Operating Systems

Question 1

1(a): UNIX Commands – Create directory, file, and view it

```
# Create a new directory called "my_directory"
mkdir my_directory

# Create a file (myfile.txt) inside the directory and write text into it
echo "Hello, CS3461!" > my_directory/myfile.txt

# Display the file content
cat my_directory/myfile.txt
```

Expected Output:

Hello, CS3461!

1(b): Shell Program – Generate Fibonacci Series

Save the following script (e.g., `fibonacci.sh`) and run it.

```
#!/bin/bash
read -p "Enter number of terms: " n
a=0
b=1
echo "Fibonacci Series:"
for (( i=0; i<n; i++ )); do
   echo -n "$a "
   fn=$((a + b))
   a=$b
   b=$fn
done
echo ""
```

Sample Run:

Enter number of terms: 7
Fibonacci Series:
0 1 1 2 3 5 8

1(c): C Program – FCFS CPU Scheduling (Average Waiting & Turnaround Time)

Save the following code as (for example) `fcfs.c` and compile with `gcc fcfs.c -o fcfs`:

```c
#include <stdio.h>
int main(){
   int n = 4;
   int arrival[] = {0, 2, 2, 3};
   int burst[]   = {5, 2, 8, 6};
   int comp[4], wait[4], tat[4];
   int time = 0;

   // FCFS scheduling (processes assumed to be already sorted by arrival time)
   for (int i = 0; i < n; i++) {
      if(time < arrival[i])
         time = arrival[i];
      time += burst[i];
      comp[i] = time;
      tat[i] = comp[i] - arrival[i];     // Turnaround Time
      wait[i] = tat[i] - burst[i];       // Waiting Time
   }

   float total_wait = 0, total_tat = 0;
   for (int i = 0; i < n; i++){
      total_wait += wait[i];
      total_tat  += tat[i];
   }

   printf("Average Waiting Time: %.2f\n", total_wait / n);
   printf("Average Turnaround Time: %.2f\n", total_tat / n);
   return 0;
}
```

Expected Output (approximate):

Average Waiting Time: 2.75
Average Turnaround Time: 10.00

Question 2

2(a): UNIX Commands – Word Count and List Directory

```
# Word count of a file named myfile.txt
wc myfile.txt

# List all files/directories in the current directory, with detailed info
ls -l
```

Expected Output:

The `wc` command prints line/word/byte counts (for example, `1 2 15 myfile.txt`) and `ls -l` lists directory details.

2(b): Shell Program – Even or Odd Checker

Save as `evenodd.sh`:

```
#!/bin/bash
read -p "Enter a number: " num
if (( num % 2 == 0 )); then
   echo "$num is even."
else
   echo "$num is odd."
fi
```

Sample Run:

```
Enter a number: 7
7 is odd.
```

2(c): C Program – Simple IPC Using a Pipe

Save as `ipc_pipe.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main(){
   int pipefd[2];
   pid_t pid;
   char write_msg[] = "Hello from parent";
   char read_msg[100];

   if(pipe(pipefd) == -1){
      perror("pipe");
      exit(1);
   }

   pid = fork();
   if(pid < 0){
      perror("fork");
      exit(1);
   }

   if(pid > 0){
```

```
    // Parent Process: write message
    close(pipefd[0]);
    write(pipefd[1], write_msg, strlen(write_msg)+1);
    close(pipefd[1]);
  }
  else{
    // Child Process: read message
    close(pipefd[1]);
    read(pipefd[0], read_msg, sizeof(read_msg));
    printf("Child read: %s\n", read_msg);
    close(pipefd[0]);
  }
  return 0;
}
```

Expected Output (child process prints):

Child read: Hello from parent

Question 3

3(a): C Program – Simple Demonstration of fork()

Save as `fork_demo.c`:

```
#include <stdio.h>
#include <unistd.h>
int main(){
  pid_t pid = fork();

  if(pid == 0){
    printf("This is the child process with PID %d\n", getpid());
  }
  else if(pid > 0){
    printf("This is the parent process with PID %d and child PID %d\n", getpid(), pid);
  }
  return 0;
}
```

Sample Output:

This is the parent process with PID 12345 and child PID 12346
This is the child process with PID 12346

3(b): Shell Program – Armstrong Number Checker

Save as `armstrong.sh`:

