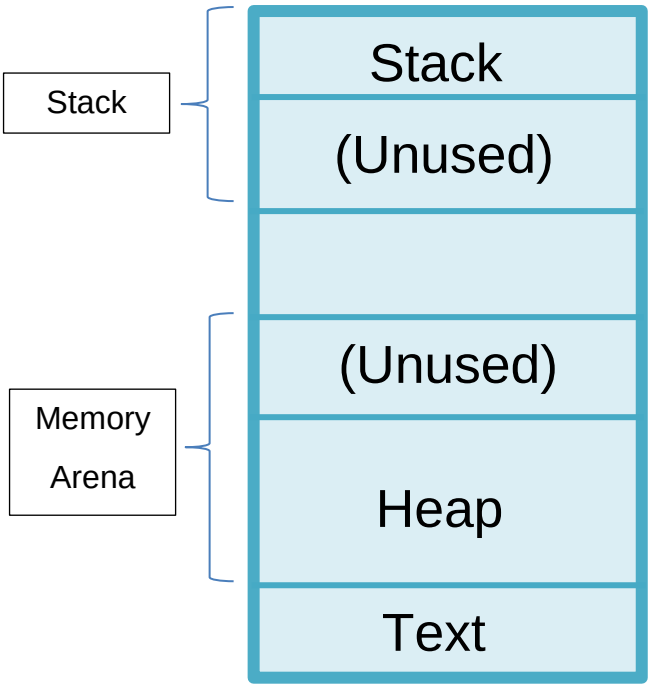

Figure 27 Memory associated with instance of a program

Memory Arena

Memory Arena (also known as break space)--the area where dynamic runtime memory is stored. The memory arena consists of the heap and unused memory.

Stack section

The stack section consists of the temporary data like returns addresses, function parameters and local variables. As seen in the diagram, the stack and heap grow in opposing directions, which is advantageous because if they both grow in the same direction, they will overlap, so growing in opposite directions is preferable. A unique memory arena and stack exists for each instance of the program (Sorfa, 2001).

For example:

```
Function add()
{
        return ( a + 1 )
}
```

The add function when called will be stored in the stack section and as soon as the function returns the value the stack section that contains that function will be deleted.

Heap Section

The heap section is used to dynamically provide memory in the time when memory is required by the program during its run time. The heap is where all user-allocated memory is located. The heap grows up from a lower memory address to a higher memory address. User-allocated memory is located in the heap in the memory arena (Sorfa, 2001).

The malloc(), free() and calloc() functions are utilized for the management of memory in heap section.

Data Section

The Data section contains the static local variables and the global variables.

Text Section

This section contains executable instructions, constants, and macros. It is a read-only area that can be shared with other processes. Several instances of the same program is able to share this area.

Process Control Blocks

A Process Control Block is a data structure in the kernel of an operating system that contains the information required to manage a specific process. The PCB is "the manifestation of a process in an OS". PCB contains the data that specifies the existence of a particular process and the information necessary to permit the process to make forward progress (Thomas Sterling, 2018).

Because multi-programming is supported by the OS, it must keep track of all processes. The PCB is used to track the process's execution status for this task. Each memory block holds information on the current state of the process, the program number, the stack pointer, the status of open files, scheduling algorithms, and so on. A PCB contains all the information about the process like registers, quantum, priority etc. (GeeksforGeeks, 2017).

