Jadavpur University

Operating System lab report

MCA First Year(Second Semester)

2024

NAME: ARGHYA BARAN NASKAR

ROLL: 002310503049

Assignment :1

1. Write a shell script which accepts length and breadth of a rectangle and calculates the area and perimeter of the rectangle.

Code:

```
echo "Enter the length of the rectangle: "
read len
echo "Enter the breadth of the rectangle: "
read br

area=$((len*br))
peri=$((2*(len+br)))

echo "The area of the rectangle is: $area"
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment1$ ./one.sh
Enter the length of the rectangle:
4
Enter the breadth of the rectangle:
5
The area of the rectangle is: 20
The Perimeter of the rectangle is: 18
```

2. Write a shell script which accepts basic salary of an employee and calculates net salary and displays the salary slip.

Code:

```
echo "Enter base salary: "
read b
read -p "Enter the percentage of da: " da
daAmount=`expr $b \* $da`
daAmount=$(echo "scale=2; $daAmount / 100" | bc)
echo "daAmount = $daAmount"
net=$(echo "scale=2; $b+$daAmount" | bc)
echo "Basic Salary: $b, da: $da, daAmount: $daAmount, Net salary is: $net"
```

Output:

```
ab/scripts/Assignment1$ ./two.sh

Enter base salary:

60000
```

```
Enter the percentage of da: 10

daAmount = 6000.00

Basic Salary: 60000, da: 10, daAmount: 6000.00, Net salary is: 66000.00
```

3. Write a shell script which accepts a five digit number and prints sum of its digits.

Code:

```
sum=0

echo "Enter a number: "
read num

while [ "$num" -ne 0 ]
do
    rem=$(($num%10))
    num=$(($num/10))
    sum=$((sum+rem))
done

echo "Sum of the digits: $sum"
```

Output:

```
Enter a number:

45623

Sum of the digits: 20
```

4. Write a shell script which accepts a five digit number and prints the reverse number.

Code:

```
rev=0

read -p "Enter a number: " num
number=$num
while [ "$num" -ne 0 ]
do
    rem=$((num%10))
    num=$((num/10))
    rev=$((rev*10+rem))
done

echo "The reverse of $number is $rev"
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment1$ ./four.sh

Enter a number: 65473

The reverse of 65473 is 37456
```

5. The /etc/passwd file stores user account information. It contains one entry per line for each user (user

account) of the system. Each line contains seven fields which are separated by a colon (:) symbol. The fields

(i)Username
(ii)Password
(iii)User Id
(iv)Group Id
(v)User Id Info
(vi)Home Directory
(vii)Login Shell

Write a shell script which accepts a user login name and displays detail information about the users as available from the file /etc/passwd.

Code:

```
read -p "Enter the login name: " username

user_info=$(grep "^$username:" /etc/passwd)

if [ -z "$user_info" ]
then
    echo "User '$username' not found"
    exit 1
fi

IFS=':' read -r user pass uid gid info home shell <<< "$user_info"

echo "User Information for '$username':"
echo "---------------------------------"
echo "Username      : $user"
echo "Password      : $pass"
echo "User ID       : $uid"
echo "Group ID      : $gid"
echo "User ID Info  : $info"
echo "Home Directory: $home"
echo "Login Shell   : $shell"
echo "---------------------------------"a
```

Output:

```
 Enter the login name: arghya
User Information for 'arghya':
---------------------------------
Username      : arghya
Password      : x
User ID       : 1000
Group ID      : 1000
User ID Info  : ,,,
Home Directory: /home/arghya
Login Shell   : /bin/bash
```

Assignment :2

1. Write a shell script which, for all files in present directory displays whether it is a regular file or a directory.

Code:

```
for file in *
do
    if [ -f "$file" ]
    then
        echo "$file: is a file"
    fi
    if [ -d "$file" ]
    then
        echo "$file: is a directory"
    fi
done
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment2$ ./one.sh
five.sh: is a file
four.sh: is a file
one.sh: is a file
three.sh: is a file
two.sh: is a file
```

2. The PATH variable is an environment variable that contains an ordered list of paths that Linux will search for

executables when running a command. Write a shell script to display all the directories in the PATH
in a simple way, i.e., one line per directory. In addition, display information about each directory, such as the
permissions and the modification times.

Code:

```
IFS=':' read -r -a directories <<< "$PATH"

for dir in "${directories[*]}"
do
    echo "Directory: $dir"
    if [ -d "$dir" ]
    then
        stat -c "\nPermission: %A\nModification Time: %y" "$dir"
    fi
done
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment2$ ./two.sh
```

