Assignment-3
Fork assignment-2

Subash Mylraj
(CED18I051)

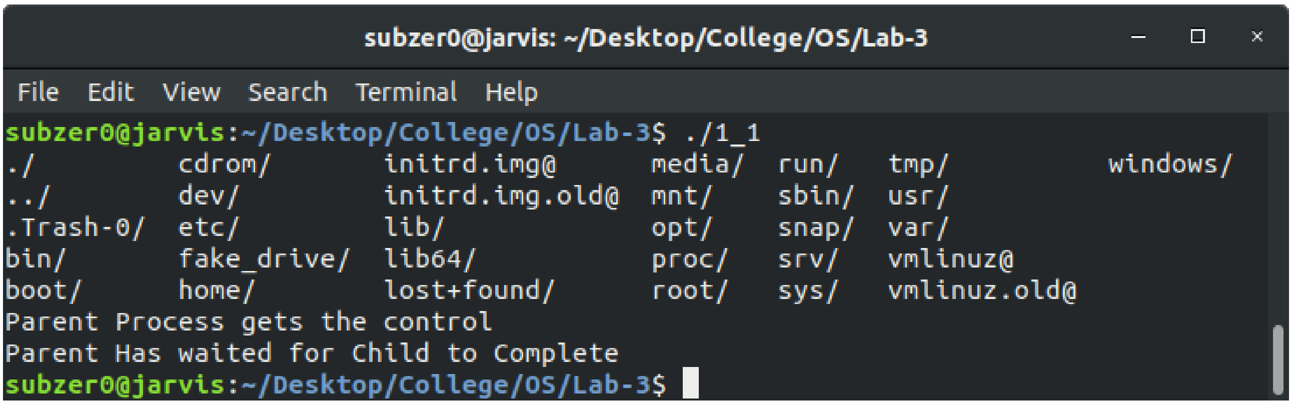
16 September 2020

Question 1: Test Drive all the examples discussed so far in the class for the usage of wait, exec call variants.

Code 1:

```
# include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main ()
{
   pid_t pid; // this is to use the pid data type - relevant headers above
   pid = fork();
   if (pid == 0){

      char * args[]={"ls","-aF","/",0};
      char * env[]={0};
      execve("/bin/ls",args,env);
   }
   else{
      wait (NULL); // parent waits for the child to complete execn.
      printf("Parent Process gets the control \n");
      printf("Parent Has waited for Child to Complete\n");
   }
   return 0;
}
```

Output:

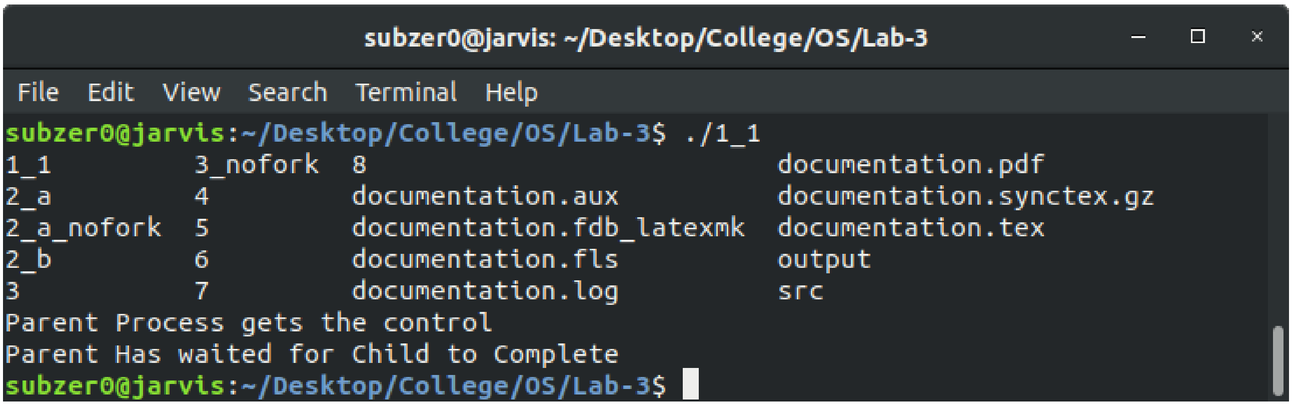

subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./1_1
./          cdrom/       initrd.img@      media/  run/    tmp/           windows/
../         dev/         initrd.img.old@  mnt/    sbin/   usr/
.Trash-0/   etc/         lib/             opt/    snap/   var/
bin/        fake_drive/  lib64/           proc/   srv/    vmlinuz@
boot/       home/        lost+found/      root/   sys/    vmlinuz.old@
Parent Process gets the control
Parent Has waited for Child to Complete
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ 
```

Code 2:

```
# include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main ()
{
   pid_t pid; // this is to use the pid data type - relevant headers above
   pid = fork();
   if (pid == 0){
      execlp("ls", "ls", NULL); // child image is now ls command
   }
   else{
      wait (NULL); // parent waits for the child to complete execn.
      printf("Parent Process gets the control \n");
      printf("Parent Has waited for Child to Complete\n");
   }
   return 0;
}
```

Output:



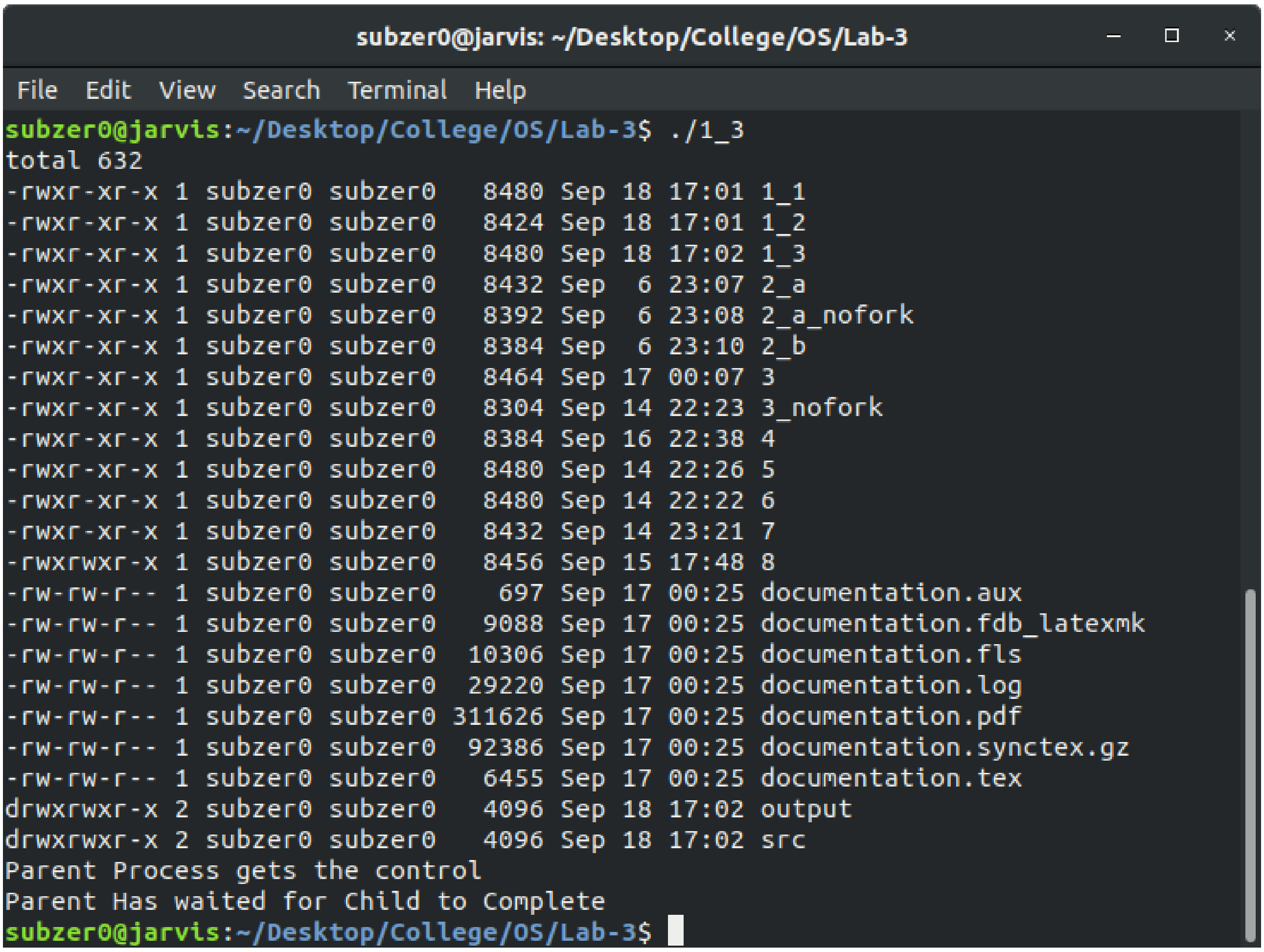
subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./1_1
1_1          3_nofork   8                        documentation.pdf
2_a          4          documentation.aux        documentation.synctex.gz
2_a_nofork   5          documentation.fdb_latexmk  documentation.tex
2_b          6          documentation.fls        output
3            7          documentation.log        src
Parent Process gets the control
Parent Has waited for Child to Complete
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

Code 3:

```
# include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main ()
{
   pid_t pid; // this is to use the pid data type - relevant headers above
   pid = fork();
   if (pid == 0){
      char *const argv[] = {"/bin/ls","-l", NULL};
      execv(argv[0], argv);
   }
   else{
      wait (NULL); // parent waits for the child to complete execn.
      printf("Parent Process gets the control \n");
      printf("Parent Has waited for Child to Complete\n");
   }
   return 0;
}
```

Output:

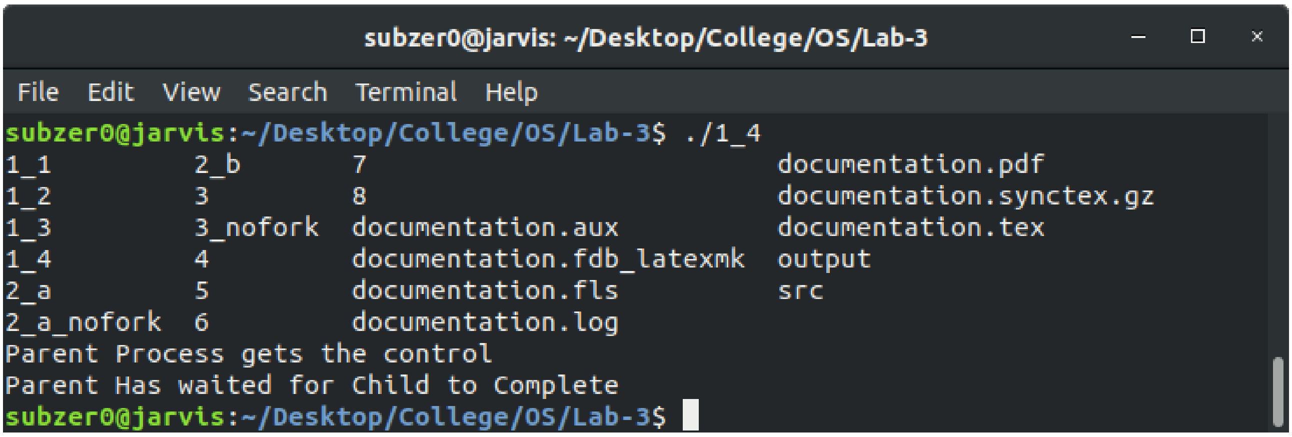

subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./1_3
total 632
-rwxr-xr-x 1 subzer0 subzer0   8480 Sep 18 17:01 1_1
-rwxr-xr-x 1 subzer0 subzer0   8424 Sep 18 17:01 1_2
-rwxr-xr-x 1 subzer0 subzer0   8480 Sep 18 17:02 1_3
-rwxr-xr-x 1 subzer0 subzer0   8432 Sep  6 23:07 2_a
-rwxr-xr-x 1 subzer0 subzer0   8392 Sep  6 23:08 2_a_nofork
-rwxr-xr-x 1 subzer0 subzer0   8384 Sep  6 23:10 2_b
-rwxr-xr-x 1 subzer0 subzer0   8464 Sep 17 00:07 3
-rwxr-xr-x 1 subzer0 subzer0   8304 Sep 14 22:23 3_nofork
-rwxr-xr-x 1 subzer0 subzer0   8384 Sep 16 22:38 4
-rwxr-xr-x 1 subzer0 subzer0   8480 Sep 14 22:26 5
-rwxr-xr-x 1 subzer0 subzer0   8480 Sep 14 22:22 6
-rwxr-xr-x 1 subzer0 subzer0   8432 Sep 14 23:21 7
-rwxrwxr-x 1 subzer0 subzer0   8456 Sep 15 17:48 8
-rw-rw-r-- 1 subzer0 subzer0    697 Sep 17 00:25 documentation.aux
-rw-rw-r-- 1 subzer0 subzer0   9088 Sep 17 00:25 documentation.fdb_latexmk
-rw-rw-r-- 1 subzer0 subzer0  10306 Sep 17 00:25 documentation.fls
-rw-rw-r-- 1 subzer0 subzer0  29220 Sep 17 00:25 documentation.log
-rw-rw-r-- 1 subzer0 subzer0 311626 Sep 17 00:25 documentation.pdf
-rw-rw-r-- 1 subzer0 subzer0  92386 Sep 17 00:25 documentation.synctex.gz
-rw-rw-r-- 1 subzer0 subzer0   6455 Sep 17 00:25 documentation.tex
drwxrwxr-x 2 subzer0 subzer0   4096 Sep 18 17:02 output
drwxrwxr-x 2 subzer0 subzer0   4096 Sep 18 17:02 src
Parent Process gets the control
Parent Has waited for Child to Complete
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

Code 4:

```c
# include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main ()
{
   pid_t pid; // this is to use the pid data type - relevant headers above
   pid = fork();
   if (pid == 0){
      char *const argv[] = {"/bin/ls", NULL};
      execvp(argv[0], argv);
   }
   else{
      wait (NULL); // parent waits for the child to complete execn.
      printf("Parent Process gets the control \n");
      printf("Parent Has waited for Child to Complete\n");
   }
   return 0;
}
```

Output:



subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./1_4
1_1         2_b        7                         documentation.pdf
1_2         3          8                         documentation.synctex.gz
1_3         3_nofork   documentation.aux         documentation.tex
1_4         4          documentation.fdb_latexmk output
2_a         5          documentation.fls         src
2_a_nofork  6          documentation.log
Parent Process gets the control
Parent Has waited for Child to Complete
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

Question 2.a: Odd and Even series generation for n terms using Parent Child relationship (say odd is the duty of the parent and even series as that of child)

Forked Implementation

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

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

   int n = atoi(argv[1]);

    pid_t pid=0;
    pid = fork();

   if(pid != 0){
        printf("Even series: ");
        for(int i=0; i<n; i+=2)
            printf("%d ", i);
        printf("\n");
    }else{
        printf("Odd series: ");
        for(int i=1; i<n; i+=2)
            printf("%d ", i);
        printf("\n");
    }


   return 0;
}
```

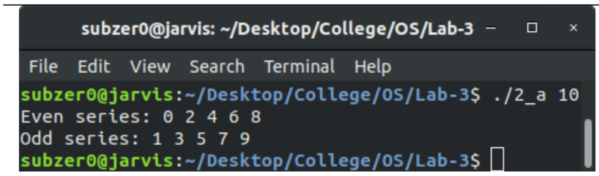
Non Forked implementation

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

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

   int n = atoi(argv[1]);

    printf("Even series: ");
    for(int i=0; i<n; i+=2)
        printf("%d ", i);
    printf("\n");

    printf("Odd series: ");
    for(int i=1; i<n; i+=2)
        printf("%d ", i);
    printf("\n");

   return 0;
}
```

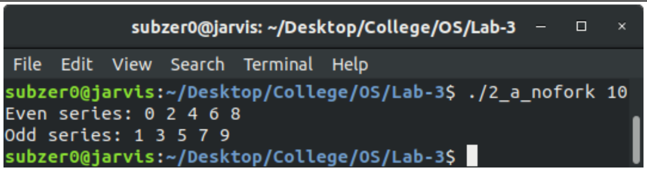
Running and timing both the programs for input of 100000 is as follows:

Forked Implementation	Non Forked implementation
0m0.73s	0m0.109s

Output: for input n=10


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./2_a 10
Even series: 0 2 4 6 8
Odd series: 1 3 5 7 9
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

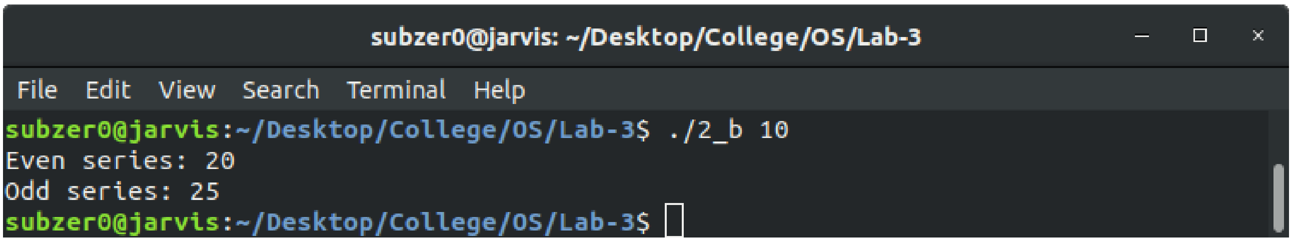
```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./2_a_nofork 10
Even series: 0 2 4 6 8
Odd series: 1 3 5 7 9
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

Question 2.b: given a series of n numbers (u can assume natural numbers till n) generate the sum of odd terms in the parent and the sum of even terms in the child process

Code:

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

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

   int n = atoi(argv[1]);

    pid_t pid=0;
    pid = fork();

   if(pid != 0){
       printf("Even series: ");
       int sum = 0;
       for(int i=0; i<n; i+=2)
           sum += i;
       printf("%d\n", sum);
    }else{
       printf("Odd series: ");
       int sum = 0;
       for(int i=1; i<n; i+=2)
           sum += i;
       printf("%d\n", sum);
    }


   return 0;
}
```

Output:


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./2_b 10
Even series: 20
Odd series: 25
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ 
```

Question 3: 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.

