Assignment-4
Fork assignment-3

Subash Mylraj
(CED18I051)

16 September 2020

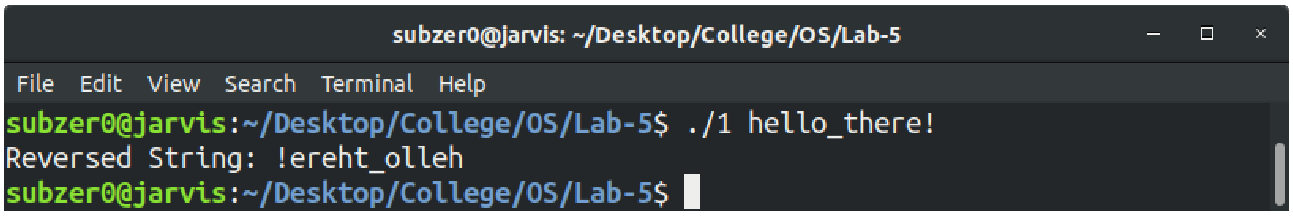
Question 1: Parent sets up a string which is read by child, reversed there and read back the parent

Code:

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

char* strrev (char*);

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }
   if(pipe(fd2) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();

   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);

         char str[100];
         read(fd1[0], str, 100);

         close(fd1[0]);

         close(fd2[0]);

         write(fd2[1], strrev(str), strlen(str));

         close(fd2[1]);

      }
      else{        // Parent block
```

```
        close(fd1[0]);

        write(fd1[1], argv[1], strlen(argv[1]));
        close(fd1[1]);

        wait(NULL);

        close(fd2[1]);

        char rev_str[100];
        read(fd2[0], rev_str, 100);
        printf("Reversed String: %s\n", rev_str);

        close(fd2[0]);
     }
   }
   return 0;
}

char *strrev(char *str)
{
     char *itr1, *itr2;

     for (itr1=str, itr2=str+strlen(str)-1; itr2>itr1; ++itr1, --itr2)
     {
          *itr1 ^= *itr2;
          *itr2 ^= *itr1;
          *itr1 ^= *itr2;
     }
     return str;
}
```

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./1 hello_there!
Reversed String: !ereht_olleh
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

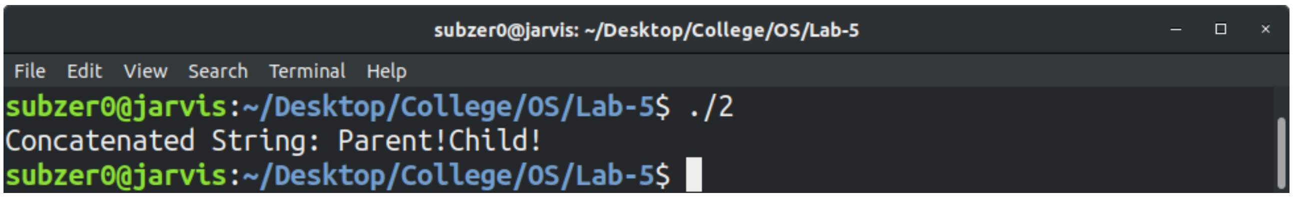
Question 2: Parent sets up string 1 and child sets up string 2. String 2 concatenated to string 1 at parent end and then read back at the child end.

Code:

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

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
```

```
  pid_t pid;

  if(pipe(fd1) == -1){
     printf("Unable to create pipe\n");
  }
  if(pipe(fd2) == -1){
     printf("Unable to create pipe\n");
  }

  pid = fork();

  if(pid < 0){
     printf("fork failed\n");
  }
  else{
     if(pid == 0){  // Child block
        close(fd1[1]);

        char str1[100], * str2 = "Child!";
        read(fd1[0], str1, 100);

        close(fd1[0]);

        close(fd2[0]);

        write(fd2[1], strcat(str1, str2), strlen(str1)+strlen(str2));

        close(fd2[1]);

     }
     else{           // Parent block
        close(fd1[0]);

        char* str = "Parent!";
        write(fd1[1], str, strlen(str));
        close(fd1[1]);

        wait(NULL);

        close(fd2[1]);

        char cat_str[100];
        read(fd2[0], cat_str, 100);
        printf("Concatenated String: %s\n", cat_str);

        close(fd2[0]);
     }
  }
  return 0;
}
```

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./2
Concatenated String: Parent!Child!
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 3: Substring generation at child end of a string setup at parent process end.