```
#!/bin/bash
read -p "Enter a number: " num
sum=0
temp=$num
n=${#num}  # number of digits
while [ $temp -gt 0 ]; do
   digit=$(( temp % 10 ))
   sum=$(( sum + digit ** n ))
   temp=$(( temp / 10 ))
done
if [ $sum -eq $num ]; then
   echo "$num is an Armstrong number."
else
   echo "$num is not an Armstrong number."
fi
```

Sample Run:

```
Enter a number: 153
153 is an Armstrong number.
```

3(c): C Program – Minimal Banker's Algorithm Example

Save as `banker.c`:

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

#define P 3  // number of processes
#define R 3  // number of resource types

bool isSafeState(int alloc[P][R], int max[P][R], int avail[R]){
   int need[P][R];
   for (int i = 0; i < P; i++)
      for (int j = 0; j < R; j++)
         need[i][j] = max[i][j] - alloc[i][j];

   bool finish[P] = {false};
   int work[R];
   for (int j = 0; j < R; j++)
      work[j] = avail[j];

   int count = 0;
   while(count < P) {
      bool found = false;
```

```
      for (int i = 0; i < P; i++) {
         if (!finish[i]) {
            bool possible = true;
            for (int j = 0; j < R; j++) {
               if (need[i][j] > work[j]) {
                  possible = false;
                  break;
               }
            }
            if(possible) {
               for (int j = 0; j < R; j++)
                  work[j] += alloc[i][j];
               finish[i] = true;
               found = true;
               count++;
            }
         }
      }
      if(!found) break;
   }
   return (count == P);
}

int main(){
   int alloc[P][R] = { {0,1,0}, {2,0,0}, {3,0,2} };
   int max[P][R]   = { {7,5,3}, {3,2,2}, {9,0,2} };
   int avail[R]    = {3,3,2};

   if(isSafeState(alloc, max, avail))
      printf("The system is in a safe state.\n");
   else
      printf("The system is NOT in a safe state.\n");

   return 0;
}
```

Expected Output:

The system is in a safe state.

Question 4

4(a): UNIX Commands – Remove a File and a Directory

```
# Remove a file
rm my_directory/myfile.txt
```

```
# Remove a directory (must be empty)
rmdir my_directory
```

Expected Outcome:

No message is printed if the operations succeed.

4(b): Shell Program – Prime Number Checker

Save as `prime.sh`:

```
#!/bin/bash
read -p "Enter a number: " num
if (( num <= 1 )); then
  echo "$num is not prime."
  exit 0
fi
flag=0
for (( i = 2; i * i <= num; i++ )); do
  if (( num % i == 0 )); then
    flag=1
    break
  fi
done
if [ $flag -eq 0 ]; then
   echo "$num is prime."
else
   echo "$num is not prime."
fi
```

Sample Run:

```
Enter a number: 17
17 is prime.
```

4(c): C Program – First Fit Memory Allocation

Save as `first_fit.c`:

```
#include <stdio.h>
int main(){
   int process[] = {80, 50, 30, 40};
   int block[]   = {20, 100, 200, 10};
   int p = 4, b = 4;
   int allocation[4];
```

```
    for(int i = 0; i < p; i++){
        allocation[i] = -1;
        for (int j = 0; j < b; j++){
            if(block[j] >= process[i]){
                allocation[i] = j;
                block[j] -= process[i];
                break;
            }
        }
    }

    for(int i = 0; i < p; i++){
        if(allocation[i] != -1)
            printf("Process %d allocated to Block %d\n", i, allocation[i]);
        else
            printf("Process %d not allocated\n", i);
    }
    return 0;
}
```

Expected Output (one possible allocation):

Process 0 allocated to Block 1
Process 1 allocated to Block 2
Process 2 allocated to Block 2
Process 3 allocated to Block 2

Question 5

5(a): UNIX Command – List Files and Directories

ls -al

Expected Output:

A detailed list of files and directories in the current folder.

