Adhesh Reghu Kumar
COE18B001

Operating Systems Lab

Lab Assignment - 5

1. Implement using pipes : parent sets up a string which is read by child, reversed there and read back to the parent.

Logic Used

In this program we are required to reverse a string using pipes between two processes created using fork().
For this program, we require two pipes (or pipe file descriptors) as we have to perform two read/write operations between the two processes. The first pipe is responsible for passing the string setup by the parent process to the child process. The second pipe is responsible for passing the reversed string from the child side to the parent side. In each process, we also close the unused pipe ends.
A `reverse()` function produces the reversed string.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#define MAX 1024

void reverse(char str[], char str_rev[]);

int main()
{
    // Define the file descriptor for the pipes
    int pipe_fd_1[2];
    int pipe_fd_2[2];

    // Create the pipes
    if(pipe(pipe_fd_1) == -1)
```

```
    {
        fprintf(stderr, "Pipe Failed\n");
        return 1;
    }

    if(pipe(pipe_fd_2) == -1)
    {
        fprintf(stderr, "Pipe 2 Failed\n");
    }

    // Fork to create process
    pid_t pid = fork();

    if(pid > 0)
    {
        // Within parent process
        char str[MAX];
        scanf("%[^\n]",str);

        // Close the read side of the first pipe & write side of second pipe
        close(pipe_fd_1[0]);
        close(pipe_fd_2[1]);

        // send the string to the child process
        write(pipe_fd_1[1],str,strlen(str)+1);

        // read reversed string from parent
        char str_rev[MAX];
        read(pipe_fd_2[0],str_rev,strlen(str)+1);

        // print the reversed string
        printf("Reversed string is:\n%s\n",str_rev);
    }
    else if(pid == 0)
    {
        // Within child process
        char str[MAX];

        // Close the write side of the first pipe & read side of second pipe
        close(pipe_fd_1[1]);
        close(pipe_fd_2[0]);

        // read the string from the parent
```

```
        read(pipe_fd_1[0],str,MAX);
        // reverse string and write to parent
        char str_rev[MAX];
        reverse(str,str_rev);
        write(pipe_fd_2[1],str_rev,strlen(str)+1);

        exit(0);
    }
    else
    {
        fprintf(stderr, "Fork Failed\n");
        return 1;
    }

    return 0;
}

void reverse(char str[], char str_rev[])
{
    int len = strlen(str),i;
    for(i=len-1;i>=0;--i)
        str_rev[len-1-i] = str[i];
    str_rev[len] = '\0';
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 1_reverse.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
ahdhehshdf
Reversed string is:
fdhshehdha
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
asdjklasjdlkajsdlkjads
Reversed string is:
sdajkldsjakldjsalkjdsa
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

2. Implement using pipes : parent sets up string 1 and child sets up string 2. String 2 is concatenated to string 1 at the parent end and is read back at the child end.

Logic Used

In this program we are required to set up two strings - one at the parent end and one at the child end. We then pass string 2 from the child process to the parent process using a pipe. At the parent end we concat the string using a custom function - concat() which produces the concatenated result of two input strings. After obtaining the concatenated string at the parent process, we write the process to the child using another pipe. In the child process, we finally display the concatenated string.

Total Number of pipes used = 2.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#define MAX 1024

void concat(char str_1[], char str_2[], char concat_str[]);

int main()
{
    // Define the file descriptor for the pipes
    int pipe_fd_1[2];
    int pipe_fd_2[2];

    // Create the pipes
    if(pipe(pipe_fd_1) == -1)
    {
       fprintf(stderr, "Pipe Failed\n");
       return 1;
    }
```

```
    if(pipe(pipe_fd_2) == -1)
    {
       fprintf(stderr, "Pipe 2 Failed\n");
    }

    // Fork to create process
    pid_t pid = fork();

    if(pid > 0)
    {
       // Within parent process
       char str_1[MAX];
       scanf("%[^\n]",str_1);

       char str_2[MAX];

       // Close the write side of the first pipe & read side of second pipe
       close(pipe_fd_1[1]);
       close(pipe_fd_2[0]);

       // read string 2 from the child
       read(pipe_fd_1[0],str_2,MAX);

       // concat strings.
       char concat_str[MAX];
       concat(str_1,str_2,concat_str);

       // write the concatenated string to child
       write(pipe_fd_2[1],concat_str,strlen(concat_str)+1);