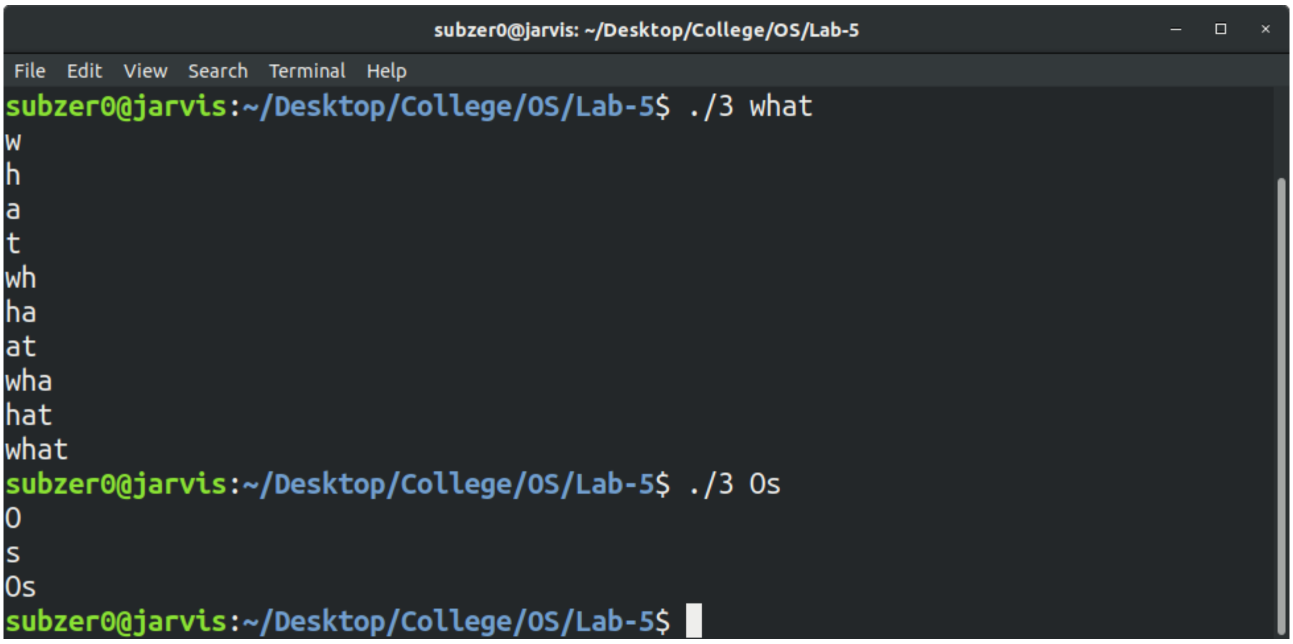
Code:

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

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();
   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);

         char str1[100];
         read(fd1[0], str1, 100);

            for(int len=1; len<=strlen(str1); len+=1){
                for(int ptr=0; ptr+len-1<strlen(str1); ptr+=1){
                    for(int i=ptr; i<ptr+len; i++){
                        printf("%c", str1[i]);
                    }
                    printf("\n");
                }
            }

         close(fd1[0]);
      }
      else{         // Parent block
         close(fd1[0]);

         write(fd1[1], argv[1], strlen(argv[1]));
         close(fd1[1]);