7

/usr/local/sbin
  Permissions: 755\n Modification Time: 2023-11-23 03:06:20.920297023 +0530
/usr/local/bin
  Permissions: 755\n Modification Time: 2023-11-23 03:06:20.916297034 +0530
/usr/sbin
  Permissions: 755\n Modification Time: 2024-05-27 16:12:49.707630474 +0530
/usr/bin
  Permissions: 755\n Modification Time: 2024-06-17 18:10:04.283782810 +0530
/sbin
  Permissions: 777\n Modification Time: 2023-11-23 03:06:20.256298855 +0530
/bin
  Permissions: 777\n Modification Time: 2023-11-23 03:06:20.252298866 +0530
/usr/games
  Permissions: 755\n Modification Time: 2022-04-18 15:58:59.000000000 +0530
/usr/local/games
  Permissions: 755\n Modification Time: 2023-11-23 03:06:20.916297034 +0530
/usr/lib/wsl/lib
  Permissions: 755\n Modification Time: 2024-06-18 00:36:08.265189080 +0530
/mnt/c/windows/system32
  Permissions: 555\n Modification Time: 2024-06-13 16:02:56.741459900 +0530
/mnt/c/windows
  Permissions: 555\n Modification Time: 2024-06-14 03:43:41.351082800 +0530
/mnt/c/windows/system32/Wbem
  Permissions: 555\n Modification Time: 2024-06-13 15:54:43.702954500 +0530
/mnt/c/Program Files/Python312/
  Permissions: 555\n Modification Time: 2024-04-29 02:10:19.396325700 +0530
/mnt/c/Program Files/Python312/Scripts/
  Permissions: 555\n Modification Time: 2024-04-29 02:10:19.446288400 +0530
/mnt/c/windows/System32/WindowsPowerShell/v1.0/
  Permissions: 555\n Modification Time: 2024-06-13 15:54:43.702954500 +0530
/mnt/c/windows/System32/OpenSSH/
  Permissions: 555\n Modification Time: 2022-05-07 11:40:04.064861500 +0530
/mnt/c/MinGW/bin
  Permissions: 777\n Modification Time: 2024-02-20 23:45:32.461008600 +0530
/mnt/c/Program Files/nodejs/

```
  Permissions: 555\n  Modification Time: 2024-02-21 01:34:04.600649100 +0530
/mnt/c/Program Files/MySQL/MySQL Server 8.0/bin
  Permissions: 555\n  Modification Time: 2024-03-23 00:44:14.197087200 +0530
/mnt/c/Program Files/MongoDB/Server/7.0/bin
  Permissions: 555\n  Modification Time: 2024-03-25 00:53:11.354714100 +0530
/mnt/c/Program Files/Git/cmd
  Permissions: 555\n  Modification Time: 2024-04-04 20:38:07.942920300 +0530
/mnt/c/Users/User/AppData/Local/Programs/Eclipse Adoptium/jdk-17.0.11.9-hotspot/bin
  Permissions: 777\n  Modification Time: 2024-04-29 02:24:02.584691700 +0530
/mnt/c/Users/User/AppData/Local/Programs/Python/Launcher/
  Permissions: 777\n  Modification Time: 2024-04-29 02:00:26.542475200 +0530
/mnt/c/Program Files/MySQL/MySQL Shell 8.0/bin/
  Permissions: 555\n  Modification Time: 2024-03-23 00:45:09.238511800 +0530
/mnt/c/Users/User/AppData/Local/Microsoft/WindowsApps
  Permissions: 777\n  Modification Time: 2024-06-17 15:40:28.571024000 +0530
/mnt/c/Users/User/AppData/Local/Programs/Microsoft VS Code/bin
  Permissions: 777\n  Modification Time: 2024-06-14 01:57:56.821939500 +0530
/mnt/c/Users/User/AppData/Roaming/npm
  Permissions: 777\n  Modification Time: 2024-05-25 18:32:16.990182700 +0530
/mnt/c/Users/User/AppData/Local/Programs/mongosh/
  Permissions: 777\n  Modification Time: 2024-03-25 01:05:39.214075800 +0530
/snap/bin
  Permissions: 755\n  Modification Time: 2024-05-26 18:07:34.033766533 +0530
```

3. Write a shell script which displays vendor-id, model name, cpu MHz, cache size information about the processor present in your computer. Hint: most of this information can be obtained bv reading the file /proc/cpuinfo. Ans-

Code:

```
grep -m 1 "vendor_id" /proc/cpuinfo
grep -m 1 "model name" /proc/cpuinfo
grep -m 1 "cpu MHz" /proc/cpuinfo
grep -m 1 "cache size" /proc/cpuinfo
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment2$ ./three.sh
vendor_id       : AuthenticAMD
model name      : AMD Ryzen 5 7520U with Radeon Graphics
```

```
cpu MHz         : 2794.657
cache size      : 512 KB
```

4. Write a shell script to show your home directory, Operating System type, version, release number, kernel version and current path setting. Hint: use uname command or use content of /proc/sys/kernel/osrelease file. Ans-

Code:

```
echo "Home Directory: $HOME"
echo "Operating System Type: $(uname -o)"
echo "Operating System Version: $(uname -v)"
echo "OS Release Number: $(uname -r)"
echo "Kernel Version: $(uname -s)"
echo "Current Path Directory: $PATH"
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment2$ ./four.sh

Home Directory: /home/arghya

Operating System Type: GNU/Linux

Operating System Version: #1 SMP Fri Mar 29 23:14:13 UTC 2024

OS Release Number: 5.15.153.1-microsoft-standard-WSL2

Kernel Version: Linux

Current Path Directory:
/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/
games:/home/arghya/.dotnet/tools
```

5. Write a shell script to display a summary of the disk space usage för each directory argument (and any subdirectories), both in terms of bytes. and kilobytes or megabytes (whichever is appropriate). (du -b]

Code:

```
echo "Enter path of the directory:"

read path

du -h $path

du -b $path
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment2$ ./five.sh

Enter path of the directory:

/home/arghya/Lab/scripts/Assignment2

24K     /home/arghya/Lab/scripts/Assignment2

4912    /home/arghya/Lab/scripts/Assignment2
```

Assignment :3

1. Write a shell script which reads a input file that contains three integers in each line. The script should display the sum of all integers in each line.

Code:

```
if [ $# -eq 0 ]
then
  echo "\nScript not being run correctly..."
  echo "Usage: $0 filename"
  exit 1
fi

while IFS=' ' read -r num1 num2 num3
do
  sum=$((num1 + num2 + num3))
  echo "Sum of integers $num1, $num2, $num3 is: $sum"
done < "$1"
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment3$ ./one.sh text

Sum of integers 1, 2, 3 is: 6

Sum of integers 4, 5, 6 is: 15

Sum of integers 7, 8, 9 is: 24
```

2. Write a shell script to find out how many file and directory are there in the current directory. Also list the

file and directory names separately.