    }
    else if(pid == 0)
    {
       // Within child process
       char str[MAX];
       scanf("%[^\n]",str);

       // Close the read side of the first pipe & write side of second pipe
       close(pipe_fd_1[0]);
       close(pipe_fd_2[1]);

       // write the string to the parent
```

```
        write(pipe_fd_1[1],str, strlen(str)+1);

        // read the concatenated string from the parent
        char concat_str[MAX];
        read(pipe_fd_2[0],concat_str,MAX);

        printf("Concatenated string is:\n%s\n",concat_str);

        exit(0);
    }
    else
    {
        fprintf(stderr, "Fork Failed\n");
        return 1;
    }

    wait(NULL);
    return 0;
}

void concat(char str_1[], char str_2[], char concat_str[])
{
    int len_1 = strlen(str_1),len_2 = strlen(str_2),i=0;
    for(i=0;i<len_1;++i)
        concat_str[i] = str_1[i];

    for(;i<len_1+len_2;++i)
        concat_str[i] = str_2[i-len_1];

    concat_str[i] = '\0';
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 2_concat.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
asdasd
fgdgdfg
Concatenated string is:
asdasdfgdgdfg
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
asjdlkasjdlkajsldk
ioweioiwoeiwoew
Concatenated string is:
asjdlkasjdlkajsldkioweioiwoeiwoew
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

3. Implement using pipes : substring generation at child end of a string setup at the parent end.

Logic Used

In this program we are required to generate a substring of a given string using pipes between two processes created using fork().
In this program, we first set up a string in the parent process. We then pass the string to the child process using a pipe. In the child process, we accept the start position and the length of the substring as input. Using substring() (custom function) we generate the substring satisfying the input conditions. Once we generate the substring, we print it in the child process.

Total number of pipes used = 1.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#define MAX 1024

void substring(char str[], char substr[], int start, int len);

int main()
{
    // Define the file descriptor for the pipes
    int pipe_fd[2];
```

```
    // Create the pipes
    if(pipe(pipe_fd) == -1)
    {
       fprintf(stderr, "Pipe Failed\n");
       return 1;
    }
    // Fork to create process
    pid_t pid = fork();

    if(pid > 0)
    {
       // Within parent process
       printf("Enter the string.\n");
       char str[MAX];
       scanf("%[^\n]",str);

       // Close the read side of the pipe
       close(pipe_fd[0]);

       // send the string to the child process
       write(pipe_fd[1],str,strlen(str)+1);

    }
    else if(pid == 0)
    {
       // Within child process
       char str[MAX];

       // Close the write side of the pipe
       close(pipe_fd[1]);

       // read the string from the parent
       read(pipe_fd[0],str,MAX);

       // get substring arguments
       int start,len;
       printf("Enter the start position.\n");
       scanf("%d",&start);
       if(start >= strlen(str))
       {
             printf("Invalid start position.\n");
             exit(0);
       }
```

```
        printf("Enter the length of substring.\n");
        scanf("%d",&len);

        // generate substring
        char substr[MAX];
        substring(str,substr,start,len);
        printf("Substring is:\n%s\n",substr);

        exit(0);
    }
    else
    {
        fprintf(stderr, "Fork Failed\n");
        return 1;
    }

    wait(NULL);
    return 0;
}

void substring(char str[], char substr[], int start, int len)
{
    int i,pos=0;
    int str_len = strlen(str);
    for(i=start;i<start+len;++i)
    {
        if(i < str_len)
              substr[pos++] = str[i];
        else
              break;
    }

    substr[pos] = '\0';
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 3_substr.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
asdasdas
Enter the start position.
0
Enter the length of substring.
3
Substring is:
asd
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: When length of substring exceeds the available length - substring is generated only up to the end of string.

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
adsaffd
Enter the start position.
4
Enter the length of substring.
5
Substring is:
ffd
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: Invalid start position input given. Validation occurs and prints an error message.

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
asdasd
Enter the start position.
8
Invalid start position.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

4. Implement using pipes : string reversal and palindrome check.

Logic Used

In this program we are required to check if an input string is a palindrome or not using pipes between two processes created using fork().
In this program we set up a string in the parent process. We then pass the string to a child process using a pipe. In the child process, we check if the string read from the pipe is a palindrome or not using a custom function - `isPalindrome()`. We get an integer value indicating the palindrome status of the string from the function. We then print the result in the child process itself.