         wait(NULL);
      }
   }
   return 0;
}
```

Explanation:

Parent sends a string to child through a pipe. Child reads the string and finds all the possible substrings and prints it to terminal.

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./3 what
w
h
a
t
wh
ha
at
wha
hat
what
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./3 Os
O
s
Os
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

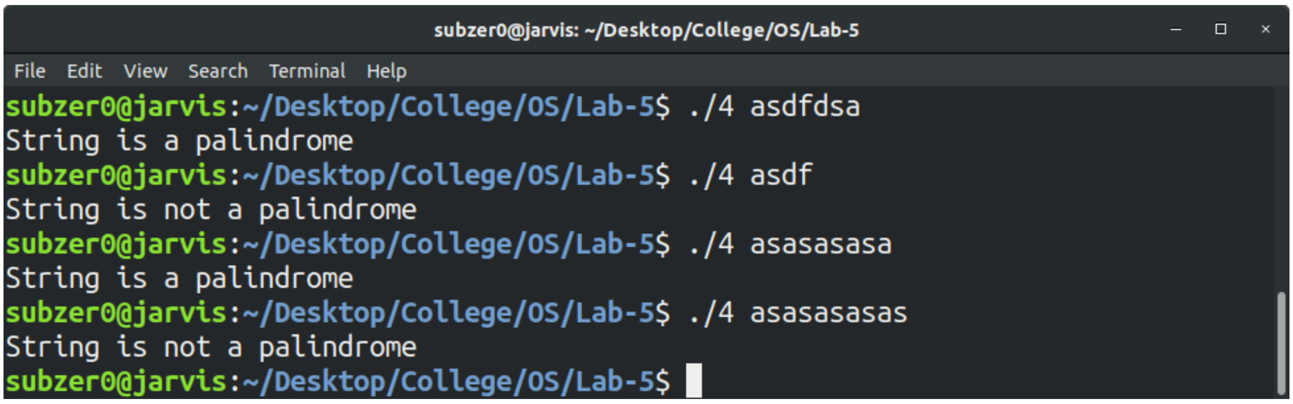
Question 4: String reversal and palindrome check using pipes / shared memory.

Code:

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

char* strrev (char*);

int main(int argc, char const *argv[])
{
   int fd1[2], fd2[2];
   pid_t pid;

   if(pipe(fd1) == -1){
      printf("Unable to create pipe\n");
   }
   if(pipe(fd2) == -1){
      printf("Unable to create pipe\n");
   }

   pid = fork();

   if(pid < 0){
      printf("fork failed\n");
   }
   else{
      if(pid == 0){  // Child block
         close(fd1[1]);
```

```
        char str[100] = {0};
        read(fd1[0], str, 100);

        close(fd1[0]);

        close(fd2[0]);
          char temp[100];
          strcat(temp, str);
          char* rev = strrev(temp);

        if(strcmp(str, rev) == 0){
             write(fd2[1], "1", 1);
          }
          else
             write(fd2[1], "0", 1);

        close(fd2[1]);

      }
      else{         // Parent block
        close(fd1[0]);

        write(fd1[1], argv[1], strlen(argv[1]));
        close(fd1[1]);

        wait(NULL);

        close(fd2[1]);

        char result[1];
        read(fd2[0], result, 1);

          if(result[0] == '0'){
             printf("String is not a palindrome\n");
          }
          else{
             printf("String is a palindrome\n");
          }

        close(fd2[0]);
      }
    }
    return 0;
}

char *strrev(char *str)
{
    char *itr1, *itr2;

    for (itr1=str, itr2=str+strlen(str)-1; itr2>itr1; ++itr1, --itr2){
        *itr1 ^= *itr2;
        *itr2 ^= *itr1;
        *itr1 ^= *itr2;
    }
    return str;
}
```

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asdfdsa
String is a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asdf
String is not a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asasasasa
String is a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./4 asasasasas
String is not a palindrome
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 5: Armstrong number generation within a range. The digit extraction, cubing can be responsibility of child while the checking for sum == no can happen in child and the output list in the child.