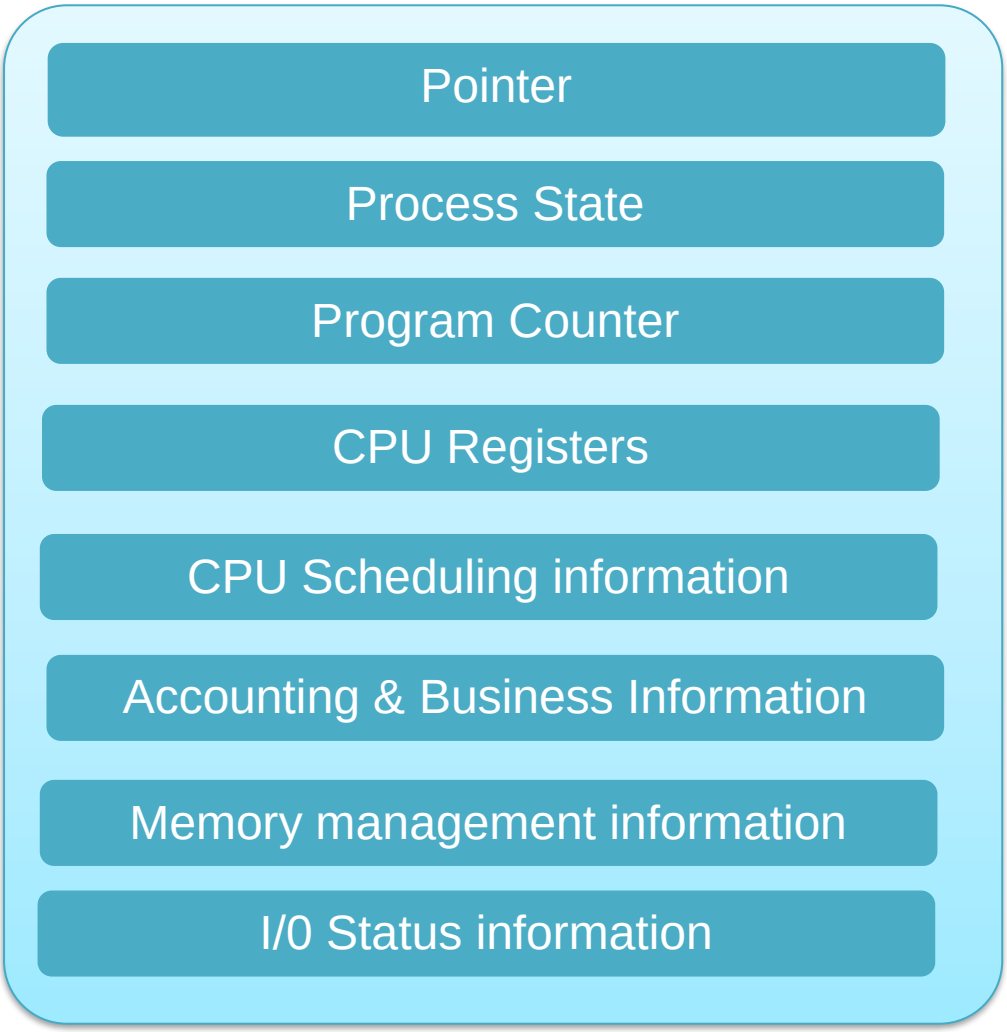

Figure 29 Process Control Block

PCB varies from OS to OS. However there are some basic parameters common to PCBs of all OSs like:

Pointer

It is a stack pointer that must be saved when the process is transitioned from one state to another to keep the process' present location.

Process State

It stores the respective state of the program i.e. ready, new running, waiting, or terminated.

Program Counter

It is used to store the counter which contains the address of the next instruction that needs to be executed in the process.

CPU Registers

This component has the index, accumulators, base, stack pointers, general-purpose registers, and information on condition code.

CPU Scheduling Information

The CPU scheduling information on the PCB includes process priority, links to scheduling queues, and so on. Any other scheduling criteria may also be included.

Accounting and Business Information

The PCB accounting information includes time limitations, account numbers, CPU usage, process numbers, and so on.

Memory management information

Depending on the memory system, the memory management information includes page tables or segment tables. It also contains the values of the base and limit registers, among other things.

I/O status information

This component contains the information of the list of I/O devices which are used by the process, the list of open files that are allocated to the process, etc.

