Operating System

COM301P

Programming Assignment

Lab - 3

By :

Sai Kaushik S

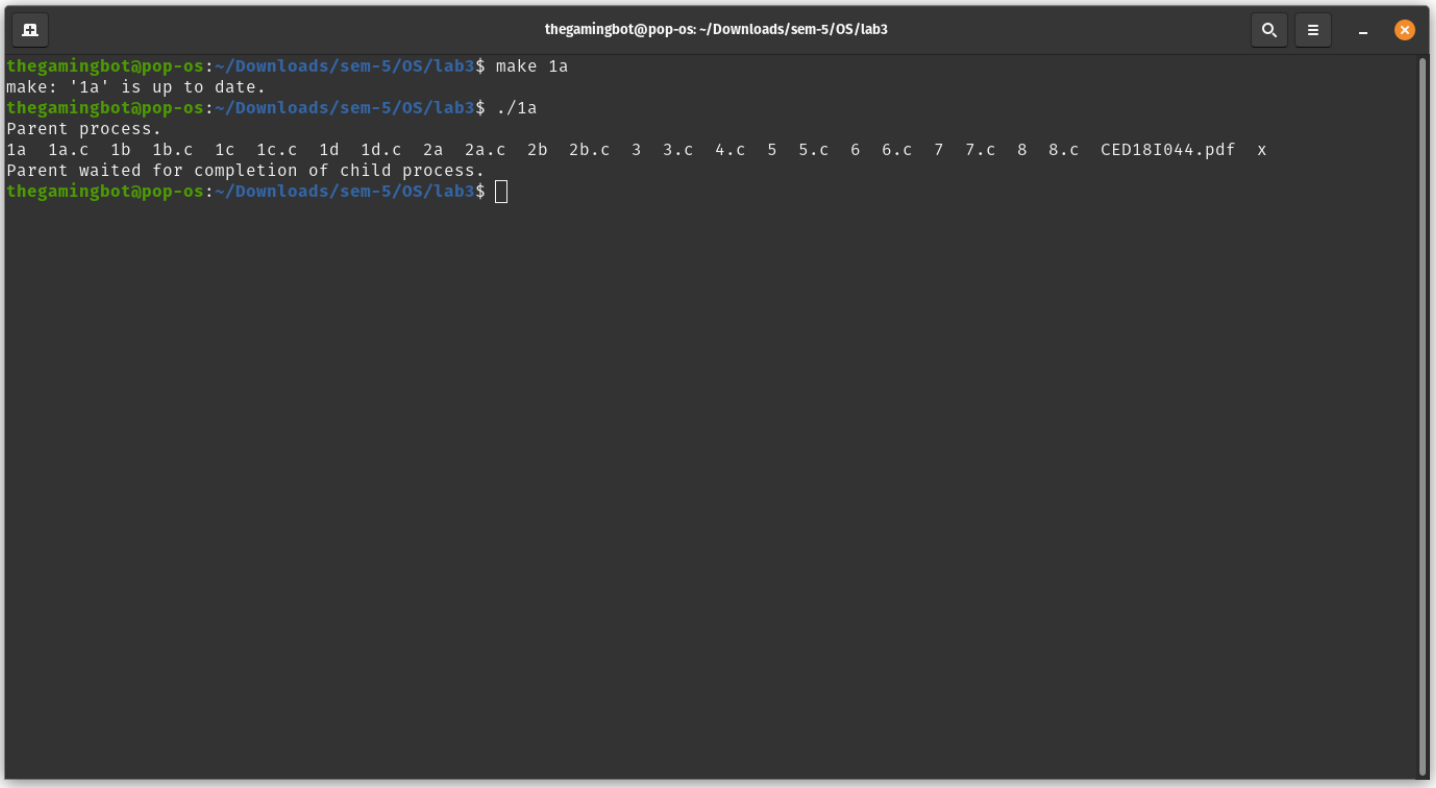
CED18I044

1. **Test Drive all the examples discussed so far in the class for the usage of wait, exec call variants. For the following questions you are free to decide the responsibility of parent / child processes. As mentioned in the class u r allowed to use vfork / file based approach to avoid the data sharing issues which we will later address using pipes in later classes to follow.**

Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Main driver function for the program
int main(){
   // Create a new process
   pid_t pid = fork();
   // If the fork failed
   if(pid < 0) printf("Fork failed.\n");
   // Child process: Executes the command ls using execl
   else if(pid == 0) execl("/bin/ls", "ls", NULL);
   // Parent process
   else {
       printf("Parent process.\n");
       // Wait for the child to complete
       wait(NULL);
       printf("Parent waited for completion of child process.\n");
   }
   exit(0);
}
```

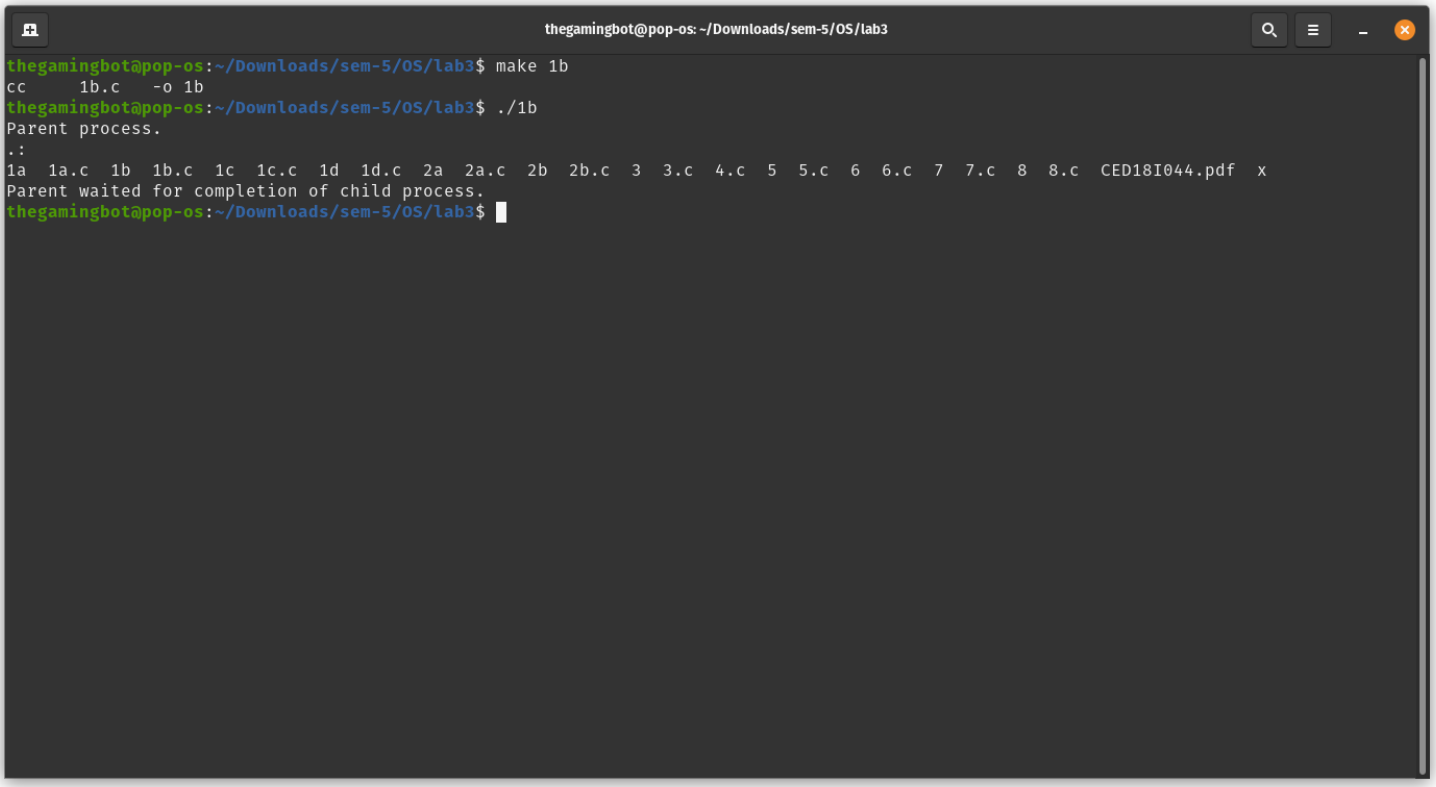
Output:



Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
// Main driver function for the program
int main(){
   // Create a new process
   pid_t pid = fork();
   // If the fork failed
   if(pid < 0) printf("Fork failed.\n");
   // Child process: Executes the command ls using execlp
   else if(pid == 0) execlp("ls", "ls", "-LR", NULL);
   // Parent process
   else {
       printf("Parent process.\n");
       // Wait for the child to complete
       wait(NULL);
       printf("Parent waited for completion of child process.\n");
   }
   exit(0);
}
```