Code:

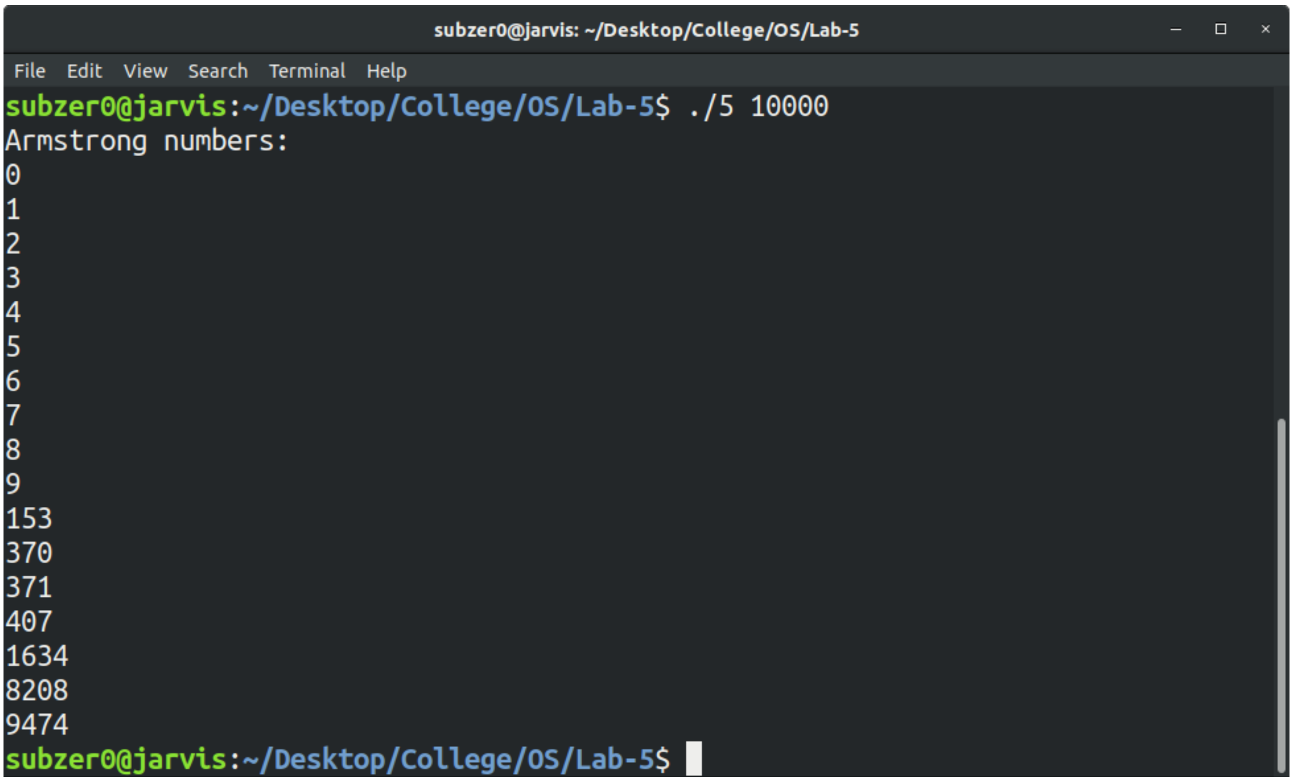
```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>
#include<math.h>
#include<sys/ipc.h>
#include<sys/shm.h>

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

   int n=atoi(argv[1]);

   int i=0, num=0, sum=0, pow_to=0, j=0;
   struct armstrong {
      int numbers[100];
      int n;
   };

   key_t key = ftok("shmfile", 65);
   int shmid = shmget(key, 5*sizeof(struct armstrong), 0666|IPC_CREAT);
   struct armstrong *armstg = shmat(shmid, 0, 0);

   for(i=0; i<n; i=i+1){

      sum = 0;
      num = i;
      pow_to = 0;
      for(; num>0; num/=10)
         pow_to+=1;
      num = i;

      pid_t pid = fork();
      if(pid == 0){
         while (num>0){
            int digit = num%10;
            sum = sum + pow(digit, pow_to);
            num = num/10;
         }
         if(sum == i){
            armstg->numbers[armstg->n] = i;
```

```
                armstg->n += 1;
            }
            exit(0);
        }

    }

    wait(NULL);

    printf("Armstrong numbers:\n");
    for(int i=0; i<armstg->n; i++)
        printf("%d\n", armstg->numbers[i]);

    shmdt(armstg);
    shmctl(shmid,IPC_RMID,NULL);

    return 0;
}
```