Process States

When a process is executed then it changes the state and the state of a program is determined by the current activity of the process (webeduclick, 2019). Each process in an OS can be in one of the following states:

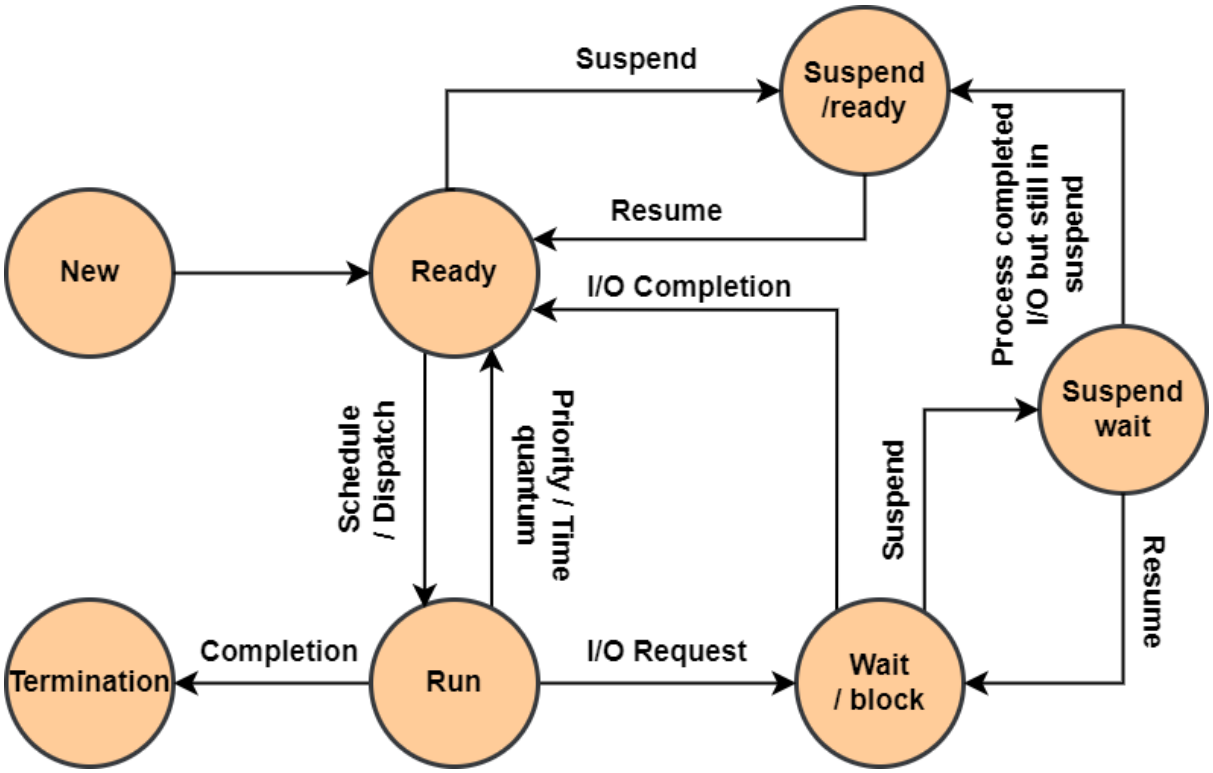

Figure 30 Process state diagram

New

A new process is a program that will be loaded into the main memory by the OS. It is the process that is being created.

Ready

Whenever a process is created, it directly enters the ready state, in which, it waits for the CPU to be assigned (javatpoint, 2021). Then from the secondary memory, the OS is responsible for selecting new processes and placing them into the main memory.

Generally, ready state processes are processes that are ready for execution and sit in the main memory. Many processes may be active in the ready state.

Running

In the running state one of the processes will be chosen from the ready state by the OS depending on the scheduling method. So, if we have an n number of processors in the system we will have n number of processes running simultaneously.