5(b): Shell Program – Calculate Area of a Circle

Save as `circle_area.sh`:

```
#!/bin/bash
read -p "Enter radius: " r
pi=3.14159
area=$(echo "$pi * $r * $r" | bc -l)
echo "Area of the circle: $area"
```

Sample Run:

Enter radius: 5
Area of the circle: 78.53975

(5(c) is essentially the same First Fit Allocation shown in Question 4(c).)

Question 6

6(a): C Program – Demonstrate getpid() System Call

Save as `getpid_demo.c`:

```
#include <stdio.h>
#include <unistd.h>
int main(){
  printf("Process ID: %d\n", getpid());
  return 0;
}
```

Expected Output:

A printed line with your process ID, for example:

Process ID: 12345

6(b): Shell Program – Celsius to Fahrenheit Converter

Save as `celsius_to_fahrenheit.sh`:

```
#!/bin/bash
read -p "Enter temperature in Celsius: " c
f=$(echo "scale=2; ($c * 9/5) + 32" | bc)
echo "$c Celsius = $f Fahrenheit"
```

Sample Run:

Enter temperature in Celsius: 25
25 Celsius = 77.00 Fahrenheit

6(c): C Program – Basic Threading Example Using pthreads

Save as `thread_demo.c` and compile with `-pthread`:

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

void* print_message(void* arg){
  printf("Hello from thread!\n");
  return NULL;
}

int main(){
  pthread_t tid;
  pthread_create(&tid, NULL, print_message, NULL);
  pthread_join(tid, NULL);
  printf("Thread finished executing.\n");
  return 0;
}
```

Expected Output:

Hello from thread!
Thread finished executing.

Question 7

7(a): UNIX Commands – Display date, calendar, and file content

```
date         # shows current date and time
cal          # displays current month's calendar
cat filename  # displays contents of filename (replace with an actual file)
```

Expected Outcome:

Each command prints its corresponding information.

7(b): Shell Program – Calculate Triangle Area

Save as `triangle_area.sh`:

```
#!/bin/bash
read -p "Enter base: " b
read -p "Enter height: " h
area=$(echo "scale=2; 0.5 * $b * $h" | bc)
echo "Area of the triangle: $area"
```

Sample Run:

Enter base: 5

Enter height: 10
Area of the triangle: 25.00

7(c): C Program – Deadlock Detection (Simplified Example)

Save as `deadlock.c`:

```
#include <stdio.h>
#include <stdbool.h>
#define P 3
#define R 3

int main(){
   int alloc[P][R] = { {0, 1, 0}, {2, 0, 0}, {3, 0, 2} };
   int max[P][R]   = { {7, 5, 3}, {3, 2, 2}, {9, 0, 2} };
   int avail[R] = {3, 3, 2};
   int need[P][R];
   bool finish[P] = {false};

   for (int i = 0; i < P; i++)
      for (int j = 0; j < R; j++)
         need[i][j] = max[i][j] - alloc[i][j];

   int count = 0;
   while(count < P) {
      bool found = false;
      for(int i = 0; i < P; i++){
         if (!finish[i]){
            bool can_finish = true;
            for(int j = 0; j < R; j++){
               if(need[i][j] > avail[j]){
                  can_finish = false;
                  break;
               }
            }
            if(can_finish){
               for(int j = 0; j < R; j++){
                  avail[j] += alloc[i][j];
               }
               finish[i] = true;
               found = true;
               count++;
            }
         }
      }
      if(!found) break;
   }
```

```
    bool deadlock = false;
    for(int i = 0; i < P; i++){
        if(!finish[i]){
            deadlock = true;
            printf("Process %d is deadlocked.\n", i);
        }
    }
    if(!deadlock)
        printf("No deadlock detected.\n");

    return 0;
}
```

Expected Output:

No deadlock detected.

Question 8

8(a): Shell Program – Factorial Calculator

Save as `factorial.sh`:

```
#!/bin/bash
read -p "Enter a number: " n
fact=1
for (( i = 1; i <= n; i++ )); do
  fact=$((fact * i))
done
echo "Factorial of $n is $fact"
```

Sample Run:

Enter a number: 5
Factorial of 5 is 120

8(b): C Program – FIFO Page Replacement for a Given Reference String

Save as `fifo_page.c`:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 18

int main(){
```

```
    int ref[SIZE] = {6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 0, 3, 2, 1, 2, 0};
    int frames[3] = {-1, -1, -1};  // initialize 3 frames
    int faults = 0, curr = 0;

    for(int i = 0; i < SIZE; i++){
        int page = ref[i];
        int found = 0;
        for (int j = 0; j < 3; j++){
            if(frames[j] == page){
                found = 1;
                break;
            }
        }
        if(!found){
            // Replace using FIFO
            frames[curr] = page;
            curr = (curr + 1) % 3;
            faults++;
        }
    }
    printf("Total page faults = %d\n", faults);
    return 0;
}
```

Expected Output:

Total page faults = 12

Question 9

9(a): Shell Program – Display First 10 Natural Numbers

Save as `natural_numbers.sh`:

```
#!/bin/bash
echo "First 10 natural numbers:"
for i in {1..10}; do
  echo -n "$i "
done
echo ""
```

Expected Output:

First 10 natural numbers:
1 2 3 4 5 6 7 8 9 10

9(b): C Program – Two-Level Directory Simulation

Save as `two_level_directory.c`:

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

typedef struct {
   char subdir[50];
} Directory;

int main(){
   // Simulate a main directory with a few subdirectories
   Directory directories[3];
   strcpy(directories[0].subdir, "Documents");
   strcpy(directories[1].subdir, "Pictures");
   strcpy(directories[2].subdir, "Music");

   printf("Two-Level Directory Structure:\n");
   for(int i = 0; i < 3; i++){
      printf("MainDirectory/%s\n", directories[i].subdir);
   }
   return 0;
}
```

Expected Output:

Two-Level Directory Structure:
MainDirectory/Documents
MainDirectory/Pictures
MainDirectory/Music

Question 10

10(a): Shell Program – Display Date with Time

Save as `show_date.sh`:

```
#!/bin/bash
echo "Current date and time: $(date)"
```

Expected Output:

The current date and time, for example:

Current date and time: Wed May 28 19:42:00 IST 2025

10(b): C Program – Disk Scheduling using FCFS

Save as `disk_fcfs.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(){
   int requests[] = {98, 183, 41, 122, 14, 124, 65, 67};
   int n = sizeof(requests) / sizeof(requests[0]);
   int head = 53;
   int movement = 0;

   for(int i = 0; i < n; i++){
      movement += abs(head - requests[i]);
      head = requests[i];
   }

   printf("Total head movement = %d cylinders\n", movement);
   return 0;
}
```

Expected Calculation:

For the given sequence, the total head movement comes out to be 632 cylinders.

Sample Output:

Total head movement = 632 cylinders

Question 11

11(a): Shell Program – Fibonacci Series (Again)

(Same as Question 1(b); you may reuse the earlier script.)

11(b): C Program – SJF CPU Scheduling (Average Waiting & Turnaround Time)

Save as `sjf.c`:

```
#include <stdio.h>
int main(){
   int n = 4;
   int arrival[] = {0, 1, 2, 3};
```

```
    int burst[]   = {8, 2, 4, 4};
    int comp[4], wait[4], tat[4];
    int time = 0, finished = 0;
    int done[4] = {0};

    // Non-preemptive SJF Scheduling
    while(finished < n) {
        int idx = -1, min_burst = 10000;
        for(int i = 0; i < n; i++){
            if(!done[i] && arrival[i] <= time && burst[i] < min_burst){
                min_burst = burst[i];
                idx = i;
            }
        }
        if(idx == -1){
            time++;
            continue;
        }
        time += burst[idx];
        comp[idx] = time;
        tat[idx] = comp[idx] - arrival[idx];
        wait[idx] = tat[idx] - burst[idx];
        done[idx] = 1;
        finished++;
    }

    float total_wait = 0, total_tat = 0;
    for(int i = 0; i < n; i++){
        total_wait += wait[i];
        total_tat  += tat[i];
    }

    printf("Average Waiting Time: %.2f\n", total_wait / n);
    printf("Average Turnaround Time: %.2f\n", total_tat / n);
    return 0;
}
```