Explanation:

This program was done using shared memory. The parent forks for every number in the range. The child processes find whether the number they forked at is an armstrong number or not. If the number is an armstrong number, then it is appended to an array where all the armstrong numbers within that range are stored. This array is made available to all the processes using shared memory. A structure was used to store both the array the next iterator to that array.

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./5 10000
Armstrong numbers:
0
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 6: Ascending Order sort within Parent and Descending order sort (or vice versa) within the child process of an input array. (u can view as two different outputs –first entire array is ascending order sorted in op and then the second part descending order output).

Code:

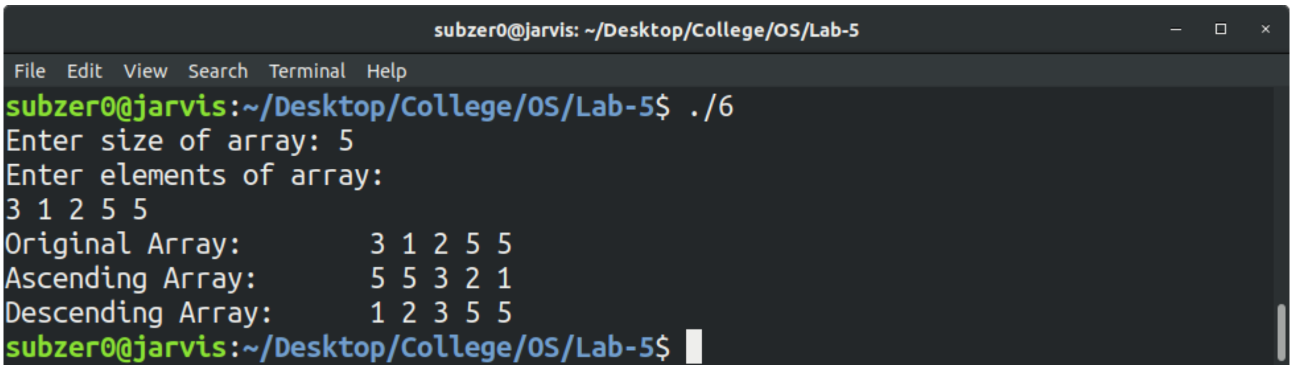
```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
#include<pthread.h>

struct arguments
{
   int arr[10];
   int n;
   int opt;
};

void* sort (void *arg){

   struct arguments* args = (struct arguments*)arg;
   int opt = args->opt;
   if(args->opt){
      printf("Descending Array:   ");
   }
   else{
      printf("Ascending Array:    ");
   }

   for(int i=0; i<args->n; i++){
      for(int j=1; j<args->n-i; j++){
         if(((opt*(args->arr[j] < args->arr[j-1])) + ((1-opt)*(args->arr[j] > args->arr[j-1])))){
            args->arr[j] += args->arr[j-1];
            args->arr[j-1] = args->arr[j] - args->arr[j-1];
            args->arr[j] -= args->arr[j-1];
         }
      }
   }

   for(int i=0; i<args->n; i++)
      printf("%d ", args->arr[i]);
   printf("\n");

   pthread_exit(NULL);
}

int main (){

   struct arguments arg_asc, arg_dsc;

   printf("Enter size of array: ");
   scanf("%d", &arg_asc.n);

   printf("Enter elements of array: ");
   for(int i=0; i<arg_asc.n; i++)
      scanf("%d", &arg_asc.arr[i]);

   printf("Original Array:     ");
   for(int i=0; i<arg_asc.n; i++)
```

```
        printf("%d ", arg_asc.arr[i]);
    printf("\n");

    arg_dsc = arg_asc;

    struct arguments *asc_ptr=&arg_asc, *dsc_ptr=&arg_dsc;
    pthread_t asc, dsc;

    arg_asc.opt = 1;
    pthread_create(&asc, NULL, sort, asc_ptr);
    arg_dsc.opt = 0;
    pthread_create(&dsc, NULL, sort, dsc_ptr);

    pthread_join(asc, NULL);
    pthread_join(dsc, NULL);

    return 0;

}
```