Block or wait

Depending on the scheduling method or the process' intrinsic behaviour, a process can move from the Running state to the Block or Wait state.

In the waiting state the process is waiting for CPU time and other resources to be allocated for execution. In the blocked stage the process is waiting for the completion of the I/O operations.

Completion or Termination

When the process completes its execution then the process comes to a termination state where all the context of the process (PCB) will also be deleted and the process will be terminated by the operating system (javatpoint, 2021).

Suspend Ready

When the primary memory is full and if a higher priority process is scheduled for execution the OS is responsible for freeing up space in the main memory by moving the lower priority processes in the secondary memory known as suspend ready state of a process. Suspend ready processes are kept in the secondary memory until the main memory is free of space.

Suspend Wait

It is preferable to remove the blocked process that is awaiting resources in the main memory and place it into the secondary memory for waiting rather than removing the process from the ready queue itself. Once the main memory frees up then these processes can continue their execution.

New -> Ready: The OS generates a process, prepares it for execution, and then moves it to "Ready Queue."

Ready -> Running: The OS selects one of the jobs from the ready queue and moves the process from ready to running state (webeduclick, 2019).

Running -> Ready: The OS switches the running process to the ready state when the processor's time slot expires.

Running -> Terminated: After the completion of the execution of the process the OS terminates that process from the running state

Running -> Waiting: The process is kept to the waiting state if the process needs an event to occur or for the completion of the I/O operation.

Waiting -> Ready: When the event for which the process has been waiting occurs, it is changed from the blocked state to the ready state.

Process Hierarchies

It is a known fact that in a computer system, we are able to run many processes at a time. Some processes even create other processes during their execution.

When a process generates a child process, the parent and child processes tend to associate with each other in specific ways. If necessary, the child process can also create other processes. This parent-child organization of processes is referred to as Process Hierarchy.

A forks B and C

B forks D, E and F

C fork G

D forks H

H forks I

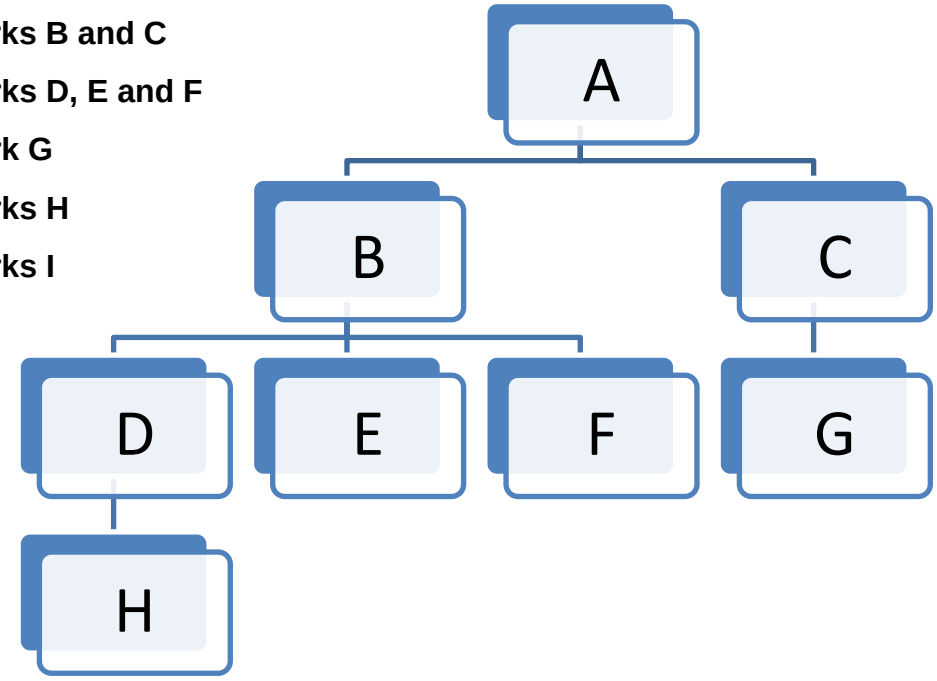

Table 16 Process Hierarchies example

New and modern OSs permit the user to create and destroy processes. The fork system call, which creates a new process, and the exit system call, which terminates the current process, are used in UNIX to do this. The parent and the child keep running and can fork off other processes. A process can decide to wait for children to finish before terminating. For example, If C used the wait() system function it would be blocked until G finishes.