Code:

```
file_count=0
dir_count=0

files=()
directories=()

for item in *; do
    if [ -f "$item" ]; then
        files+=("$item")
        ((file_count++))
    elif [ -d "$item" ]; then
        directories+=("$item")
        ((dir_count++))
    fi
done

echo "Files in the current directory:"
printf '%s\n' "${files[@]}"
```

```
echo "Directories in the current directory:"
printf '%s\n' "${directories[@]}"

echo "Total files: $file_count"
echo "Total directories: $dir_count"
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment3$ ./two.sh
Files in the current directory:
five.sh
four.sh
one.sh
rev_five.sh
six.sh
text
three.sh
two.sh
Directories in the current directory:

Total files: 8
Total directories: 0
```

3. Write a script that adds up the sizes reported by the Is command for the files in the current directory. The

script should print out only the total number of bvtes used.

Code:

```
sum=$(ls -l | awk '{total += $5} END {print total}')
echo "Total size of all files in current directory: $sum bytes"
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment3$ ./three.sh

Total size of all files in current directory: 1496 bytes
```

4. Write a shell scripts that delete all temporary les (end with ~) in current directory.

Code:

```
for file in *~; do
  if [ -f "$file" ]; then
    rm "$file"
    echo "Deleted: $file"
  fi
done
```

Output:

```
There were no temporary files so no output shown
```

5. Write a shell script to rename file having extension .sh to .exe.
Code:

```
      for file in *.sh; do
    if [ -f "$file" ]; then
        new_name="${file%.sh}.exe"
        mv "$file" "$new_name"
        echo "Renamed: $file -> $new_name"
    fi
done
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment3$ ./five.sh
Renamed: five.sh -> five.exe
Renamed: four.sh -> four.exe
Renamed: one.sh -> one.exe
Renamed: rev_five.sh -> rev_five.exe
Renamed: six.sh -> six.exe
Renamed: three.sh -> three.exe
Renamed: two.sh -> two.exe
```

6. Write a shell script to count number of shell scripts (with .sh extension) present in the current directory.

Code:

```
count=0

for file in *; do
    if [ -f "$file" ] && [[ "$file" == *.sh ]]; then
        ((count++))
    fi
done

echo "Total number of shell scripts (.sh) in the current directory: $count"
```

Output:

```
      arghya@LAPTOP-MQQ6UVEK:~/Lab/scripts/Assignment3$ ./six.sh
Total number of shell scripts (.sh) in the current directory: 7
```

MENU-DRIVEN PROGRAM FOR ASSIGMENT 1,2,3

Code:

```
ROOT_DIR=$(pwd)

choose_directory() {
  while true; do
    echo "Choose a directory:"
    echo "1) Assignment 1"
    echo "2) Assignment 2"
    echo "3) Assignment 3"
    echo "4) Quit"
    read -p "Enter your choice: " choice

    case $choice in
      1) DIRECTORY="Assignment 1";;
      2) DIRECTORY="Assignment 2";;
      3) DIRECTORY="Assignment 3";;
      4) echo "Quitting..."; exit 0 ;;
      *) echo "Invalid option. Try again." ; continue ;;
    esac

    cd "$ROOT_DIR/$DIRECTORY" || { echo "\nFailed to change directory"; exit 1; }
    list_scripts
  done
}

list_scripts() {
  while true; do
    echo "Scripts in $DIRECTORY:"
    ls
    echo "Enter the name of the script to run (type 'back' to change directory):"
    read -p "Script name: " script
```

```
    if [ "$script" == "back" ]; then
      cd "$ROOT_DIR"
      return
    elif [ "$DIRECTORY" == "Assignment 3" ] && [ "$script" == "one.sh" ]; then
       echo "Enter filename: "
       read filename
       ./one.sh $filename
    elif [ -f "$script" ]; then
      chmod +x "$script"
      echo "Running $script..."
      ./"$script"
    else
      echo "Script not found. Try again."
    fi
  done
}

choose_directory
```

Assignment :4

1. Write a C program to create a child process. The parent process must wait until the child finishes. Both the processes must print their own pid and parent pid. Additionally the parent process should print the exit status of the child.

Code:

```
#include<unistd.h> //for fork()
#include<stdio.h>
#include<stdlib.h> // for EXIT_FAILURE
#include<sys/types.h> // for pid_t
#include<sys/wait.h>
// #include<errno.h>  //for error handling -> perror()


int main(){
    pid_t pid;
    int status;

    pid = fork();

    if(pid < 0){
        perror("fork failure");
        exit(EXIT_FAILURE);
    }

    if(pid == 0){
        printf("Child process: PID = %d, parent PID = %d\n", getpid(), getppid());
        sleep(2); //simulate some work in the child process
        exit(EXIT_SUCCESS);
    }else {
        printf("Parent process: PID = %d, parent PID = %d\n", getpid(), getppid());

        if(wait(&status) == -1){
            perror("wait failed");
            exit(EXIT_FAILURE);
        }
```

```
        if(WIFEXITED(status)){
            printf("parent process: Child exited with status = %d\n",
WEXITSTATUS(status));
        }else{
            printf("Parent process: Child did not exit successfully\n");
        }
    }

    return 0;
}
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 4$ gcc one.c
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 4$ ./a.out
Parent process: PID = 40882, parent PID = 40341
Child process: PID = 40883, parent PID = 40882
parent process: Child exited with status = 0
```

2. Write a C program which prints prime numbers between the range 1 to 10,00,000 by creating ten child processes and subdividing the task equally among all child processes, i.e., the first child should print prime numbers in the range 1 to 1,00,000, the second child in the range 1,00,001 to 2,00,000, ... The child processes must run in parallel and the parent process must wait until all the child processes finish.