Explanation:

Ascending and Descending sorts are done in 2 separate threads. There is only one sort function which accepts an additional argument for the type of sort. The conditional statement within the sort makes sure that both ascending and descending sorting can be performed. The conditional statement resembles to the expression of a multiplexor.

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./6
Enter size of array: 5
Enter elements of array:
3 1 2 5 5
Original Array:         3 1 2 5 5
Ascending Array:        5 5 3 2 1
Descending Array:       1 2 3 5 5
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 7: Implement a multiprocessing version of binary search where the parent searches for the key in the first half and subsequent splits while the child searches in the other half of the array. By default u can implement a search for the first occurrence and later extend to support multiple occurrence (duplicated elements search as well).

Code:

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

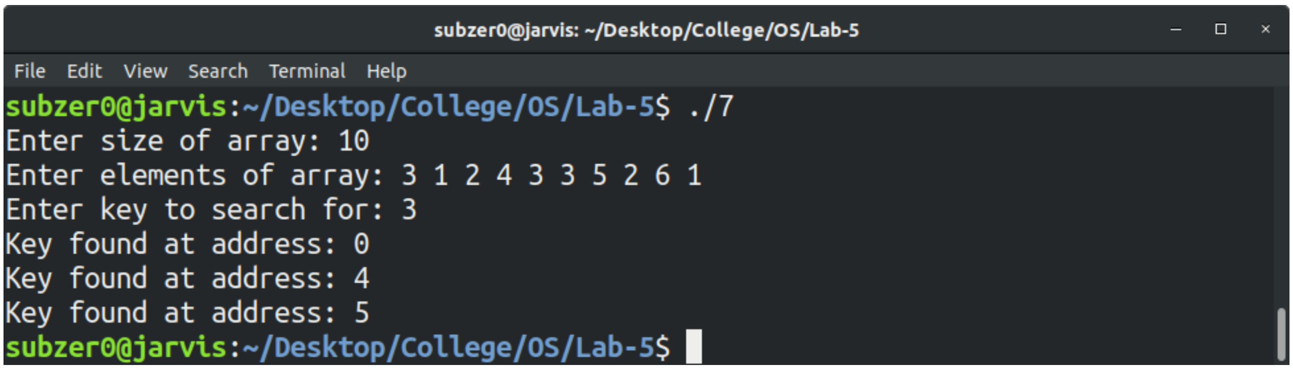
```c
#include<stdlib.h>
#include<pthread.h>

struct arguments{
   int arr[20][2];
   int start;
   int end;
   int key;
};

pthread_t threads[100];
int itr=0;

void* binary_search (void *);

int main (){

   struct arguments *args = (struct arguments *)malloc(sizeof(struct arguments));
   int n;
   printf("Enter size of array: ");
   scanf("%d", &n);

   printf("Enter elements of array: ");
   for(int i=0; i<n; i++)
      scanf("%d", &args->arr[i][0]);

   printf("Enter key to search for: ");
   scanf("%d", &args->key);

   args->end = n;
   args->start = 0;

   binary_search(args);

   for(int i=0; i<itr; i++){
      pthread_join(threads[i], NULL);
   }

   for(int i=0; i<n; i++)
      if(args->arr[i][1] == 1)
         printf("Key found at address: %d\n", i);

   return 0;
}

void* binary_search (void* a){

   struct arguments* args = (struct arguments*) a;
   int mid = (args->end + args->start)/2;
   int end = args->end, start = args->start;

   if(args->start > args->end){
      return NULL;
   }
   if(args->start == args->end){
      if(args->arr[mid][0] == args->key){
         args->arr[mid][1] = 1;
      }
      return NULL;
   }
   if(args->start+1 == args->end){

         args->end=args->start;
         pthread_create(&threads[itr], NULL, binary_search, args);itr++;

         args->start=args->end=end;
```

```
        pthread_create(&threads[itr], NULL, binary_search, args);itr++;
        args->start=start; args->end=end;
    }
    if(args->arr[mid][0] == args->key){
      args->arr[mid][1] = 1;
    }

      args->end = mid-1;
      binary_search(args);
      pthread_create(&threads[itr], NULL, binary_search, args);itr++;

      args->start = mid+1;
      args->end = end;
      binary_search(args);
      pthread_create(&threads[itr], NULL, binary_search, args);itr++;
}
```

