OPERATING SYSTEMS (CS F372)
ASSIGNMENT

WEIGHTAGE – 10%, MARKS – 20, TYPE – OPEN BOOK

MARKS DIVISION – PROBLEM 1: 12 MARKS, PROBLEM: 4 MARKS, VIVA (TO BE CONDUCTED DURING DEMO): 4 MARKS

PROBLEM 1: [12 MARKS]

Write a C program (that can be executed on a Linux platform) to implement the following scenario. A parent process creates 3 child processes – **Child1**, **Child2** and **Child3**. The parent process creates a pipe, **Pipe1**, to communicate with **Child1**. Parent process takes a number **n1 (0 < n1 < 50)** as input from the user. The parent process sends **n1** to **Child1** via **Pipe1**. **Child1** reads this number from **Pipe1** and prints **n1** on the console. **Child1** also prints **n1** terms of the Fibonacci series on the console. For eg., If **n1** = 5, then **Child1** prints 1 1 2 3 5 (considering the first 2 terms of the Fibonacci series to 1 and 1).

The parent process creates 3 pipes, **Pipe2**, **Pipe3** and **Pipe4**, to communicate with **Child2**. Parent process takes an integer number **n2 (0 < n2 ≤ 10)** as input from the user and sends **n2** to **Child2** via **Pipe2**. Parent process takes another integer number **n3 (0 < n3 ≤ 1000)** as input from user and sends **n3** to **Child2** via **Pipe3**. Parent process also takes an array, **arr**, as input from user such that **arr** contains **n3** number of elements and sends the array to **Child2** via **Pipe4**. Assume that **arr** contains only integers in the range [1, 1000] and there are no duplicate elements in **arr**. It is not necessary to that the elements in **arr** will be present in sorted order. After reading **n2**, **n3** and **arr** respectively from **Pipe2**, **Pipe3** and **Pipe4**, **Child2** prints them on the console. **Child2** also creates **n2** number of threads, **Thread1**, **Thread2**, ..., **Threadn2**, and divides the array **arr** among the created threads. Assume that **n3** is exactly divisible by **n2** so that each thread works on the same number of elements. When **Child2** creates a thread, **Child2** will inform that thread the lower and upper bounds of the sub-portion of **arr** on which that thread is supposed to operate as well as the elements of **arr** present between the specified upper and lower bounds. Also, when **Child2** creates a thread, **Child2** assigns an index number to that thread (i.e., the first thread is assigned an index of 1, the second thread is assigned an index of 2 and so on) and communicates this index number to the thread during creation. ***Note that this index number is not the same as thread ID. Also, note that you cannot use a global data structure or any interprocess communication mechanism to facilitate communication between Child2 and the threads. Also, arr cannot be declared as a global array***. However, you are free to use any kind of data structure local to either **Child2** or the threads. Each thread computes the maximum and minimum elements from the assigned sub-portion of **arr**, prints them on the console and communicates their respective computed maximum and minimum elements to **Child2**. **Child2** then computes the overall maximum and minimum element of **arr** from the values reported by the individual threads and prints them on the console. ***Note that Child2 should not compute the maximum and minimum elements directly from arr. Also, note that you are not allowed to write multiple functions for defining the task to be carried out by the different threads. Moreover, you are not allowed to sort the elements of arr.*** For eg., If **n2**

= 3, **Child2** will create 3 threads – **Thread1**, **Thread2** and **Thread3**. If **n3** = 9 and **arr** = {1, 2, 4, 6, 3, 9, 7, 10, 5}, then **Thread1** operates between **arr[0]** and **arr[2]**, **Thread2** operates between **arr[3]** and **arr[5]** and **Thread3** operates between **arr[6]** and **arr[8]**. **Thread1** reports 1 as minimum and 4 as maximum to **Child2**. **Thread2** reports 3 as minimum and 9 as maximum to **Child2**. **Thread3** reports 5 as minimum and 10 as maximum to **Child2**. **Child2** finally computes 1 as the overall minimum of **arr** and 10 as the overall maximum of **arr**. ***Note that Child2 waits for all 3 threads to finish execution. You cannot use sleep() to make Child2 wait for the threads.***

Parent process creates **Pipe5** to communicate with **Child3**. Parent process takes a number **n4** as input from user and sends **n4** to **Child3** via **Pipe5**. **Child3** prints the value of **n4** on the console, computes the factorial of **n4** and prints it. For eg., If **n4** = 5, **Child3** computes the factorial as 120.