Total Number of pipes used = 1.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#define MAX 1024

int isPalindrome(char str[]);

int main()
{
    // Define the file descriptor for the pipes
    int pipe_fd[2];

    // Create the pipes
    if(pipe(pipe_fd) == -1)
    {
       fprintf(stderr, "Pipe Failed\n");
       return 1;
    }

    // Fork to create process
    pid_t pid = fork();
```

```
    if(pid > 0)
    {
       // Within parent process
       printf("Enter the string.\n");
       char str[MAX];
       scanf("%[^\n]",str);

       // Close the read side of the pipe
       close(pipe_fd[0]);

       // send the string to the child process
       write(pipe_fd[1],str,strlen(str)+1);

    }
    else if(pid == 0)
    {
       // Within child process
       char str[MAX];

       // Close the write side of the pipe
       close(pipe_fd[1]);

       // read the string from the parent
       read(pipe_fd[0],str,MAX);

       // Check if the string is a palindrome
       int status = isPalindrome(str);

       if(status == 1)
             printf("%s is a palindrome.\n",str);
       else
             printf("%s is NOT a palindrome.\n",str);

       exit(0);
    }
    else
    {
       fprintf(stderr, "Fork Failed\n");
       return 1;
    }
    return 0;
}
```

```c
int isPalindrome(char str[])
{
    int len = strlen(str), i, flag =0;

    for(i=0; i<len;++i)
       if(str[i] != str[len-i-1])
             return 0;

    return 1;
}
```

Output

Case: When input string is not a palindrome

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 4_palindrome.c
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
asdasfd
asdasfd is NOT a palindrome.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
hqkjwehkjqwe
hqkjwehkjqwe is NOT a palindrome.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: When input string is a palindrome

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
malayalam
malayalam is a palindrome.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the string.
racecar
racecar is a palindrome.
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

5. Implement using threads : Armstrong number generation up to a given range.

Logic Used

In this program we are supposed to generate Armstrong numbers up to a given range n using multi-threading.
In this program, we loop from 1 to n and check if the number is an Armstrong number (sum of cubes of digits == number) in the runner function of the thread. That means we create a new thread for each number in the loop. Therefore we create n threads in the entire program. After checking and printing the result we exit the thread.

Total number of threads created = n

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <string.h>
#include <pthread.h>
#define MAX 1024

void* runner(void *arg);

int main()
{
    // Get range.
    int n;
    printf("Enter the range.\n");
    scanf("%d",&n);
    printf("Armstrong numbers upto %d are:\n",n);

    // Loop to range.
    pthread_t tid[n];
    int i;
    for(i=0;i<=n;++i)
    {
       // Declare the argument to be passed to the thread
       int *arg = malloc(sizeof(*arg));
```

```
        *arg = i;

        // Create the thread
        pthread_create(&tid[i],NULL,&runner,arg);
    }

    return 0;
}

void* runner(void *arg)
{
    long int num = *((int *) arg);
    free(arg);

    // Perform the Armstrong Number Check
    long int cube_num =0, temp = num;
    int digit=0;
    while(temp!=0)
    {
        digit = temp%10;
        cube_num = cube_num + (digit * digit * digit);
        temp = temp/10;
    }

    if(cube_num == num)
        printf("%ld\n",num);

    pthread_exit(NULL);
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 5_armstrong.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the range.
400
Armstrong numbers upto 400 are:
0
1
153
370
371
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the range.
2000
Armstrong numbers upto 2000 are:
0
1
153
370
371
407
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

6. Implement using threads : Sort an input array in ascending and descending order in multi-threaded fashion.

Logic Used

In this program we are required to sort an input array to ascending order in one thread and in descending order in another thread.
Inorder to accomplish this, we need to pass the array, then length of the array and the order of sorting to the runner() function of the thread. So we create a structure, arrayWrapper that we pass to the runner function through pthread_create() function. The arrayWrapper structure looks like this:

```c
// allows the array to be passed by value (not by reference)
struct arrayWrapper
{
    int arr[MAX];
    int len;
    int order;
};
```

We first accept the input as command line arguments. Then after validating the inputs as integers using isInteger() function, we store the integer array into arr variable of an *arrayWrapper A*. We also set the len to the number of integer inputs.
Since we wish to pass the *arrayWrapper* by value, we create a copy *B* of *arrayWrapper A*. We then set the order of one structure to 0 (descending) and another structure to 1 (ascending) and pass them to different threads. The runner function sorts the *arr* of the structures according to the order given. After sorting, we exit the thread and join the main program. Once both threads have completed execution, we print the sorted array in the main program. Printing the result in the main program ensures that the order of printing is maintained.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <pthread.h>
#define MAX 1000

int isInteger(char* string);
void sortArray(int arr[],int n,int order);

// allows the array to be passed by value (not by reference)
struct arrayWrapper
{
    int arr[MAX];
    int len;
    int order;
};

void *runner(void *arg);

int main(int argc, char *argv[])
{

    struct arrayWrapper *A = (struct arrayWrapper *)malloc(sizeof(struct
arrayWrapper));


    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc;
       int n=0, i=1;
       for(;i<len;++i)
       {
             // Validation function. Warning if not integer.
             if(isInteger(argv[i]))
                   A->arr[n++] = atoi(argv[i]);
```

```
            else
                  printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
      }
      A->len = n;

      struct arrayWrapper B = *A;

      // Warning if entire argument list is invalid.s
      if(n == 0 )
            printf("The input list is invalid. Please enter an integer
array.\n");
      else
      {
            /* -- Sort the array in two arrays in two threads
                 Create two threads -- */
            pthread_t tid[2];
            A->order = 1;
            pthread_create(&tid[0],NULL,runner,A);
            B.order = 0;
            pthread_create(&tid[1],NULL,runner,(void *)&B);

            // // Join both threads
            pthread_join(tid[0],NULL);
            pthread_join(tid[1],NULL);

            // print the results
            int i=0;
            printf("The sorted (ascending) output is:\n");
            for(;i<n;++i)
                  printf("%d ",A->arr[i]);
            puts("");

            printf("The sorted (descending) output is:\n");
            i=0;
            for(;i<n;++i)
                  printf("%d ",B.arr[i]);
            puts("");
      }
   }
   return 0;
}
```

```
// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
       character = str[i];
       // In case of '-'' symbol, continue if not last character in string.
       if(character == 45 && i<strlen(str)-1)
             continue;
       // Else get the decimal value (-48) and check condition.
       character -= 48;
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}


// Runner function for thread
void *runner(void *arg)
{
    int *arr = ((struct arrayWrapper*)arg)->arr;
    int order = ((struct arrayWrapper*)arg)->order;
    int n = ((struct arrayWrapper*)arg)->len;

    /* -- Sort -- */

    // For swap purposes
    int temp;


    // Initiate Loops for simple Bubble Sort.
    int i=0;
    for(;i<n;++i)
    {
       int j=0;
       for(;j<n-i-1;++j)
       {
             if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
             {
```

```
                    // Swap arr[j] and arr[j+1]
                    temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }
        pthread_exit(NULL);
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 6_sort.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 1 24 2 3 4 1 2 4 5 6
The sorted (ascending) output is:
1 1 2 2 3 4 4 5 6 24
The sorted (descending) output is:
24 6 5 4 4 3 2 2 1 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 7 5 2 3 1 24 2 3 5 12 3 2 1 23
The sorted (ascending) output is:
1 1 2 2 2 3 3 3 5 5 7 12 23 24
The sorted (descending) output is:
24 23 12 7 5 5 3 3 3 2 2 2 1 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: With invalid inputs

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 6_sort.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out a d s 1 2 4 5 62 2 1
warning: 'a' is not an integer. will be ignored.
warning: 'd' is not an integer. will be ignored.
warning: 's' is not an integer. will be ignored.
The sorted (ascending) output is:
1 1 2 2 4 5 62
The sorted (descending) output is:
62 5 4 2 2 1 1
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

7. Implement using threads : Binary Search.

Logic Used

In this program we are required to implement binary search in a multi-threaded fashion. One idea would be to split the entire array into 4 partitions and perform binary search in each of the partitions and increment a global counter upon finding the key *x.* Each of the four partitions could perform binary search simultaneously with the help of threads.

For this program, we maintain several global variables:

- `arr[MAX]` - input integer array
- `arr_len` - tells the length of the input array
- `num_splits` - tells the number of partitions the input array is divided into.
- `x` - the key to be searched
- `count` - the count of the key. Initialized to 0.

Note: For array size less than 4, the `num_split` = 2.

First we accept the input as command line arguments. We then validate the inputs and store them in the global array. The last integer input is taken as the key variable to be searched. Based on the num_split parameter we send the `start` pos of each split to a newly created thread. In the runner() of the thread, we first calculate the `end`. Since we assume that the array need not be sorted, we first sort each split in the thread using `sortArray()` function.
Once the split is sorted, we call the `binarySearch()` function on the split to find the position of the key. It returns the position of the key or -1 in case the key doesn't exist in the split. If the key exists in the split, we also would have to check for duplicates in the split. For that we do a linear search on either side of the returned position. We keep updating a `local_cnt` of the number of occurrences of the key in each split. At the end of the thread we update the global count and exit the thread.
In the main(), we join all the threads and print the global count.

Code

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