Code:

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>

#define NUM_CHILDREN 10
#define RANGE_SIZE 10000

int is_prime(int number){
      if(number <= 1) return 0; // 0 and 1 are not prime
      if(number <=3) return 1; // 2 and 3 are primes

      for(int i = 2; i*i<=number; i++){
            if(number % i == 0) return 0;
      }

      return 1;
}

void print_prime(int start, int end){
      for(int i = start; i<= end; i++){
```

```
            if(is_prime(i)) printf("%d ", i);
      }
      printf("\n");
}

int main(){
      pid_t pid;
      int range_start = 1;
      int segment_size = RANGE_SIZE / NUM_CHILDREN;

      for(int i = 0; i< NUM_CHILDREN; i++){
            pid = fork();

            if(pid < 0){
                  perror("fork failed");
                  exit(EXIT_FAILURE);
            }

            if(pid == 0){
                  printf("child %d: PID = %d, parent PID = %d\n", i+1, getpid(),
getppid());

            int child_start = segment_size*i + range_start;
            int child_end = child_start + segment_size -1;

            print_prime(child_start, child_end);

                  exit(EXIT_SUCCESS);
            }
      }

    for(int i = 0; i<NUM_CHILDREN; i++){
            wait(NULL);
      }

      return 0;
}
```

Output:

```
child 1: PID = 42172, parent PID = 42171

child 2: PID = 42173, parent PID = 42171

child 3: PID = 42174, parent PID = 42171

child 4: PID = 42175, parent PID = 42171

child 5: PID = 42176, parent PID = 42171

child 6: PID = 42177, parent PID = 42171

child 9: PID = 42180, parent PID = 42171

child 8: PID = 42179, parent PID = 42171

5003 5009 5011 5021 5023 5039 5051 5059 5077 5081 5087 5099 5101 5107 5113 5119 5147
5153 5167 5171 5179 5189 5197 5209 5227 5231 5233 5237 5261 5273 5279 5281 5297 5303
5309 5323 5333 5347 5351 5381 5387 5393 5399 5407 5413 5417 5419 5431 5437 5441 5443
5449 5471 5477 5479 5483 5501 5503 5507 5519 5521 5527 5531 5557 5563 5569 5573 5581
5591 5623 5639 5641 5647 5651 5653 5657 5659 5669 5683 5689 5693 5701 5711 5717 5737
```

5741 5743 5749 5779 5783 5791 5801 5807 5813 5821 5827 5839 5843 5849 5851 5857 5861
5867 5869 5879 5881 5897 5903 5923 5927 5939 5953 5981 5987

8009 8011 8017 8039 8053 8059 8069 8081 8087 8089 8093 8101 8111 8117 8123 8147 8161
8167 8171 8179 8191 8209 8219 8221 8231 8233 8237 8243 8263 8269 8273 8287 8291 8293
8297 8311 8317 8329 8353 8363 8369 8377 8387 8389 8419 8423 8429 8431 8443 8447 8461
8467 8501 8513 8521 8527 8537 8539 8543 8563 8573 8581 8597 8599 8609 8623 8627 8629
8641 8647 8663 8669 8677 8681 8689 8693 8699 8707 8713 8719 8731 8737 8741 8747 8753
8761 8779 8783 8803 8807 8819 8821 8831 8837 8839 8849 8861 8863 8867 8887 8893 8923
8929 8933 8941 8951 8963 8969 8971 8999

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109
113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353
359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479
487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617
619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907
911 919 929 937 941 947 953 967 971 977 983 991 997

child 7: PID = 42178, parent PID = 42171

child 10: PID = 42181, parent PID = 42171

4001 4003 4007 4013 4019 4021 4027 4049 4051 4057 4073 4079 4091 4093 4099 4111 4127
4129 4133 4139 4153 4157 4159 4177 4201 4211 4217 4219 4229 4231 4241 4243 4253 4259
4261 4271 4273 4283 4289 4297 4327 4337 4339 4349 4357 4363 4373 4391 4397 4409 4421
4423 4441 4447 4451 4457 4463 4481 4483 4493 4507 4513 4517 4519 4523 4547 4549 4561
4567 4583 4591 4597 4603 4621 4637 4639 4643 4649 4651 4657 4663 4673 4679 4691 4703
4721 4723 4729 4733 4751 4759 4783 4787 4789 4793 4799 4801 4813 4817 4831 4861 4871
4877 4889 4903 4909 4919 4931 4933 4937 4943 4951 4957 4967 4969 4973 4987 4993 4999

6007 6011 6029 6037 6043 6047 6053 6067 6073 6079 6089 6091 6101 6113 6121 6131 6133
6143 6151 6163 6173 6197 6199…

3. Write a C program which creates a child process. The parent process sends a string (input by user) which the

child process inspects and sends ”YES” back to the parent if the string is a palindrome, otherwise it sends ”NO”. The IPC to be used is pipe. Both the processes terminate when the input string is “quit”

Code:

```
      #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define BUFFER_SIZE 100

int is_palindrome(const char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            return 0;
        }
    }
```

```
    return 1;
}

int main() {
    int parent_to_child[2]; // Pipe from parent to child
    int child_to_parent[2]; // Pipe from child to parent
    char buffer[BUFFER_SIZE];