Output:

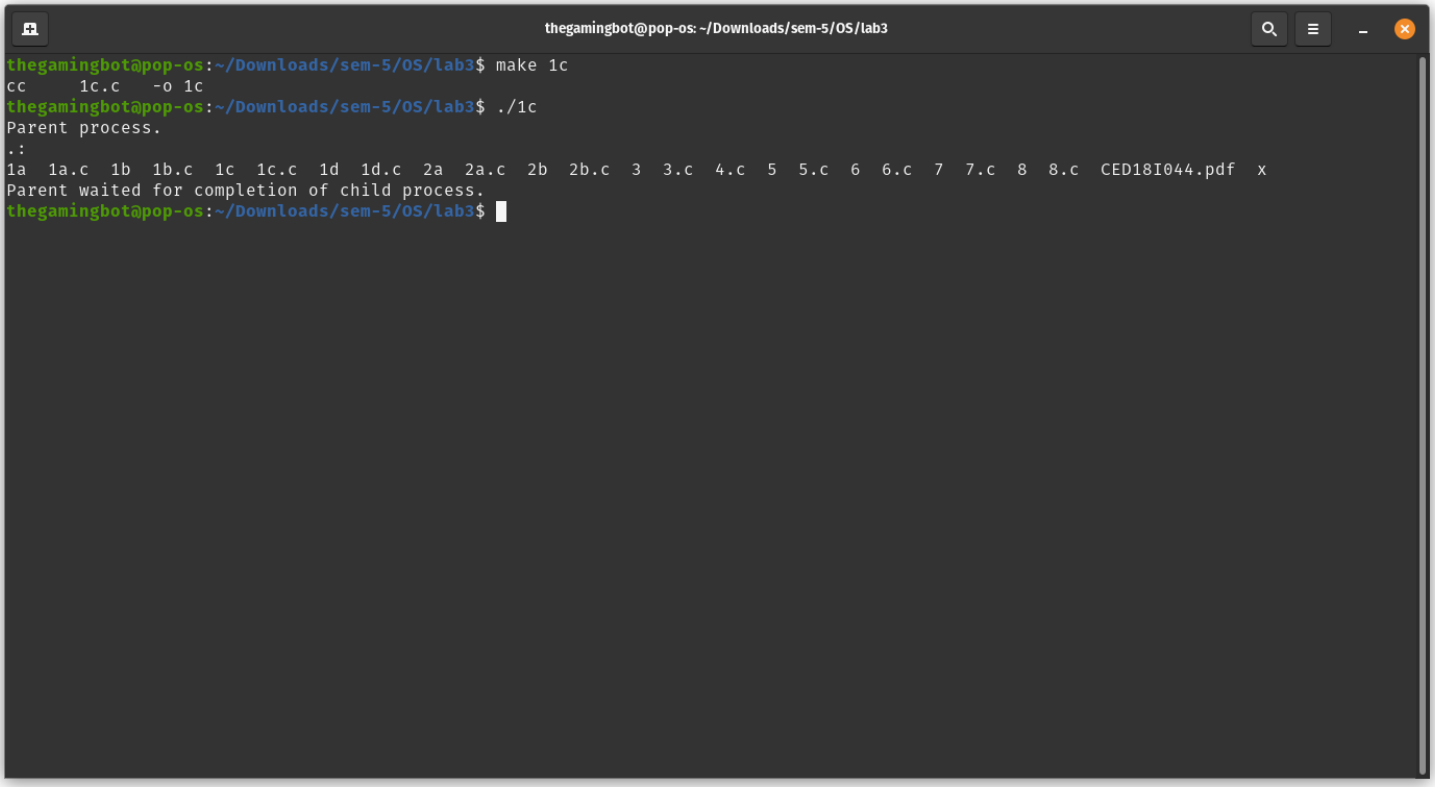


Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Main driver function for the program
int main(){
   // Arguments for the execv command
   char *args[] = {"/bin/ls", "-LR", NULL};
   // Create a new process
   pid_t pid = fork();
   // If the fork failed
   if(pid < 0) printf("Fork failed.\n");
   // Child process: Executes the command ls using execv
   else if(pid == 0) execv("/bin/ls", args);
   // Parent process
   else {
       printf("Parent process.\n");
       // Wait for the child to complete
       wait(NULL);
       printf("Parent waited for completion of child process.\n");
```

```
    }
    exit(0);
}
```

Output:



Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Main driver function for the program