Output:

Explanation:

This code implements a search algorithm (non-sorted array) by initially checking if the middle element is the key. If not, then both the left and the right sub-halves are checked in the same recursively. The checking for the left and right sub-halves is done using threading. Hence this is a parallelized implementation. This is slightly similar to that of binary search in the fact that the middle element is checked in every iteration.


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./7
Enter size of array: 10
Enter elements of array: 3 1 2 4 3 3 5 2 6 1
Enter key to search for: 3
Key found at address: 0
Key found at address: 4
Key found at address: 5
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

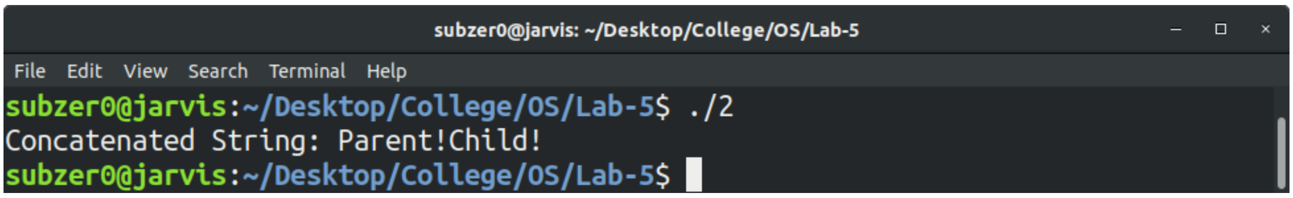
Question 8: Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a parent child relationship to contribute a faster version of fib series generation.

Explanation:

Fibonacci series depends on its previous 2 elements. Hence whatever algo is used, to calculate the nth element, its previous elements have to be calculated. So, the time complexity can never be below O(n).

For example, to calculate fib(7), we need to find fib(6), fib(5), ... fib(2), fib(1). To calculate fib(6), we need to find fib(5), fib(4), ... fib(2), fib(1). Hence finding all these in a linear sequential manner would lead to a time complexity of about O(2 n).