Expected Calculations (approximate):

- Average Waiting Time ≈ 6.50
- Average Turnaround Time ≈ 11.00

Sample Output:

Average Waiting Time: 6.50
Average Turnaround Time: 11.00

Question 12

12(a): Shell Program – Check if a Number is Positive or Negative

Save as `pos_neg.sh`:

```
#!/bin/bash
read -p "Enter a number: " num
if [ $num -gt 0 ]; then
   echo "$num is positive."
elif [ $num -lt 0 ]; then
   echo "$num is negative."
else
   echo "The number is zero."
fi
```

Sample Run:

Enter a number: -5
-5 is negative.

12(b) – C Program: Semaphore Using POSIX

Save as `semaphore_demo.c`:

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t sem;
void* tfunc(void *arg){
   int id = *(int*)arg;
   sem_wait(&sem);
   printf("Thread %d in CS\n", id);
   sleep(1);
   printf("Thread %d out CS\n", id);
   sem_post(&sem);
   return NULL;
}
int main(){
   pthread_t t[3]; int id[3] = {1,2,3};
   sem_init(&sem, 0, 1);
   for (int i = 0; i < 3; i++) pthread_create(&t[i], NULL, tfunc, &id[i]);
   for (int i = 0; i < 3; i++) pthread_join(t[i], NULL);
   sem_destroy(&sem);
   return 0;
```

```
}
```

Expected Output (order may vary):

Thread 1 in CS
Thread 1 out CS
Thread 2 in CS
Thread 2 out CS
Thread 3 in CS
Thread 3 out CS

Question 13:

13(a) – Shell Program: Armstrong Number Check

Save as `armstrong.sh`:

```
#!/bin/bash
read -p "Enter number: " n; orig=$n; s=0; d=${#n}
while [ $n -gt 0 ]; do r=$(( n % 10 )); s=$(( s + r**d )); n=$(( n / 10 )); done
[ $s -eq $orig ] && echo "Armstrong" || echo "Not Armstrong"
```

Sample Run:

$ ./armstrong.sh
Enter number: 153
Armstrong

13(b) – C Program: Optimal Page Replacement

Save as `optimal_page.c`:

```
#include <stdio.h>
#define NUM 20
#define FRAMES 3
int main(){
   int ref[NUM] = {6,1,1,2,0,3,4,6,0,2,1,2,1,2,0,3,2,1,4,0}, f[FRAMES] = {-1,-1,-1}, faults = 0;
   for (int i = 0; i < NUM; i++){
      int page = ref[i], found = 0;
      for (int j = 0; j < FRAMES; j++){
         if (f[j] == page){ found = 1; break; }
      }
      if(found) continue;
      faults++;
      int placed = 0;
      for (int j = 0; j < FRAMES; j++){
```

```
            if(f[j] == -1){ f[j] = page; placed = 1; break; }
        }
        if(placed) continue;
        int maxDist = -1, rep;
        for (int j = 0; j < FRAMES; j++){
            int k;
            for(k = i+1; k < NUM; k++){
                if(ref[k] == f[j])
                    break;
            }
            int dist = (k == NUM) ? 1000 : k;
            if(dist > maxDist){ maxDist = dist; rep = j; }
        }
        f[rep] = page;
    }
    printf("Faults: %d\n", faults);
    return 0;
}
```

Expected Output:

Faults: 11

Question 14:

14(a) – Shell Program: Area & Circumference of a Circle

Save as `circle_metrics.sh`:

```
#!/bin/bash
read -p "Radius: " r
pi=3.14159
area=$(echo "$pi * $r * $r" | bc -l)
circ=$(echo "2 * $pi * $r" | bc -l)
echo "Area: $area, Circumference: $circ"
```

Sample Run:

```
$ ./circle_metrics.sh
Radius: 5
Area: 78.53975, Circumference: 31.41590
```

14(b) – C Program: Best Fit Allocation

Save as `best_fit.c`:

```
#include <stdio.h>
#define P 4
#define B 5
int main(){
   int proc[P] = {40, 10, 30, 60}, block[B] = {100, 50, 30, 120, 35}, alloc[P], used[B] = {0};
   for (int i = 0; i < P; i++){
      alloc[i] = -1;
      int best = -1;
      for (int j = 0; j < B; j++){
         if (!used[j] && block[j] >= proc[i] && (best == -1 || block[j] < block[best]))
            best = j;
      }
      if (best != -1){ alloc[i] = best; used[best] = 1; }
   }
   for (int i = 0; i < P; i++){
      if(alloc[i] != -1)
         printf("P%d -> B%d\n", i, alloc[i] + 1);
      else
         printf("P%d not allocated\n", i);
   }
   return 0;
}
```

Expected Output (allocation may vary based on best-fit selection):

P0 -> B2
P1 -> B3
P2 -> B5
P3 -> B1

Question 15:

15(a) – Shell Program: Area of a Rectangle

```
#!/bin/bash
read -p "Enter length: " l
read -p "Enter breadth: " b
echo "Area: $(( l * b ))"
```

Sample Run:

$ ./rectangle_area.sh
Enter length: 5
Enter breadth: 3
Area: 15

15(b) – C Program: SSTF Disk Scheduling

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

int main(){
   int req[10] = {4, 34, 10, 7, 19, 73, 2, 15, 6, 20};
   int n = 10, head = 50, move = 0;
   bool done[10] = {0};

   for (int i = 0; i < n; i++){
      int idx = -1, min = INT_MAX;
      for (int j = 0; j < n; j++){
         if (!done[j]){
            int d = abs(req[j] - head);
            if(d < min){
               min = d;
               idx = j;
            }
         }
      }
      if(idx == -1) break;
      done[idx] = true;
      move += min;
      head = req[idx];
   }
   printf("Total head movement: %d\n", move);
   return 0;
}
```

Expected Output (using the fixed request queue):

Total head movement: 119

Question 16:

16(a) – Shell Program: Display First 10 Even Numbers

```
#!/bin/bash
for (( i = 2; i <= 20; i += 2 )); do echo -n "$i "; done; echo
```

Expected Output:

2 4 6 8 10 12 14 16 18 20

16(b) – C Program: Sequential File Allocation

```
#include <stdio.h>
int main(){
   int start, len;
   printf("Enter starting block and length: ");
   scanf("%d %d", &start, &len);
   printf("File allocated from block %d to %d\n", start, start + len - 1);
   return 0;
}
```

Sample Run:

Enter starting block and length: 10 5
File allocated from block 10 to 14

Question 17:

17(a) – Shell Program: Greatest of Two Numbers

```
#!/bin/bash
read -p "Enter two numbers: " a b
(( a >= b )) && echo "$a is greatest" || echo "$b is greatest"
```

Sample Run:

$ ./greatest.sh
Enter two numbers: 27 15
27 is greatest

17(b) – C Program: Hierarchical Directory Simulation

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

struct Dir {
   char name[20];
   struct Dir *sub[5];
   int count;
};

int main(){
   struct Dir root, d1, d2;
   strcpy(root.name, "root");
   strcpy(d1.name, "Docs");
```

```
    strcpy(d2.name, "Pics");
    root.sub[0] = &d1;
    root.sub[1] = &d2;
    root.count = 2;
    printf("%s\n", root.name);
    for (int i = 0; i < root.count; i++)
        printf("  |-- %s\n", root.sub[i]->name);
    return 0;
}
```

Expected Output:

```
root
  |-- Docs
  |-- Pics
```

Question 18:

18(a) – Shell Program: Squares of the First 10 Natural Numbers

```
#!/bin/bash
for i in {1..10}; do
  echo "Square of $i is $(( i * i ))"
done
```

Sample Run & Output:

```
$ ./squares.sh
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
Square of 5 is 25
Square of 6 is 36
Square of 7 is 49
Square of 8 is 64
Square of 9 is 81
Square of 10 is 100
```

18(b) – C Program: Paging Simulation

This program demonstrates a simple paging concept by mapping a logical address to a physical address using a fixed page table.

Assumptions:

– Page size is 512 bytes.

– The page table covers pages 0–4.