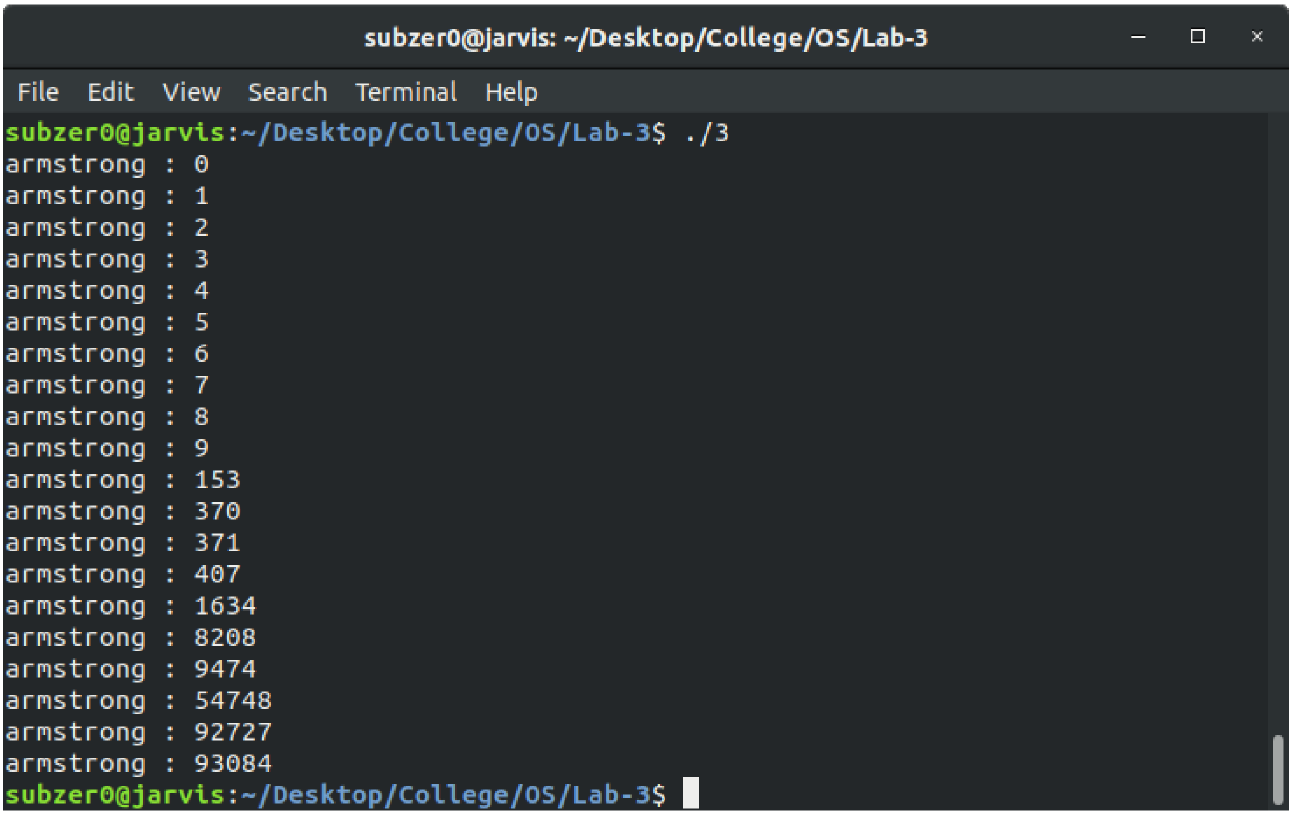
Code:

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

int main (){

   int n=100000;

   int i=0, num=0, sum=0, pow_to=0, j=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 = vfork();
        if(pid == 0){
            while (num>0){
                int digit = num%10;
                sum = sum + pow(digit, pow_to);
                num = num/10;
            }
            exit(0);
        }
        else{
            wait(NULL);
            if(sum == i)
                printf("armstrong : %d\n", i);
        }
    }