```c
#include <sys/wait.h>
#include <string.h>
#include <pthread.h>
#define MAX 1000

int isInteger(char* string);
void sortArray(int arr[],int start,int end,int order);
int binarySearch(int arr[], int start, int end, int x);

// define global array
int arr[MAX], arr_len;
// define size of each split - 4
int num_splits = 4;
// define global key
int x;
// global variable to maintain count
int count = 0;

void *runner(void *arg);

int main(int argc, char *argv[])
{
    if(argc < 2)
       printf("Insufficent arguments.\n");
    else
    {
       int len = argc-1;
       x = atoi(argv[len]);
       int n=0, i=1;
       for(;i<len;++i)
       {
             // Validation function. Warning if not integer.
             if(isInteger(argv[i]))
                   arr[n++] = atoi(argv[i]);
             else
                   printf("warning: '%s' is not an integer. will be
ignored.\n",argv[i]);
       }
       // Assign length
       arr_len = n;

       // Warning if the entire argument list is invalid.
       if(n == 0 )
```

```
            printf("The input list is invalid. Please enter an integer
array.\n");
        else
        {
            /* -- Search for x in ? splits using Binary Search -- */
            if(arr_len < num_splits)
                num_splits = 2;

            // Declare the threads
            pthread_t tid[num_splits];

            int split_no = 0;
            int split_size = arr_len/num_splits;

            for(i=0;i<num_splits;++i)
            {
                int *arg = malloc(sizeof(*arg));
                *arg = split_no;
                pthread_create(&tid[i],NULL,runner,arg);
                ++split_no;
            }

            for(i=0;i<num_splits;++i)
                pthread_join(tid[i],NULL);

            printf("%d is present %d times in the given array\n",x,count);
        }
    }
    return 0;
}

// Validation Function
int isInteger(char* str)
{
    int character,i=0;
    for(;i<strlen(str);++i)
    {
        character = str[i];
        // In case of '-'' symbol, continue if not last character in string.
        if(character == 45 && i<strlen(str)-1)
            continue;
        // Else get the decimal value (-48) and check condition.
        character -= 48;
```

```
       if(character < 0 || character > 9)
             return 0;
    }
    return 1;
}

// Runner function for thread
void *runner(void *arg)
{
    int split_no = *((int *) arg);
    int start = (arr_len * split_no)/num_splits;
    int end = (arr_len * (split_no + 1))/num_splits;

    sortArray(arr,start,end,1);

    /* -- Perform Binary Search -- */
    int pos = binarySearch(arr,start,end - 1,x);
    int local_cnt = 0;
    if(pos!=-1)
    {
       ++local_cnt;
       int j = pos+1;
       if(pos+1 < end)
       {
             while((arr[j]==x) && (j<end))
             {
                   ++local_cnt;
                   j = j+1;
             }
       }
       j = pos-1;
       if(pos-1 >= start)
       {
             while((arr[j] == x) && (j>=start))
             {
                   ++local_cnt;
                   j = j-1;
             }
       }
       printf("%d found in split-%d of the array %d
times.\n",x,split_no+1,local_cnt);
    }
    else
```

```
       printf("%d not found in split-%d of the array.\n",x,split_no+1);

    count += local_cnt;
    pthread_exit(NULL);
}


void sortArray(int arr[],int start,int end,int order)
{
    int temp;
       // For swap purposes