```c
#include <stdio.h>
int main(){
   int page_table[5] = {2, 4, 1, 3, 0};  // Mapping: Page number -> Frame number
   int page_size = 512;
   int logical_address;

   printf("Enter logical address: ");
   scanf("%d", &logical_address);

   int page_num = logical_address / page_size;
   int offset = logical_address % page_size;

   if(page_num < 5){
      int frame = page_table[page_num];
      int physical_address = frame * page_size + offset;
      printf("Logical Address: %d -> Page: %d, Offset: %d -> Physical Address: %d\n",
           logical_address, page_num, offset, physical_address);
   }
   else {
      printf("Invalid logical address: page number out of range.\n");
   }
   return 0;
}
```

Sample Run & Output:

(Assume the user enters 1025)
Enter logical address: 1025
Logical Address: 1025 -> Page: 2, Offset: 1 -> Physical Address: 513

Question 19:

19(a) – Shell Program: Celsius to Fahrenheit Conversion

```
#!/bin/bash
read -p "Enter temperature in Celsius: " c
f=$(echo "scale=2; ($c*9/5)+32" | bc)
echo "$c Celsius = $f Fahrenheit"
```

Sample Run & Output:

$ ./celsius_to_fahrenheit.sh
Enter temperature in Celsius: 25
25 Celsius = 77.00 Fahrenheit

19(b) – C Program: Demonstrate wait() and exit()

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main(){
   pid_t pid = fork();
   if(pid < 0){
      perror("fork");
      exit(1);
   }
   if(pid == 0) {
      // Child process
      printf("Child process executing...\n");
      exit(5);
   } else {
      int status;
      wait(&status);
      printf("Parent: Child terminated with exit code %d\n", WEXITSTATUS(status));
   }
   return 0;
}
```

Sample Run & Output:

Child process executing...
Parent: Child terminated with exit code 5

Question 20:

20(a) – C Program: Banker's Algorithm

This minimal Banker's Algorithm simulation checks if the system is in a safe state and prints the safe sequence.
Assumptions and data:
– 5 processes and 3 resource types
– Sample allocation, maximum, and available arrays are used for demonstration.

```
#include <stdio.h>
#include <stdbool.h>
#define P 5
#define R 3

int main(){
   int alloc[P][R] = { {0,1,0}, {2,0,0}, {3,0,2}, {2,1,1}, {0,0,2} };
```

```
int max[P][R]   = { {7,5,3}, {3,2,2}, {9,0,2}, {2,2,2}, {4,3,3} };
int avail[R]    = {3,3,2};
int need[P][R];
bool finish[P] = {0};
int safeSeq[P];
int count = 0;

// Calculate need matrix = max - alloc
for(int i = 0; i < P; i++){
   for(int j = 0; j < R; j++){
      need[i][j] = max[i][j] - alloc[i][j];
   }
}

while(count < P){
   bool found = false;
   for(int i = 0; i < P; i++){
      if(!finish[i]){
         bool possible = true;
         for(int j = 0; j < R; j++){
            if(need[i][j] > avail[j]){
               possible = false;
               break;
            }
         }
         if(possible){
            for(int j = 0; j < R; j++){
               avail[j] += alloc[i][j];
            }
            safeSeq[count++] = i;
            finish[i] = true;
            found = true;
         }
      }
   }
   if(!found)
      break;
}

if(count == P){
   printf("System is in safe state.\nSafe Sequence: ");
   for(int i = 0; i < P; i++){
      printf("P%d ", safeSeq[i]);
   }
   printf("\n");
}
else {
   printf("System is not in safe state.\n");
```

```
    }

    return 0;
}
```

Expected Output:

System is in safe state.
Safe Sequence: P1 P3 P0 P2 P4

UNIX Commands:

For Shell Scripts

1.Create file:

vim filename.sh

2.Make it executable:

chmod +x filename.sh

3.Run the script:

./filename.sh

For C Programs

1.Create file:

vim filename.c

2.Compile the code:

gcc filename.c -o filename

3.Run the executable:

./filename