Note that unlike plants and animals that use sexual reproduction, a process has only one parent (but zero, one, two, or more children). So a process is more like a hydra than like, say, a cow (Andrew S. Tanenbaum, 2014).

Implementation of Process

Process Table and PCB are used to implement the Process Model and keep track of all process information. A process table is used to implement the process model which is maintained by the OS with only one entry per process. The entry contains very important information about the process' state, including its program counter, memory allocation, stack pointer, the status of its open files, accounting and scheduling information, and everything about the process that needs to be saved when the process is switched from running state to ready state or blocked state so that it can be restarted later as if it had never been stopped (Andrew S. Tanenbaum, 2014).

Process Management	Memory Management	File Management
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User Identity
Process state		Group Identity
Priority		
Scheduling parameters		
Process Identity		
Parent process		
Process group		
Signals		
Time when process started		
Central Processing Unit time used		
Central Processing Unit time of Children		
Time of next alarm		

Table 17 Process table

From the above table the 1st, 2nd, and 3rd columns relate to process management, memory management and file management respectively. It should be emphasized that the exact fields that the process table contains are largely system

dependent, however, this diagram provides an overview of the types of information required.

Context saving is the process of saving the status of a running process in its PCB. After a process's context has been saved, the relevant event handling method is called. For example, if an I/O device is in the wait state, it is sent to the blocked processes queue. After that, because the current process has been halted, another process must be despatched to the CPU. As a result, a task from the ready queue is scheduled using the scheduler. After selecting the process from the ready queue, its PCB is loaded and dispatched to the CPU for execution (tutorialsSpace.com, 2022).

The processes are implemented in this fashion by storing their context in PCBs and allowing for state modification.

Conclusion

From task B portion of the coursework we learn about the important concepts that lie in the process management of an operating system. The report helped in clarifying what a process is, the architecture of the process, the various states that a process can undergo during its execution, process control blocks or the data structures of the processes that contains information about the processes necessary to permit the further progress of the processes that lie within the Kernel, the process Hierarchies and the implementation of the process themselves.

Furthermore, additional research was also done about the scheduling methods and the algorithm on how and in which order different processes are executed. We learned about two methods which are Priority Scheduling and Round Robin scheduling methods which clarified how multiple processes handle the removal of the process that is running from the CPU and the selection of other process based on a particular strategy.

Some hurdles needed to be passed during the research i.e. to find more detailed information about the topics which were overcome by researching via google scholar. The module materials also helped a lot to create the skeletal flow on how to approach the topic.

The coursework helped to enlighten about different terms that come in UNIX OS which I was able to understand. I was able to finally understand how different processes were managed in a computer system which is very useful.

References

Andrew S. Tanenbaum, H. B., 2014. Implementation of processes. In: T. Johnson, ed. *Modern Operating Systems.* California: Pearson, pp. 94-95.

Andrew S. Tanenbaum, H. B., 2014. Process Hierarchies. In: T. Johnson, ed. *Modern Operating System.* 4th ed. California: Pearson, pp. 91-92.

DBNSTJ, 2022. *DBNSTJ : A Process Management Scheme in a Network Operating System.* [Online]
Available at: https://dbnst.nii.ac.jp/english/detail/457
[Accessed 16 April 2022].

GeeksforGeeks, 2015. *Introduction of Process Management - GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-of-process-management/
[Accessed 16 April 2022].