    // Create pipes
    if (pipe(parent_to_child) == -1 || pipe(child_to_parent) == -1) {
        perror("pipe failed");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // Child process
        close(parent_to_child[1]); // Close write end of pipe from parent
        close(child_to_parent[0]); // Close read end of pipe to parent

        while (1) {
            // Read string from parent
            if (read(parent_to_child[0], buffer, BUFFER_SIZE) == -1) {
                perror("read failed");
                exit(EXIT_FAILURE);
            }

            // Check if the string is "quit"
            if (strcmp(buffer, "quit") == 0) {
                // Send "quit" to parent and exit
                if (write(child_to_parent[1], "quit", strlen("quit") + 1) == -1) {
                    perror("write failed");
                    exit(EXIT_FAILURE);
                }
                exit(EXIT_SUCCESS);
            }

            // Check if the string is a palindrome
            if (is_palindrome(buffer)) {
                if (write(child_to_parent[1], "YES", strlen("YES") + 1) == -1) {
                    perror("write failed");
                    exit(EXIT_FAILURE);
                }
            } else {
                if (write(child_to_parent[1], "NO", strlen("NO") + 1) == -1) {
                    perror("write failed");
                    exit(EXIT_FAILURE);
                }
            }
        }
    } else {
        // Parent process
        close(parent_to_child[0]); // Close read end of pipe from parent
```

```
        close(child_to_parent[1]); // Close write end of pipe to parent

        while (1) {
            printf("Enter a string: ");
            fgets(buffer, BUFFER_SIZE, stdin);
            buffer[strcspn(buffer, "\n")] = '\0'; // Remove trailing newline

            // Send string to child
            if (write(parent_to_child[1], buffer, strlen(buffer) + 1) == -1) {
                perror("write failed");
                exit(EXIT_FAILURE);
            }

            // Wait for response from child
            if (read(child_to_parent[0], buffer, BUFFER_SIZE) == -1) {
                perror("read failed");
                exit(EXIT_FAILURE);
            }

            if (strcmp(buffer, "quit") == 0) {
                break; // Exit loop if child sent "quit"
            }

            // Print result
            printf("Child response: %s\n", buffer);
        }
    }

    return 0;
}
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 4$ ./a.out
Enter a string: madaM

Child response: NO

Enter a string: MadaM

Child response: YES

Enter a string: adkfjls

Child response: NO

Enter a string: QuIt

Child response: NO

Enter a string: quit
```

4. Write a C program which prints the following menu

1. **ls**
2. **pwd**
3. **uname**
4. **exit**

When, the user provides an input, the parent process creates a child process [if user's choice is between 1-3]

**and executes the corresponding command [use execv() system call]. The main process waits for the child to
finish and displays the menu again. The parent process terminates if user’s choice is 4.**

Code:

```
      #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_COMMAND_LEN 50

int main() {
    int choice;
    char command[MAX_COMMAND_LEN];
    char *args[2];

    while (1) {
        printf("1. ls\n");
        printf("2. pwd\n");
        printf("3. uname\n");
        printf("4. exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        // Flush stdin to clear any remaining characters
        while (getchar() != '\n');

        switch (choice) {
            case 1:
                strcpy(command, "/bin/ls");
                args[0] = command;
                args[1] = NULL;
                break;
            case 2:
                strcpy(command, "/bin/pwd");
                args[0] = command;
                args[1] = NULL;
                break;
            case 3:
                strcpy(command, "/bin/uname");
                args[0] = command;
                args[1] = NULL;
                break;
            case 4:
                exit(EXIT_SUCCESS);
            default:
                printf("Invalid choice. Please enter a number between 1 and 4.\n");
                continue;
        }

        pid_t pid = fork();

        if (pid < 0) {
            perror("fork failed");
            exit(EXIT_FAILURE);
```

```
        }

        if (pid == 0) {
            // Child process
            execv(args[0], args);
            // If execv returns, an error occurred
            perror("execv failed");
            exit(EXIT_FAILURE);
        } else {
            // Parent process
            wait(NULL); // Wait for the child to finish
        }
    }