Note that the parent process waits for all the 3 child processes to finish execution. You cannot make the parent process wait using sleep(). The parent process exits by printing the message "**Goodbye**" on the console. ***Note that other than the 5 pipes mentioned above, you are not allowed to use any other pipe or any other interprocess communication mechanism. Also, note that you are not allowed to hard-code the values of n1, n2, n3 and n4 and the elements of arr. In case, you do not adhere to any of the mentioned constraints, marks will be deducted.***

A sample of the expected output is given below.

```
Enter to parent the no. of terms of Fibonacci series: 5

Child1 received 5

The Fibonacci Series printed by Child1 is: 1 1 2 3 5

Enter to parent the no. of child threads to be created: 3

Child2 received 3 as no. of threads

Enter to parent number of array elements for Child2: 9

Child2 received 9 as number of array elements

Enter to parent the array elements for Child2: 1 2 4 6 3 9 7
10 5

Lower bound for Thread1 is 0
Upper bound for Thread1 is 2
Computed by Thread1: max = 4, min = 1

Lower bound for Thread2 is 3
Upper bound for Thread2 is 5
Computed by Thread2: max = 9, min = 3
```

```
Lower bound for Thread3 is 6
Upper bound for Thread3 is 8
Computed by Thread3: max = 10, min = 5

Calculated by Child2: max = 10, min = 1

Enter the no. for factorial computation in parent: 5

Child3 received 5 for factorial computation

Factorial computed by Child3 is 120

Goodbye
```

PROBLEM 2: [4 MARKS]

Write a C program (that can be executed on a Linux platform) to implement the following scenario. Consider a scheduling algorithm known as the **Prioritized Longest Remaining Time First (PLRTF) algorithm**. PLRTF is a **preemptive** scheduling algorithm. This algorithm considers 2 weight values – **`w1`** and **`w2`**. **`w1`** is associated with CPU burst of a process (total or remaining) and **`w2`** is associated with the priority of the process. Assume **`0 < w1, w2 < 1`**. There is no relation between **`w1`** and **`w2`** implying that **`w1 < w2`** or **`w1 > w2`** or **`w1 = w2`**. For this scenario, assume that higher integer values indicate higher process priorities. Suppose that **`T`** denotes the CPU burst cycle (total or remaining) of a process and **`Pr`** denotes the priority of the process, then for this algorithm, the CPU scheduler picks the process having the maximum value of (**`w1*T + w2*Pr`**) and allocates the CPU to it. In case of a tie, the process having a lower value of arrival time is selected. If while executing a process **`P`**, another process **`Q`** arrives at the ready queue and the weighted sum of total CPU burst cycle and priority of **`Q`** is higher than the weighted sum of the remaining CPU burst cycle and priority of **`P`**, then **`Q`** will preempt **`P`**. Otherwise, **`P`** will continue execution until it terminates or is pre-empted by any other newly arrived process. Write a C program to implement the above scheduling algorithm. Your program should take as inputs from the user the number of processes, arrival time, CPU burst cycle and priority of each process and the values of **`w1`** and **`w2`**. Your program should print the individual waiting time of each process, the average waiting time, the order in which the processes are executed and the time of termination of each process. ***Note that you are not allowed to hard-code any of the values that are mentioned to be provided as user inputs. If you do not adhere to this constraint, marks will be deducted.***

A sample run describing the inputs and outputs are mentioned below.

```
Enter the number of processes: 3

Enter arrival time of P1: 0
Enter CPU burst cycle of P1: 10
Enter priority of P1: 2

Enter arrival time of P2: 2
Enter CPU burst cycle of P2: 5
Enter priority of P2: 1
```

```
Enter arrival time of P3: 5
Enter CPU burst cycle of P3: 12
Enter priority of P3: 1

Enter value of w1: 0.1
Enter value of w2: 0.1

Waiting time of P1: 12
Waiting time of P2: 20
Waiting time of P3: 0

Average waiting time: 10.66

Process Execution Order: P1->P3->P1->P2

P1 finished at: 22
P2 finished at: 27
P3 finished at: 17
```

****