GeeksforGeeks, 2017. *Process Table and Process Control Block (PCB) - GeeksforGeeks.* [Online]
Available at: https://www.geeksforgeeks.org/process-table-and-process-control-block-pcb/?ref=gcse
[Accessed 16 April 2022].

javatpoint, 2021. *OS Process States - javatpoint.* [Online]
Available at: https://www.javatpoint.com/os-process-states
[Accessed 16 April 2022].

Luther, E., 2012. *Process Control Block Operating system, Kernal (computing), Program Counter.* 1st ed. Santiago del Estero: Acu Publishing.

Sorfa, P., 2001. Debugging Memory on Linux. *Debugging Memory on Linux | Linux Journal,* 2001(87).

Thomas Sterling, M. A. M. B., 2018. Operating Systems: Process Control Block. In: M. A. M. B. Thomas Sterling, ed. *High Performance Computing Modern Systems and Practices.* s.l.:Morgan Kaufmann, pp. 347-362.

Tutorialspoint, 2022. *Operating System Scheduling algorithms: Tutorialspoint.* [Online]
Available at:

https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm#:~:text=Priority%20scheduling%20is%20a%20non,first%20come%20first%20served%20basis.
[Accessed 16 April 2022].

tutorialsSpace.com, 2022. *Implementation Of Processes In Operating System In HINDI.* [Online]
Available at: http://www.tutorialsspace.com/Operating-System/15-Processes-Implementation-Of-Processes.aspx#:~:text=Process%20Model%20is%20implemented%20by,setup%20data%20space%20for%20it%20.
[Accessed 6 April 2022].

webeduclick, 2019. *Process States in Operating System - Webeduclick.com.* [Online]
Available at: https://webeduclick.com/process-states-in-operating-system/
[Accessed 4 April 2022].

Appendix

Appendix – A (Glossary)

I/O: Input Output

UNIX: UNiplexed Information Computing System

PCB: Process Control Block

OS: Operating System

Kernel: An operating system's basic component that serves as the primary interface between the computer's physical hardware and the processes that execute on it.

Intrinsic: belonging to the essential nature or constitution of a thing

Pre-emptive: taken as a measure against something possible, anticipated, or feared

Appendix - B (Process Scheduling Queues)

Process scheduling is a technique used by the process manager that handles the removal of the process that is running from the CPU and the selection of other process based on a particular strategy. It is an essential part for multiprogramming operating system. Multiple processes can be loaded into executable memory at the same time in such operating systems, and the loaded processes share the CPU utilizing temporal multiplexing.

The operating system maintains following process scheduling queues:

Job queue

This queue helps to keep all the processes in the system

Ready queue

This queue maintains a list of all processes in main memory that are ready to run. This queue is always filled with new processes.

Device queues

This queue is made up of processes that have been halted due to the lack of an I/O device.