int main(){
    // Arguments for the execvp command
    char *args[] = {"/bin/ls", "-LR", NULL};
    // Create a new process
    pid_t pid = fork();
    // If the fork failed
    if(pid < 0) printf("Fork failed.\n");
    // Child process: Executes the command ls using execvp
    else if(pid == 0) execvp("ls", args);
    // Parent process
    else {
        printf("Parent process.\n");
```

```c
        // Wait for the child to complete
        wait(NULL);
        printf("Parent waited for completion of child process.\n");
    }
    exit(0);
}
```

Output:

2.

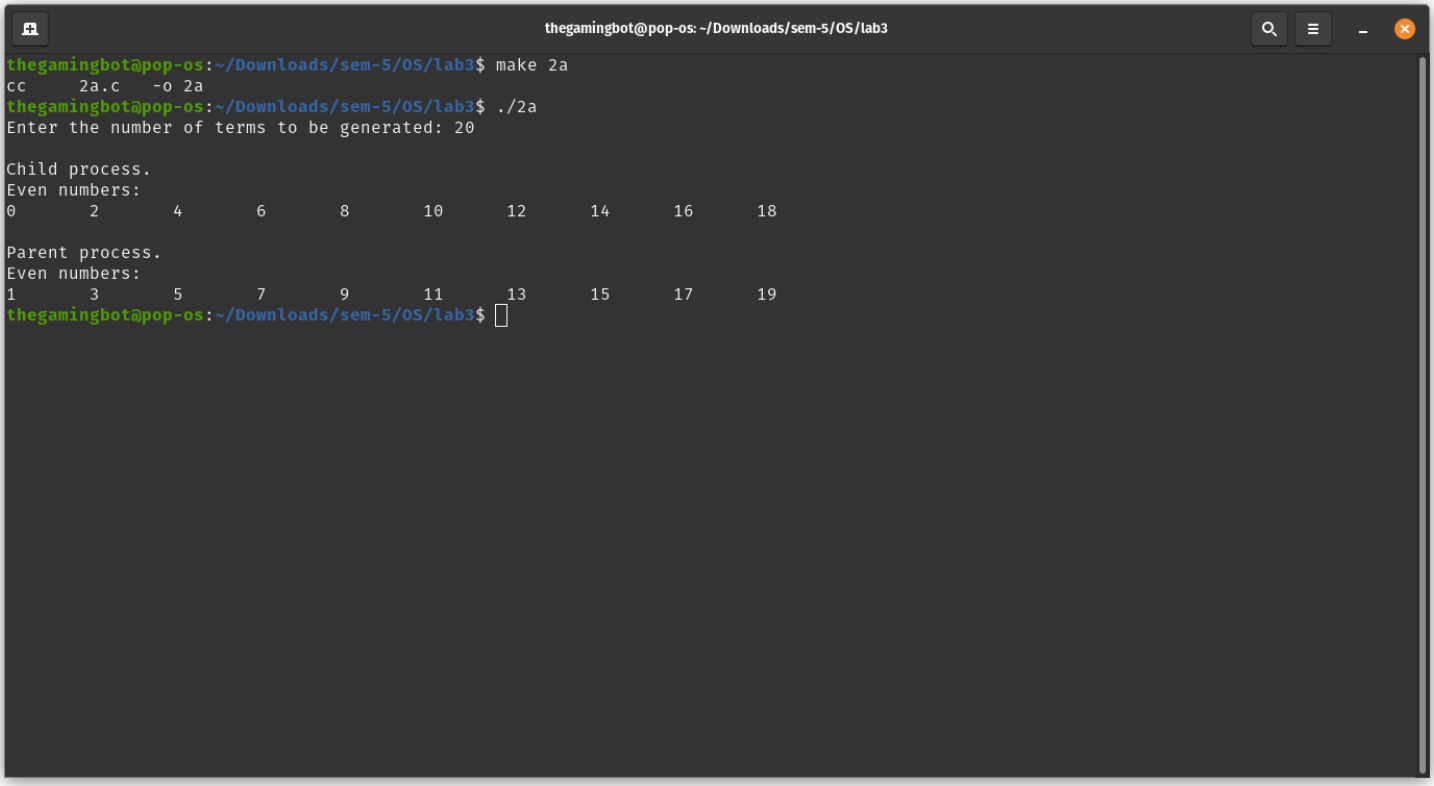
a. Odd and Even series generation for n terms using Parent Child relationship (say odd is the duty of the parent and even series as that of child)

Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Main driver function for the program
int main(){
   // Scan the number of elements to be generated
   int n;
   printf("Enter the number of terms to be generated: ");
   fflush(stdin);
   scanf("%d", &n);
   // Create a new process
   pid_t pid = fork();
   // If the fork fails
   if(pid < 0) printf("Fork failed.\n");
   // Child process
   else if(pid == 0){
       printf("\nChild process.\n");
       printf("Even numbers: \n");
       // Loop through the indices in steps of 2 from 0
       for(int i = 0; i < n; i += 2)
           // Print the even numbers
           printf("%d\t", i);
       printf("\n");
   }
   // Parent process
   else {
       // Wait for the execution of the child
       wait(NULL);
       printf("\nParent process.\n");
       printf("Even numbers: \n");
       // Loop through the indices in steps of 2 from 1
       for(int i = 1; i < n; i += 2)
           // Print the odd numbers
           printf("%d\t", i);
       printf("\n");
```

```
    }
    exit(0);
}
```

Output:



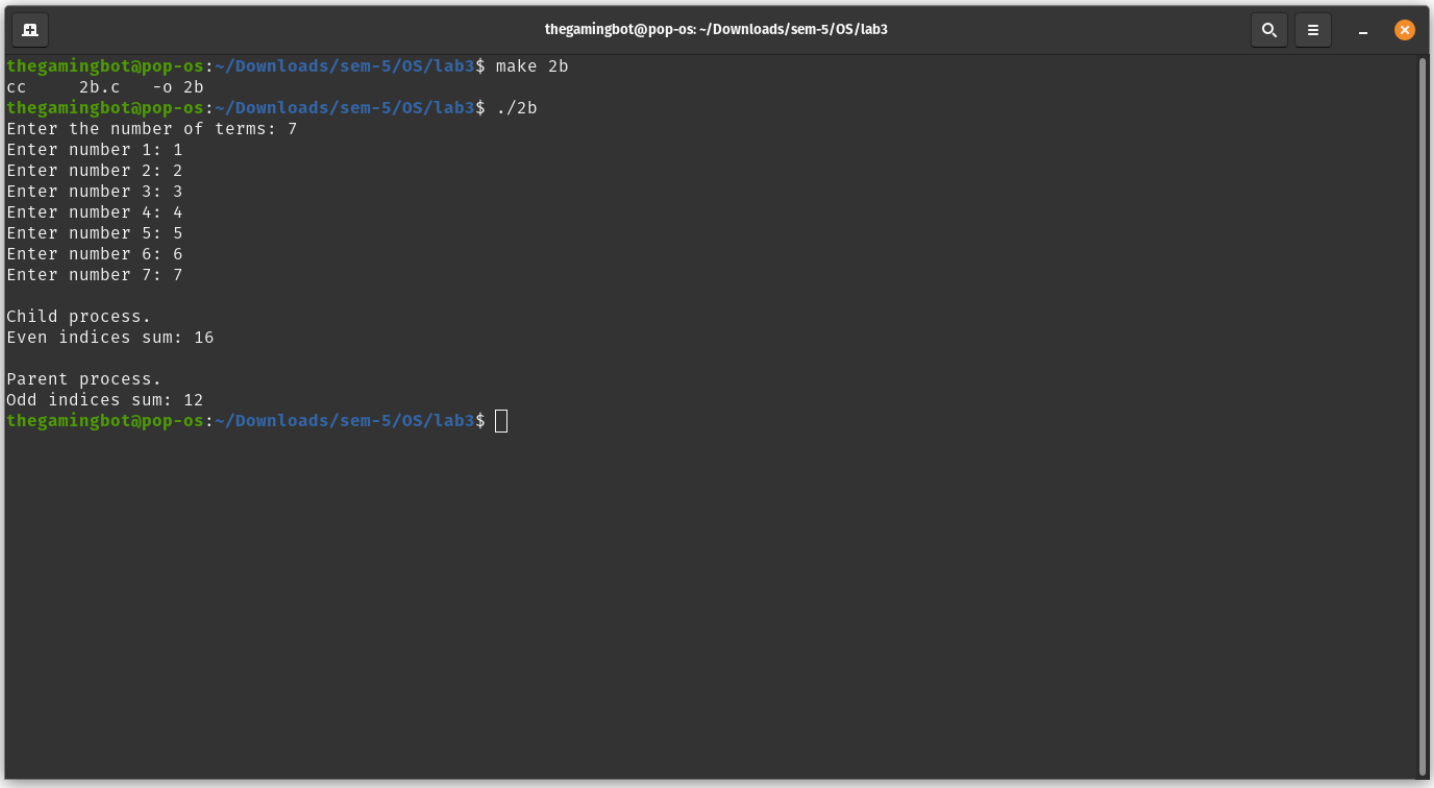
b. given a series of n numbers (u can assume natural numbers till n) generate the sum of odd terms in the parent and the sum of even terms in the child process.

Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Main driver function for the program
int main(){
    // Scan the number of elements to be generated
    int n, x;
    printf("Enter the number of terms: ");
    fflush(stdin);