    return 0;
}
```

Output:

Something to be noted here is that, this question is not asking for a efficient parallelized output. Since this assignment is to train one's skills in using fork, this code does not efficiently parallelize output. Nevertheless, the required output using the algorithm stated in the question is as follows.


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./3
armstrong : 0
armstrong : 1
armstrong : 2
armstrong : 3
armstrong : 4
armstrong : 5
armstrong : 6
armstrong : 7
armstrong : 8
armstrong : 9
armstrong : 153
armstrong : 370
armstrong : 371
armstrong : 407
armstrong : 1634
armstrong : 8208
armstrong : 9474
armstrong : 54748
armstrong : 92727
armstrong : 93084
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

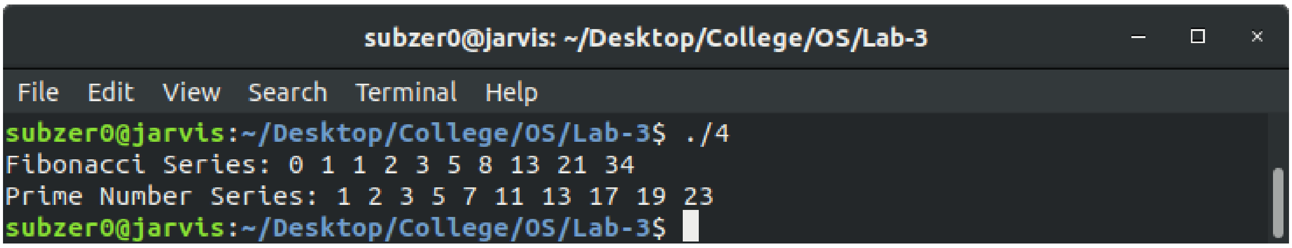
Question 4: Fibonacci Series AND Prime parent child relationship (say parent does fib Number generation using series and child does prime series)

Code:

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

int main (){

   pid_t pid = fork();
   int n = 10;

   if(pid == 0){
      int flag = 0, num = 3;
      printf("Prime Number Series: 1 2 ");

      while (n > 2){
         flag = 0;
         for(int i=2; i<=num/2+1; i++){
            if(num%i == 0){
               flag = 1;
               break;
            }
         }

         if(flag == 0){
            printf("%d ", num);
            n -= 1;
         }
         num += 1;
      }
      printf("\n");
   }
   else{
      printf("Fibonacci Series: 0 1 ");
      int fib_a = 0, fib_b = 1;
      for(int i=2; i<n; i++){
         printf("%d ", fib_a+fib_b);
         fib_b = fib_a + fib_b;
         fib_a = fib_b - fib_a;
      }
      printf("\n");
   }

   return 0;
}
```

Output:

This code generates a series of length n for both Fibonacci and prime series. Both the series are generated in parallel. The child process takes care of generating the prime series while the parent process process takes care of generating the Fibonacci series.


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./4
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
Prime Number Series: 1 2 3 5 7 11 13 17 19 23
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

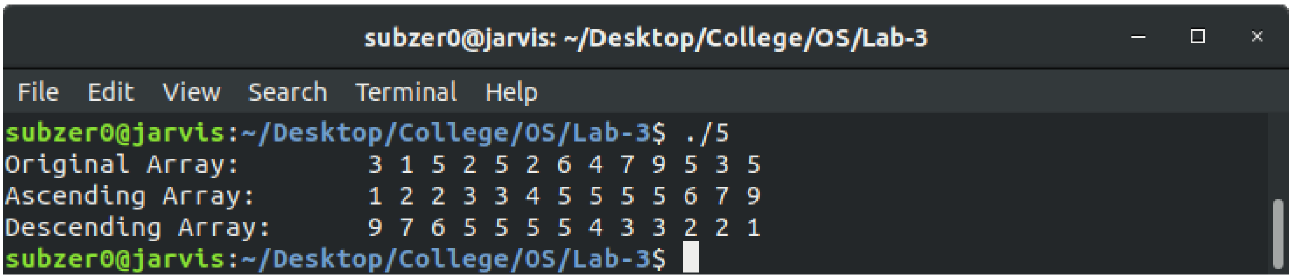
Question 5: Ascending Order sort within Parent and Descending order sort (or vice versa) within the child process of an input array. (u can view as two different outputs –first entire array is asc order sorted in op and then the second part desc order output)

Code:

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

int main (){

    int arr[] = {3, 1, 5, 2, 5, 2, 6, 4, 7, 9, 5, 3, 5}, n = sizeof(arr)/sizeof(int);

    printf("Original Array:     ");
    for(int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    pid_t pid = vfork();

    if(pid == 0){

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

        printf("Ascending Array:    ");
        for(int i=0; i<n; i++)
            printf("%d ", arr[i]);
        printf("\n");

        exit(0);
    }
    else{

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

        printf("Descending Array:   ");
        for(int i=0; i<n; i++)
            printf("%d ", arr[i]);
        printf("\n");