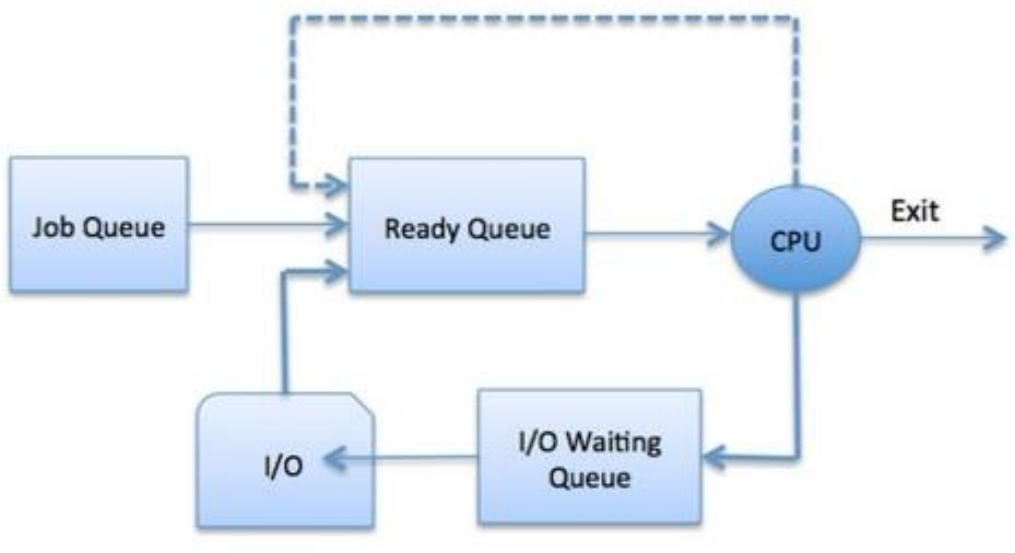

Figure 31 Process scheduling queue

The OS scheduler controls how processes are moved between the ready and run queues, each of which can only have one item per processor core on the system. It has been integrated with the CPU in the above figure. The OS is able to apply different policies to govern each que for example (Priority scheduling, Round Robin scheduling, FIFO, etc.)

Appendix -C (Priority Scheduling)

Priority scheduling is one of the most common scheduling algorithms in batch systems. In this system of scheduling each process is given or assigned a priority where the process with the highest priority gets executed first and the process on the same priority gets executed in first come first serve basis. Priorities are generally decided based on time requirements, memory requirements or other kinds of resource requirements.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0

Figure 32 Table of processes and their information

The following above table contains the processes along with their arrival time, execution time, priority and service time where 1 is the lowest priority and 3 is the highest priority.

So the processes would be scheduled and executed as follows:

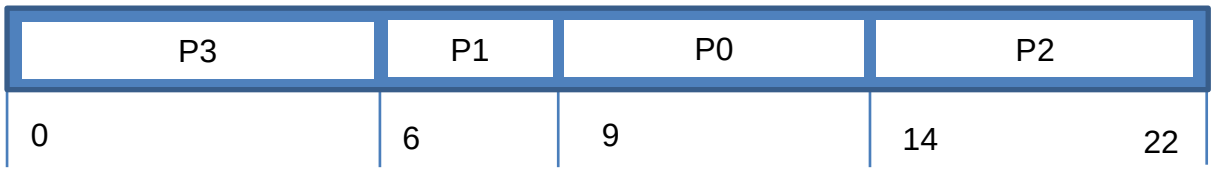

Figure 33 Execution Process order

As P3 has the highest priority it gets executed first, then P1 gets executed as it has the 2nd highest priority, then as P0 and P2 has the same priority The one which had come first will be execute i.e. P0 and at last P2 gets executed.

Waiting time for each process would be:

Process	Waiting Time
P0	0 – 0 = 0
P1	11 – 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

Table 18 Wait time for each process in Priority scheduling

Average waiting time: (0 + 10 + 12 + 2) / 4 =24 / 4 = 6

Appendix –D (Round Robin Scheduling)

Round Robin is another type of pre-emptive scheduling algorithm where each process is provided a fixed time to be executed called quantum. After a process has run for a set amount of time, it is pre-empted and another process runs for the same amount of time. Context switching is used for saving the states of the pre-empted processes (Tutorialspoint, 2022).