    // Initiate Loops for simple Bubble Sort.
    int i=start;
    int n = end - start;
    for(;i<end;++i)
    {
       int j=start;
       for(;j<start+end-i-1;++j)
       {
             if(((order == 1) && (arr[j]>arr[j+1])) || ((order == 0) &&
(arr[j]<arr[j+1])))
             {
                   // Swap arr[j] and arr[j+1]
                   temp = arr[j];
                   arr[j] = arr[j+1];
                   arr[j+1] = temp;
             }
       }
    }
}

int binarySearch(int arr[], int start, int end, int x)
{
      if (end >= start)
      {
      int mid = start + (end - start) / 2;

      if (arr[mid] == x)
            return mid;

      if (arr[mid] > x)
            return binarySearch(arr, start, mid - 1, x);
```

```
        return binarySearch(arr, mid + 1, end, x);
        }
        return -1;
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 7_binarySearch.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 3 4 5 1 2 3 4 1 3 3
3 found in split-1 of the array 1 times.
3 not found in split-2 of the array.
3 found in split-3 of the array 1 times.
3 found in split-4 of the array 1 times.
3 is present 3 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 3 3 4 5 6 3 3 4 1 2 3 1
1 not found in split-1 of the array.
1 not found in split-3 of the array.
1 not found in split-2 of the array.
1 found in split-4 of the array 1 times.
1 is present 1 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 3 3 4 5 6 3 3 4 1 2 3 67
67 not found in split-1 of the array.
67 not found in split-4 of the array.
67 not found in split-3 of the array.
67 not found in split-2 of the array.
67 is present 0 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: With invalid inputs

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 7_binarySearch.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out 2 2 3d s df w 2 2 2
warning: '3d' is not an integer. will be ignored.
warning: 's' is not an integer. will be ignored.
warning: 'df' is not an integer. will be ignored.
warning: 'w' is not an integer. will be ignored.
2 found in split-1 of the array 1 times.
2 found in split-2 of the array 1 times.
2 found in split-4 of the array 1 times.
2 found in split-3 of the array 1 times.
2 is present 4 times in the given array
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

8. Implement using threads : Generation of nth Fibonacci number.

Logic Used

In this program we are required to generate the nth fibonacci number using multi-threading.
We cannot parallelize the bottom-up approach [O(n) method] as there are data dependencies. However we can parallelize the recursive version of the algorithm. Let us first take a look at the standard recursive algorithm.

```
long int fibonacci(int  n)
{
      if(n<=2)
      return 1;
      else
      return fibonacci(n-1) + fibonacci(n-2);
}
```

Here the fibonacci(n-1) and fibonacci(n-2) can be calculated in two different threads and their results can be added.
The runner function of the thread returns 1 when n<=2, otherwise it creates two more threads to calculate fibonacci(n-1) and fibonacci(n-2), it then waits for the result from the child threads and then combines the result to get the nth fibonacci number and returns it.

Code

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

void *fib_thread(void *arg);

int main()
{
```

```
    // Accept input n
    int n;
    printf("Enter n\n");
    scanf("%d",&n);

    // Generate nth fibonacci number
    int *arg = malloc(sizeof(*arg));
    *arg = n;

    pthread_t tid;
    pthread_create(&tid,NULL,fib_thread,arg);

    // Join back the thread
    int *fib_n;
    pthread_join(tid,(void*)&fib_n);

    printf("%dth Fibonacci number is: %d\n",n,*fib_n);

    return 0;
}

void *fib_thread(void *arg)
{
    // cast to int
    int n = *((int *) arg);

    // Declare the return variable
    int *ret = malloc(sizeof(int));
    *ret = 0;

    if(n <= 2)
       *ret = 1;
    else
    {
       int *arg_1 = malloc(sizeof(*arg));
       int *arg_2 = malloc(sizeof(*arg));

       // Declare two pthreads
       pthread_t tid[2];

       // Declare two return integer pointers
       int *ret_1, *ret_2;
```

```
        // Create two threads for each recursion tree
        *arg_1 = n-1;
        pthread_create(&tid[0],NULL,fib_thread,arg_1);

        *arg_2 = n-2;
        pthread_create(&tid[1],NULL,fib_thread,arg_2);

        // Join both threads back
        pthread_join(tid[0],(void*)&ret_1);
        pthread_join(tid[1],(void*)&ret_2);

        // return the sum
        *ret = *ret_1 + *ret_2;
    }
    pthread_exit(ret);
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 8_fibonacci.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
4
4th Fibonacci number is: 3
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
8
8th Fibonacci number is: 21
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

More Examples:

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
12
12th Fibonacci number is: 144
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
16
16th Fibonacci number is: 987
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter n
18
18th Fibonacci number is: 2584
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

9. **[Extra Credits]** Implement using threads : Find the length of the longest common subsequence of two strings.