    }

    return 0;
}
```

Output:

This code sorts the given array in both ascending and descending order. Both the sorts are done in parallel. The child process takes care of sorting the array in ascending order while the parent process takes care of sorting the array in descending order.



subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./5
Original Array:        3 1 5 2 5 2 6 4 7 9 5 3 5
Ascending Array:       1 2 2 3 3 4 5 5 5 5 6 7 9
Descending Array:      9 7 6 5 5 5 5 4 3 3 2 2 1
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ 
```

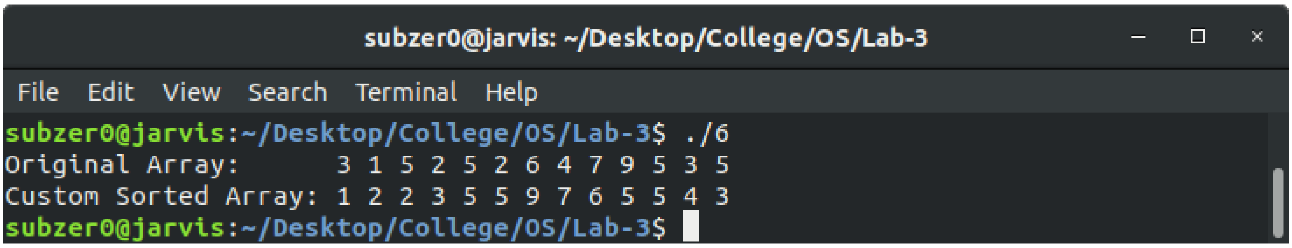
Question 6: Given an input array use parent child relationship to sort the first half of array in ascending order and the trailing half in descending order (parent / child is ur choice)

Code:

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

int main (){

    int arr[] = {3, 1, 5, 2, 5, 2, 6, 4, 7, 9, 5, 3, 5}, n = sizeof(arr)/sizeof(int);

    printf("Original Array:   ");
    for(int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    pid_t pid = vfork();

    if(pid == 0){

        for(int i=0; i<n/2; i++){
            for(int j=1; j<n/2-i; j++){
                if(arr[j] < arr[j-1]){
                    arr[j] += arr[j-1];
                    arr[j-1] = arr[j] - arr[j-1];
                    arr[j] -= arr[j-1];
                }
            }
        }
        exit(0);
    }
    else{

        for(int i=n/2; i<n; i++){
            for(int j=n/2+1; j<n-i+n/2; j++){
                if(arr[j] > arr[j-1]){
                    arr[j] += arr[j-1];
                    arr[j-1] = arr[j] - arr[j-1];
                    arr[j] -= arr[j-1];
                }
            }
        }
    }
    printf("Custom Sorted Array: ");
    for(int i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

Output:

This code sorts the first half of the array in ascending and the other half in descending order. vfork() was used to achieve this goal. One thing to be noted is that, vfork does not achieve true parallelism when used. So this code is not a parallelized implementation but rather a multiprocessed implementation.


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./6
Original Array:       3 1 5 2 5 2 6 4 7 9 5 3 5
Custom Sorted Array: 1 2 2 3 5 5 9 7 6 5 5 4 3
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

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)

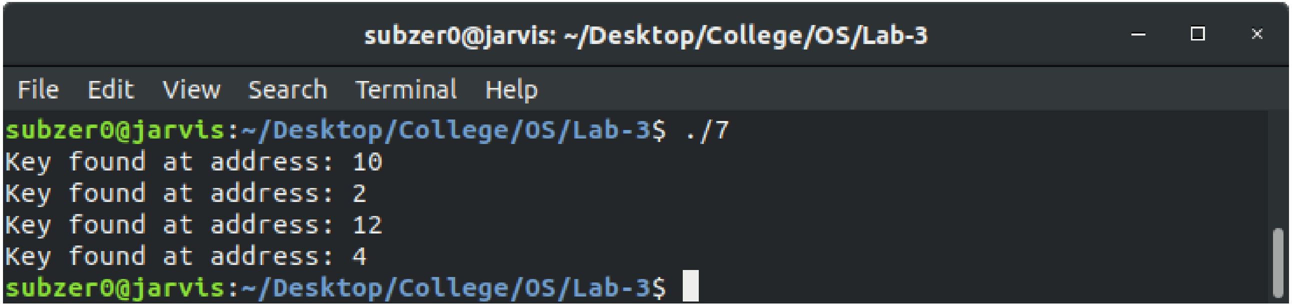
Code:

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

void binary_search (int [], int, int, int);

int main (){
    int arr[] = {3, 1, 5, 2, 5, 2, 6, 4, 7, 9, 5, 3, 5}, n = sizeof(arr)/sizeof(int);
    int key = 5;
    binary_search(arr, 0, n, key);

    return 0;