    return 0;
}
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 4$ gcc four.c
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 4$ ./a.out
1. ls
2. pwd
3. uname
4. exit
Enter your choice: 1
a.out  four.c  one.c  three.c  two.c
1. ls
2. pwd
3. uname
4. exit
Enter your choice: 2
/home/arghya/Lab/practice/Assignment 4
1. ls
2. pwd
3. uname
4. exit
Enter your choice: 3
Linux
1. ls
2. pwd
3. uname
4. exit
Enter your choice: 4
```

Assignment :5

1. Write a C program which creates a child process. The parent and child process communicate using a shared memory segment. The parent process generates 100 random integers and writes it into the shared memory segment. The child process then computes the maximum, minimum and average of all these 100 numbers and writes the result back into the shared memory segment, from where the parent process reads the result and displays it. Add appropriate code to synchronize the parent and child process. [Hint: It is an example of strict alteration where access to the shared memory segment alternates between the parent and child process]

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/wait.h>

#include <sys/ipc.h>
#include <sys/sem.h>

#define NUM_VALUES 10

typedef struct {
    int max;
    int min;
    double avg;
} Result;

struct sembuf P = {0, -1, 0}; // P operation wait
struct sembuf V = {0, 1, 0};  // V operation siganl

void generateRandomNumbers(double* numbers) {
    srand(time(NULL));
    for (int i = 0; i < NUM_VALUES; i++) {
        numbers[i] = rand() % 1000;
    }
}

int main() {
    const int SIZE = (NUM_VALUES+5)*sizeof(double);
    pid_t childPid;
    Result* result;

    //creating semaphore
    key_t key = ftok("semfile", 65);
```

```
    int sem_A = semget(key, 1, 0666 | IPC_CREAT);

    semctl(sem_A, 0, SETVAL, 0);

    // Create child process
    childPid = fork();

    if (childPid < 0) {
        perror("fork");
        exit(1);
    } else if (childPid == 0) {
        // Child process

        // usleep(1000);

        semop(sem_A, &P, 1);

        const char *name = "q2_shm";

        int shm_fd;
        double *ptr;

        shm_fd = shm_open(name, O_RDWR, 0666);
        // ftruncate(shm_fd, SIZE);


        ptr = (double *)mmap(0, SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, shm_fd, 0);


        double max = ptr[0];
        double min = ptr[0];
        // printf("\nmax%.2f\n", max);
        // printf("\nmin %.2f\n", min);
        double s = 0;
        for(int i=0;i<NUM_VALUES;i++){
            // printf("%d ", ptr[i]);
            if(ptr[i]>max){
                // printf("\ngreater %.2f\n", ptr[i]);
                max = ptr[i];
            }
            if(ptr[i]<min){
                // printf("\nsmaller %.2f\n", ptr[i]);
                min = ptr[i];
            }
            s += ptr[i];
        }
        ptr[NUM_VALUES] = max;
        ptr[NUM_VALUES+1] = min;
        ptr[NUM_VALUES+2] = s/NUM_VALUES;
        // semop(sem_A, &V, 1);

        semctl(sem_A, 0, IPC_RMID, 0);

        exit(0);
    } else {
        // Parent process
        const char *name = "q2_shm";

        int shm_fd;
```

```
        double *ptr;

        shm_fd = shm_open(name, O_CREAT|O_RDWR, 0666);
        ftruncate(shm_fd, SIZE);


        ptr = (double *)mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

        double numbers[NUM_VALUES];
        generateRandomNumbers(numbers);

        for(int i = 0; i < NUM_VALUES; i++){
            printf("%.1f ", numbers[i]);
        }
        // Write numbers into shared memory
        for (int i = 0; i < NUM_VALUES; i++) {
            // sharedMemory[i] = numbers[i];
            ptr[i] = numbers[i];
        }
        printf("\n");
        semop(sem_A, &V, 1);

        wait(NULL);

        printf("Maximum: %.2f\n", ptr[NUM_VALUES]);
        printf("Minimum: %.2f\n", ptr[NUM_VALUES+1]);
        printf("Average: %.2f\n", ptr[NUM_VALUES+2]);

        semctl(sem_A, 0, IPC_RMID, 0);
    }

    return 0;
}
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 5$ gcc one.c
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 5$ ./a.out
555.0 641.0 943.0 664.0 336.0 960.0 376.0 333.0 265.0 816.0
Maximum: 960.00
Minimum: 265.00
Average: 588.90
```

2. P1, P2 and P3 are three processes executing their respective tasks. They should synchronize among themselves

using semaphores such that the string ABCCAB gets printed 10 times. Write codes for process P1, P2 and
P3 to get the desired output. [Hint: Write code for the main process which creates and initializes necessary
semaphores and then creates three child processes for executing tasks of process P1, P2 and P3 respectively.]

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/wait.h>


struct sembuf P= {0, -1, 0}; //P operation
struct sembuf V = {0, 1, 0}; //V operation

void P1(int sem_A, int sem_B){
    int val_A;
    for(int i = 0; i<20; i++){
        semop(sem_A, &P, 1);

        printf("A");

        fflush(stdout);

        semop(sem_B, &V, 1);
    }
}

void P2(int sem_B, int sem_C, int sem_A){
    int turn = 0;
    for(int i = 0; i<20; i++){
        semop(sem_B, &P, 1);

       printf("B");
       if((i & 1)) printf(" ");

        fflush(stdout);
        if(turn == 0){
            semop(sem_C, &V, 1);
            turn++;
        }else{
            semop(sem_A, &V, 1);
            turn = 0;
        }
    }
}


void P3(int sem_C, int sem_A){
    int turn = 0;
    for(int i = 0; i<20; i++){
        semop(sem_C, &P, 1);

        printf("C");

        fflush(stdout);
        if(turn == 0){
            semop(sem_C, &V, 1);
            turn = 1;
        }else{
```

```
            semop(sem_A, &V, 1);
            turn = 0;
        }
    }
}


int main(){
    key_t key = ftok("semfile", 65); //Generating a unique key

    int sem_A = semget(key, 1, 0666 | IPC_CREAT);
    int sem_B = semget(key+1, 1, 0666 | IPC_CREAT);
    int sem_C = semget(key+2, 1, 0666 | IPC_CREAT);

    semctl(sem_A, 0, SETVAL, 1);
    semctl(sem_B, 0, SETVAL, 0);
    semctl(sem_C, 0, SETVAL, 0);

    if(fork() == 0){
        P1(sem_A, sem_B);
        exit(0);
    }

    if(fork() == 0){
        P2(sem_B, sem_C, sem_A);
        exit(0);
    }

    if(fork() == 0){
        P3(sem_C, sem_A);
        exit(0);
    }

    for(int i = 0; i<3; i++){
        wait(NULL);
    }

    printf("\n");

    semctl(sem_A, 0, IPC_RMID, 0);
    semctl(sem_B, 0, IPC_RMID, 0);
    semctl(sem_C, 0, IPC_RMID, 0);