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0

Table 19 Table for process information

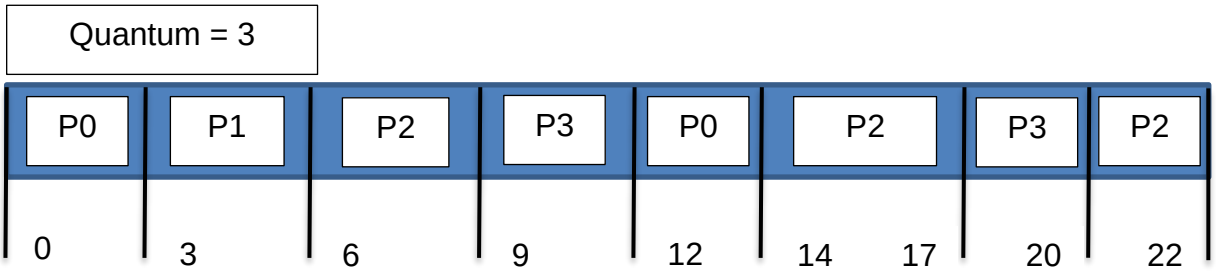

Figure 34 Process execution with Quantum=3

Wait time for each process:

Process	Wait Time: Service time – Arrival time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Table 20 Wait time for each process in Round Robin scheduling

Average wait time: (9+2+12+11) / 4 = 8.5

Appendix –E (Process deadlock)

A process in OS uses resources by requesting the resource, using the resource and then releasing the resource. A **deadlock** occurs when a group of processes is stalled because each process is holding a resource and waiting for another process to obtain it.

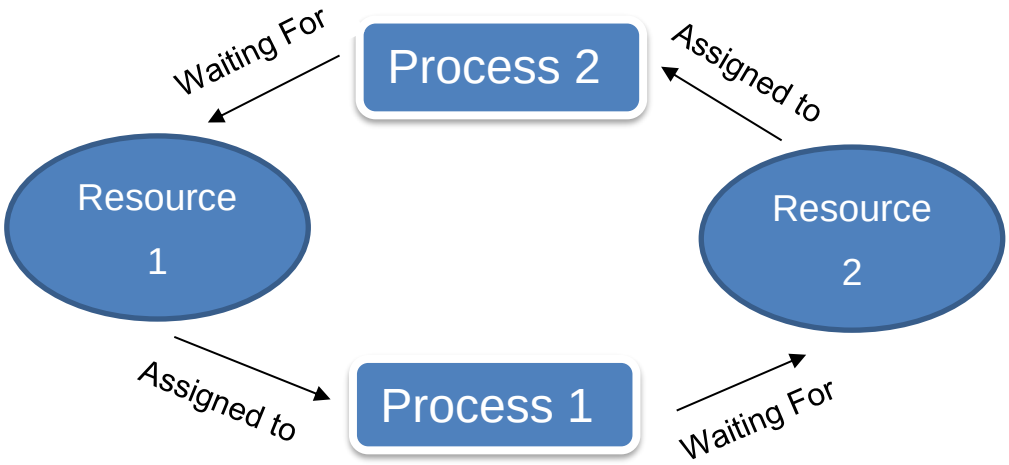

Figure 35 Process deadlock example

Considering an example when two train is coming from two opposite direction in the same one track and none of the trains are able to move once they are in front of each other. For example, from the above picture Process 1 is currently holding resource 1 and is waiting for resource 2 which is assigned to process 2, and process 2 is waiting for resource 1.

How deadlock arises:

- Mutual exclusion: 2 or more processes are not sharable
- Hold and wait: process holding at least one resource and is waiting for resources
- No Pre-emption: until the resource is released by the process a resource can't be taken from that process
- Circular wait: A set of processes are waiting for each other in circular form (GeeksforGeeks, 2015).

How deadlocks are handled:

- Deadlock prevention/ avoidance: prevention done by avoiding the deadlock from the above point from how it arises. By using strategy of “Avoidance”, we have to make an assumption. Prior to the process's execution, we must ensure that we have all of the knowledge regarding the resources that the process will require. We use Banker’s algorithm (Which is in-turn a gift from Dijkstra) in order to avoid deadlock (GeeksforGeeks, 2015).
- Deadlock detection/ recovery: Allow for deadlock to occur, then use pre-emption to deal with it after it has.
- Ignore: Let the deadlock happen (which is rare) then reboot the system which is an approach used by both Windows and UNIX.