But with parallelization or by calculating the series till n from a bottom up manner, we can get the time complexity down to O(n). Using threading or forking will not improve the theoretical complexity of the algorithm.



```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./2
Concatenated String: Parent!Child!
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```

Question 9: Longest Common Subsequence (Extra-Credits)

Code:

```
#include<iostream>
#include<string.h>
#include<pthread.h>
#include<vector>
#include<bits/stdc++.h>

using namespace std;

vector<string> str1_subseq;

void* check_common_seq (void* arg);
void subsequence(string str);


string str1, str2;
vector<string> common;

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

   if(strlen(argv[1]) < strlen(argv[2])){
      str1=argv[1];
      str2=argv[2];
   }
   else{
      str1=argv[2];
      str2=argv[1];
   }

   // finds all the subsequences of str1
   subsequence(str1);
   sort(str1_subseq.begin(), str1_subseq.end());
   str1_subseq.erase(std::unique(str1_subseq.begin(), str1_subseq.end()), str1_subseq.end());

   pthread_t threads[100];
   int itr=0;

   // find all common subsequences
   for(int i=0; i<str1_subseq.size(); i++){

      pthread_create(&threads[itr], NULL, check_common_seq, (void*)&str1_subseq[i]);
      pthread_join(threads[itr], NULL);
      itr++;
   }

   // Find the longest common sequence from all common sequences
   string lcs = common[0];
   for(int i=1; i<common.size(); i++){
      if(lcs.length() < common[i].length())
         lcs = common[i];\
   }
```

```
    if(lcs.length() >= 2){
        cout<<"Longest Common Subsequence: "<<lcs<<endl;
    }
    else{
        cout<<"Longest Common Subsequence of length more than 2 does not exist"<<endl;
    }
    return 0;
}

void* check_common_seq (void* arg){

    string* s = (string*)arg;

    int i, j=0;
    for(i=0; i<(*s).length(); i++){
        int flag=0;
        for(j; j<str2.length(); j++){
            if((*s)[i] == str2[j]){
                j++;
                flag = 1;
                break;
            }
        }
        if(flag == 0){
            break;
        }
    }
    if(i == (*s).length()){
        // cout<<*s<<endl;
        common.push_back(*s);
    }

    pthread_exit(NULL);
}

void subsequence(string str){
    for (int i = 0; i < str.length(); i++)
    {
        for (int j = str.length(); j > i; j--)
        {
            string sub_str = str.substr(i, j);
            str1_subseq.push_back(sub_str);

            for (int k = 1; k < sub_str.length() - 1; k++)
            {
                string sb = sub_str;
                sb.erase(sb.begin() + k);
                subsequence(sb);
            }
        }
    }
}
```

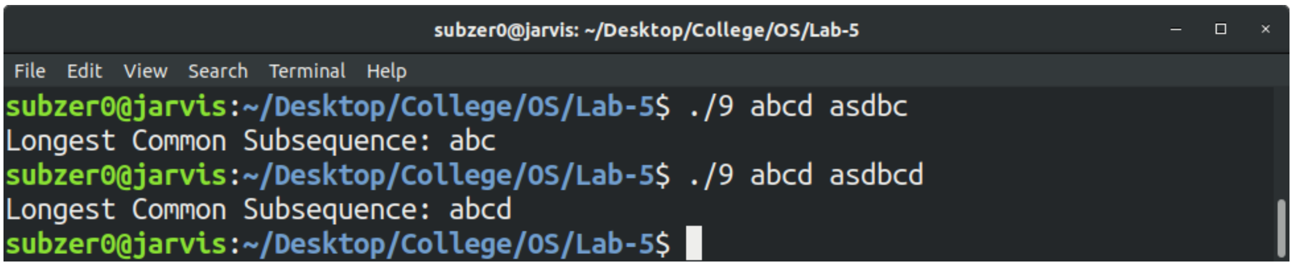
Explanation:

The program can be viewed as two parts:

1. The first part finds all the subsequences of the first string.
2. The second part checks if any of those subsequences matches with any subsequence of the second string.

In this program, I have multithreaded the second part.

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-5
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./9 abcd asdbc
Longest Common Subsequence: abc
subzer0@jarvis:~/Desktop/College/OS/Lab-5$ ./9 abcd asdbcd
Longest Common Subsequence: abcd
subzer0@jarvis:~/Desktop/College/OS/Lab-5$
```