Logic Used

In this program we are required to find the length of the longest common subsequence of two strings in a multi-threaded fashion. We first accept both the strings and store them in global character arrays s1 and s2.
First let us take a look at a naive recursive algorithm to find the length of the longest common subsequence of two strings.

```
int lcs(int m, int n )
{
   if (m == 0 || n == 0)
     return 0;
   if (s1[m-1] == s2[n-1])
     return 1 + lcs(m-1, n-1);
   else
     return max(lcs(m, n-1), lcs(m-1, n));
}
```

Clearly we can parallelize the recursive algorithm by implementing the algorithm in the runner function of the thread - lcs_thread. In this runner function, each call of lcs would create a new thread with arguments m and n passed to the runner of the thread. Since we have to pass two arguments to each runner function of thread, we use a structure called Param. Let us take a look at the structure.

```
// structure to pass positional values for two strings
struct Param
{
    int m,n;
};
```

In the runner function we calculate the return value and pass it back to the calling function (either a parent thread or main) via pthread_exit(ret) call. Once we get the return value to the main (after exit of the first thread), we print the result and end the program.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <inttypes.h>
#include <pthread.h>
#include <string.h>
#define MAX 1000

// structure to pass positional values for two strings
struct Param
{
    int m,n;
};

// Global declaration of the two strings
char s1[MAX], s2[MAX];

void *lcs_thread(void *arg);
int max(int a, int b);

int main()
{
    // Accept both strings as input
    printf("Enter the first string\n");
    scanf("%s",s1);

    printf("Enter the second string\n");
    scanf("%s",s2);

    // Instantiate a struct
    struct Param *param = (struct Param*)malloc(sizeof(struct Param));
    param->m = strlen(s1);
    param->n = strlen(s2);

    // Create a main thread (initial thread)
    pthread_t tid;
```

```
    pthread_create(&tid,NULL,lcs_thread,param);

    // Define return value from thread
    int *max_len;
    pthread_join(tid,(void*)&max_len);

    printf("Length of LCS is %d\n",*max_len);

    return 0;
}

// Define the LCS Thread
void *lcs_thread(void *arg)
{
    int m = ((struct Param*)arg)->m;
    int n = ((struct Param*)arg)->n;

    // Define return variable
    int *ret = malloc(sizeof(int));
    *ret = 0;

    // Trivial condition
    if(m == 0 || n == 0)
       pthread_exit(ret);

    if(s1[m-1] == s2[n-1])
    {
       // Define a new struct copied from the arg
       struct Param arg_1 = *((struct Param*)arg);
       --arg_1.m;
       --arg_1.n;

       // Recursively call the thread again
       pthread_t tid;
       pthread_create(&tid,NULL,lcs_thread,(void *)&arg_1);

       // Get return value from thread
       int *ret_1;
       pthread_join(tid,(void*)&ret_1);
       *ret = *ret_1 + 1;
    }
    else
    {
```

```
        // Define new structs copied from the arg
        struct Param arg_1 = *((struct Param*)arg);
        --arg_1.n;
        struct Param arg_2 = *((struct Param*)arg);
        --arg_2.m;

        // Recursively call the thread again
        pthread_t tid[2];
        pthread_create(&tid[0],NULL,lcs_thread,(void *)&arg_1);
        pthread_create(&tid[1],NULL,lcs_thread,(void *)&arg_2);

        // Get return value from both threads
        int *ret_1, *ret_2;
        pthread_join(tid[0],(void*)&ret_1);
        pthread_join(tid[1],(void*)&ret_2);

        *ret = max(*ret_1,*ret_2);
    }
    pthread_exit(ret);
}

int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
```

Output

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ gcc -o out 9_lcs.c -lpthread
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the first string
abcdef
Enter the second string
abcef
Length of LCS is 5
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the first string
hdaskdjahskjdha
Enter the second string
hhajl
Length of LCS is 3
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```

Case: When then length of lcs is zero.

```
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$ ./out
Enter the first string
jdaskljlk
Enter the second string
oooo
Length of LCS is 0
adheshreghu@adheshreghu-Inspiron-5570:~/Documents/SEM5/OS/Lab/Week5$
```