    return 0;
}
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 5$ gcc q2.c

arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 5$ ./a.out

ABCCAB ABCCAB ABCCAB ABCCAB ABCCAB ABCCAB ABCCAB ABCCAB ABCCAB ABCCAB
```

3. Implement the solution to the producer-consumer problem using semaphores

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>

#define BUFFER_SIZE 10
#define NUM_ITEMS 1000
#define SHM_NAME "/shared_memory"

typedef struct {
    int buffer[BUFFER_SIZE];
    int in;
    int out;
    sem_t empty;
    sem_t full;
    sem_t mutex;
} shared_data;

void producer(shared_data *data) {
    int item;
    for (int i = 0; i < NUM_ITEMS; i++) {
        item = rand() % 100;

        // Check if buffer is full
        if (sem_trywait(&data->empty) == -1) {
            printf("Buffer is full. Waiting...\n");
            sem_wait(&data->empty); // Wait for an empty slot
        }

        // Wait for exclusive access to the buffer
        sem_wait(&data->mutex);
```

```
        // Add the item to the buffer
        data->buffer[data->in] = item;
        data->in = (data->in + 1) % BUFFER_SIZE;
        printf("Producer produced item %d at: %d\n", item, data->in);

        // Release exclusive access to the buffer
        sem_post(&data->mutex);
        // Signal that a full slot is available
        sem_post(&data->full);

        sleep(1); // Producer's production time
    }
}

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    shared_data *data = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (data == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    producer(data);

    munmap(data, sizeof(shared_data));

    return 0;
}
```

Consumer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>

#define BUFFER_SIZE 10
#define NUM_ITEMS 1000  // Increased number of items for continuous consumption
#define SHM_NAME "/shared_memory"

typedef struct {
    int buffer[BUFFER_SIZE];
    int in;
    int out;
    sem_t empty;
    sem_t full;
    sem_t mutex;
} shared_data;

void consumer(shared_data *data) {
    int item;
    for (int i = 0; i < NUM_ITEMS; i++) {
        // Check if buffer is empty
        if (sem_trywait(&data->full) == -1) {
            printf("No item to consume. Waiting...\n");
            sem_wait(&data->full); // Wait for a filled slot
        }
        // Wait for exclusive access to the buffer
        sem_wait(&data->mutex);

        // Remove the item from the buffer
        item = data->buffer[data->out];
        data->out = (data->out + 1) % BUFFER_SIZE;
        printf("Consumer consumed item %d from: %d\n", item, data->out);
```

```
        // Release exclusive access to the buffer
        sem_post(&data->mutex);
        // Signal that an empty slot is available
        sem_post(&data->empty);

        sleep(2); // Consumer's consumption time
    }
}

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    shared_data *data = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (data == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    consumer(data);

    munmap(data, sizeof(shared_data));

    return 0;
}
```

Executor.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "shared.h"
#include<time.h>

int main() {
    pid_t pid;
    int num_readers = 3;
    int num_writers = 2;

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, sizeof(shared_data)) == -1) {
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    shared_data *data = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (data == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    data->read_count = 0;
    //the third argument initializes the value of the semaphore
    //rw_mutex semaphore is used to control access to the shared resource when writing,
    // ensuring that only one writer can access the resource at a time.
    sem_init(&data->rw_mutex, 1, 1); // semaphore is shared between threads or processes.
A value 0 will
    sem_init(&data->mutex, 1, 1); // indicate semaphore is private to the current process.
```

```
    //data->mutex to provide mutual exclusion while accessing read_count


    for (int i = 0; i < num_readers; i++) {
        if ((pid = fork()) == 0) {
            char *args[] = {"./reader", NULL};
            execv(args[0], args);
            perror("execv");
            exit(EXIT_FAILURE);
        } else if (pid < 0) {
            perror("fork");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_writers; i++) {
        if ((pid = fork()) == 0) {
            char *args[] = {"./writer", NULL};
            execv(args[0], args);
            perror("execv");
            exit(EXIT_FAILURE);
        } else if (pid < 0) {
            perror("fork");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_readers + num_writers; i++) {
        wait(NULL);
    }

    printf("All readers and writers have finished execution.\n");

    sem_destroy(&data->rw_mutex);
    sem_destroy(&data->mutex);
    munmap(data, sizeof(shared_data));
```

```
    shm_unlink(SHM_NAME);

    return 0;
}
```

Output:

```
arghya@LAPTOP-MQQ6UVEK:~/Lab/practice/Assignment 5$ ./a.out

Reader 49990: reading data: 0 0 0 0
Reader 49989: reading data: 0 0 0 0

Writer 49991: writing data

Reader 49988: reading data: 6 28 61 39

Writer 49992: writing data


Reader 49990: reading data: 6 28 61 39
Reader 49989: reading data: 6 28 61 39

Reader 49988: reading data: 6 28 61 39
6 28 61 39
Writer 49991: writing data
6 28 61 39
Writer 49992: writing data


Reader 49990: reading data: 57 24 19 57
Reader 49989: reading data: 57 24 19 57

Reader 49988: reading data: 57 24 19 57
57 24 19 57
Writer 49991: writing data
57 24 19 57
Writer 49992: writing data
87 93 24 13
Writer 49991: writing data
87 93 24 13
Writer 49992: writing data


Reader 49990: reading data: 6 51 63 61
Reader 49989: reading data: 6 51 63 61

.

.

.
```

3. Implement the solution to the Reader-Writers problem using semaphores

Code:

Code:

Shared.h

```
#ifndef SHARED_H
#define SHARED_H

#include <semaphore.h>

#define SHM_NAME "/rw_shared_memory"
#define BUFFER_SIZE 4

typedef struct {
    int data[BUFFER_SIZE];
    sem_t rw_mutex;
    sem_t mutex;
    int read_count;
} shared_data;

#endif
```