}

void binary_search (int arr[], int start, int end, int key){

    int mid = (end + start)/2;

    if(start > end){
        return;
    }
    if(start == end){
        if(arr[mid] == key){
            printf("Key found at address: %d\n", mid);
        }
        return;
    }
    if(start+1 == end){
        binary_search(arr, start, start, key);
        binary_search(arr, end, end, key);
    }
    if(arr[mid] == key){
        printf("Key found at address: %d\n", mid);
    }

    pid_t pid = fork();
    if(pid == 0){
        binary_search(arr, start, mid-1, key);
    }
    else{
        binary_search(arr, mid+1, end, key);
    }

}
```

Output:

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 forks. 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.

The inputs can be viewed at the program source code.


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./7
Key found at address: 10
Key found at address: 2
Key found at address: 12
Key found at address: 4
subzer0@jarvis:~/Desktop/College/OS/Lab-3$
```

Question 8: * Non Mandatory [extra credits] Read upon efficient ways of parallelizing the generation of Fibonacci series and apply the logic in a parent child relationship to contributes a faster version of fib series generation as opposed to sequential logic in (4)

Code:

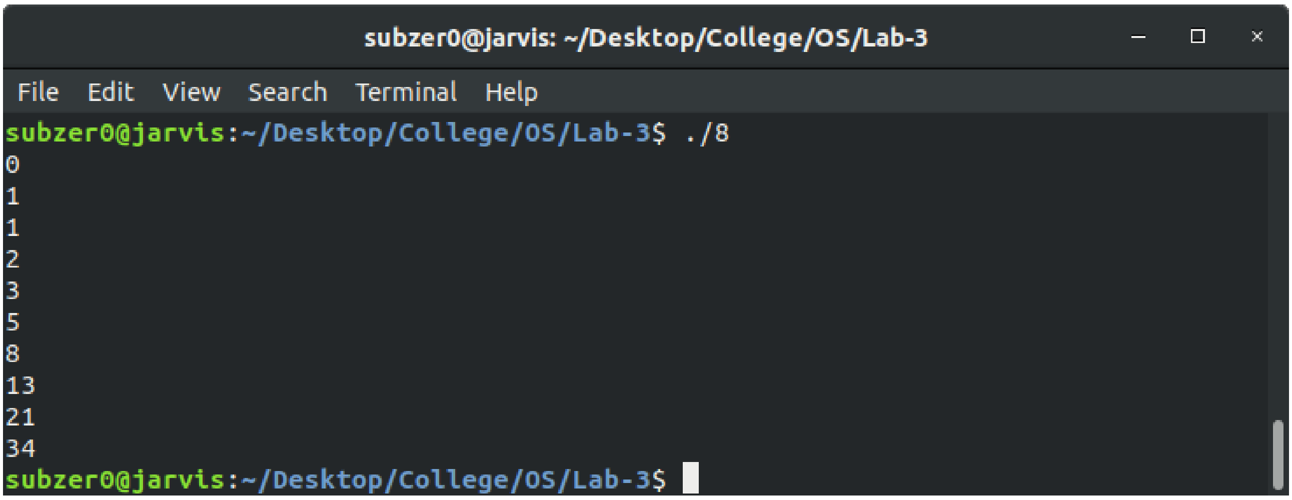
```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>

int parl_fib (int n){

   if(n == 1){
      return 0;
   }
   if(n == 2){
      return 1;
   }
   else{
      int fib_a=0, fib_b=0;

      pid_t child1 = vfork();
      if(child1 == 0){
         fib_a = parl_fib(n-1);
         exit(0);
      }
      else{
         pid_t child2 = vfork();
         if(child2 == 0){
            fib_b = parl_fib(n-2);
            exit(0);
         }
         else{
            wait(NULL);
            return fib_a + fib_b;
         }
      }

   }

}

int main (){

   printf("%d\n", parl_fib(1));
   printf("%d\n", parl_fib(2));
   printf("%d\n", parl_fib(3));
   printf("%d\n", parl_fib(4));
   printf("%d\n", parl_fib(5));
   printf("%d\n", parl_fib(6));
   printf("%d\n", parl_fib(7));
   printf("%d\n", parl_fib(8));
   printf("%d\n", parl_fib(9));
   printf("%d\n", parl_fib(10));

   return 0;

}
```

Output:

One thing to note here is that 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).


subzer0@jarvis: ~/Desktop/College/OS/Lab-3
File Edit View Search Terminal Help

```
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ ./8
0
1
1
2
3
5
8
13
21
34
subzer0@jarvis:~/Desktop/College/OS/Lab-3$ 
```