Reader.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "shared.h"
#include<time.h>

void reader(shared_data *data) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&data->mutex);
        data->read_count++;
        if (data->read_count == 1) {
            sem_wait(&data->rw_mutex);
        }
        sem_post(&data->mutex);

        printf("\nReader %d: reading data: ", getpid()); //reading data
```

```
        for (int j = 0; j < BUFFER_SIZE; j++) {
            printf("%d ", data->data[j]);
        }
        printf("\n");

        sem_wait(&data->mutex);
        data->read_count--;
        if (data->read_count == 0) {
            sem_post(&data->rw_mutex);
        }
        sem_post(&data->mutex);

        usleep(rand() % 1000000);
    }
}

int main() {
    srand(time(NULL));

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    shared_data *data = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (data == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    reader(data);

    munmap(data, sizeof(shared_data));
```

```
    return 0;
}
```

Writer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "shared.h"
#include<time.h>

void writer(shared_data *data) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&data->rw_mutex);

        // Writing data
        printf("\nWriter %d: writing data\n", getpid());
        for (int j = 0; j < BUFFER_SIZE; j++) {
            data->data[j] = rand() % 100;
            printf("%d ", data->data[j]);
        }

        sem_post(&data->rw_mutex);
        usleep(rand() % 1000000);
    }
}

int main() {
    srand(time(NULL));

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
```

```
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    shared_data *data = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (data == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }



    writer(data);


    munmap(data, sizeof(shared_data));

    return 0;
}
```

Executor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "shared.h"
#include<time.h>

int main() {
    pid_t pid;
    int num_readers = 3;
    int num_writers = 2;
```

```
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, sizeof(shared_data)) == -1) {
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    shared_data *data = (shared_data *)mmap(NULL, sizeof(shared_data), PROT_READ |
PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (data == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    data->read_count = 0;
    //the third argument initializes the value of the semaphore
    //rw_mutex semaphore is used to control access to the shared resource when
writing,
    // ensuring that only one writer can access the resource at a time.
    sem_init(&data->rw_mutex, 1, 1); // semaphore is shared between threads or
processes. A value 0 will
    sem_init(&data->mutex, 1, 1); // indicate semaphore is private to the current
process.
    //data->mutex to provide mutual exclusion while accessing read_count



    for (int i = 0; i < num_readers; i++) {
        if ((pid = fork()) == 0) {
            char *args[] = {"./reader", NULL};
            execv(args[0], args);
            perror("execv");
            exit(EXIT_FAILURE);
        } else if (pid < 0) {
```

```
            perror("fork");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_writers; i++) {
        if ((pid = fork()) == 0) {
            char *args[] = {"./writer", NULL};
            execv(args[0], args);
            perror("execv");
            exit(EXIT_FAILURE);
        } else if (pid < 0) {
            perror("fork");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < num_readers + num_writers; i++) {
        wait(NULL);
    }

    printf("All readers and writers have finished execution.\n");

    sem_destroy(&data->rw_mutex);
    sem_destroy(&data->mutex);
    munmap(data, sizeof(shared_data));
    shm_unlink(SHM_NAME);

    return 0;
}
```

Output:

```
Reader 54021: reading data: 0 0 0 0
Reader 54020: reading data: 0 0 0 0
```

```
Reader 54022: reading data: 0 0 0 0

Writer 54023: writing data

Writer 54024: writing data
27 81 22 17
Writer 54023: writing data
27 81 22 17
Writer 54024: writing data


Reader 54021: reading data: 49 62 79 52
Reader 54020: reading data: 49 62 79 52

Reader 54022: reading data: 49 62 79 52
49 62 79 52
Writer 54023: writing data
49 62 79 52
Writer 54024: writing data


Reader 54021: reading data: 10 97 81 52
Reader 54020: reading data: 10 97 81 52

Reader 54022: reading data: 10 97 81 52


Reader 54021: reading data: 10 97 81 52
Reader 54020: reading data: 10 97 81 52

Reader 54022: reading data: 10 97 81 52
10 97 81 52
Writer 54023: writing data
10 97 81 52
Writer 54024: writing data



Reader 54021: reading data: 80 83 63 1
Reader 54020: reading data: 80 83 63 1
Reader 54022: reading data: 80 83 63 1
80 83 63 1
Writer 54023: writing data
80 83 63 1
Writer 54024: writing data



Reader 54020: reading data: 35 77 37 46
Reader 54021: reading data: 35 77 37 46
Reader 54022: reading data: 35 77 37 46
35 77 37 46
Writer 54023: writing data
35 77 37 46
Writer 54024: writing data
```

```
Reader 54022: reading data: 19 44 63 74
Reader 54021: reading data: 19 44 63 74
Reader 54020: reading data: 19 44 63 74
19 44 63 74
Writer 54023: writing data
19 44 63 74
Writer 54024: writing data
87 1 26 9
Writer 54023: writing data
87 1 26 9
Writer 54024: writing data
68 58 33 0
Writer 54023: writing data
68 58 33 0
Writer 54024: writing data


Reader 54020: reading data: 89 10 59 22
Reader 54022: reading data: 89 10 59 22

Reader 54021: reading data: 89 10 59 22
89 10 59 22
Writer 54023: writing data
89 10 59 22
Writer 54024: writing data
2 2 45 66 2 2 45 66

Reader 54020: reading data: 2 2 45 66
Reader 54022: reading data: 2 2 45 66

Reader 54021: reading data: 2 2 45 66



Reader 54020: reading data: 2 2 45 66
Reader 54022: reading data: 2 2 45 66
Reader 54021: reading data: 2 2 45 66
All readers and writers have finished execution
```

The End