    scanf("%d", &n);
    int arr[n];
    // Scan for the elements from the user
```

```c
    for(int i = 0; i < n; i++){
        printf("Enter number %d: ", i + 1);
        fflush(stdin);
        scanf("%d", &arr[i]);
    }
    // Create a new process
    pid_t pid = vfork();
    // If the fork fails
    if(pid < 0) printf("Fork failed.\n");
    // Child process
    else if(pid == 0){
        x = 0;
        printf("\nChild process.\n");
        // Loop through the array in steps of 2 from 0
        for(int i = 0; i < n; i += 2)
            // Add the numbers
            x += arr[i];
        printf("Even indices sum: %d\n", x);
    }
    // Parent process
    else {
        // Wait for the execution of the child
        wait(NULL);
        x = 0;
        printf("\nParent process.\n");
        // Loop through the indices in steps of 2 from 1
        for(int i = 1; i < n; i += 2)
            // Add the numbers
            x += arr[i];
        printf("Odd indices sum: %d\n", x);
    }
    exit(0);
}
```

Output:


thegamingbot@pop-os: ~/Downloads/sem-5/OS/lab3
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ make 2b
cc   2b.c   -o 2b
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ ./2b
Enter the number of terms: 7
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
Enter number 4: 4
Enter number 5: 5
Enter number 6: 6
Enter number 7: 7
Child process.
Even indices sum: 16
Parent process.
Odd indices sum: 12
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$

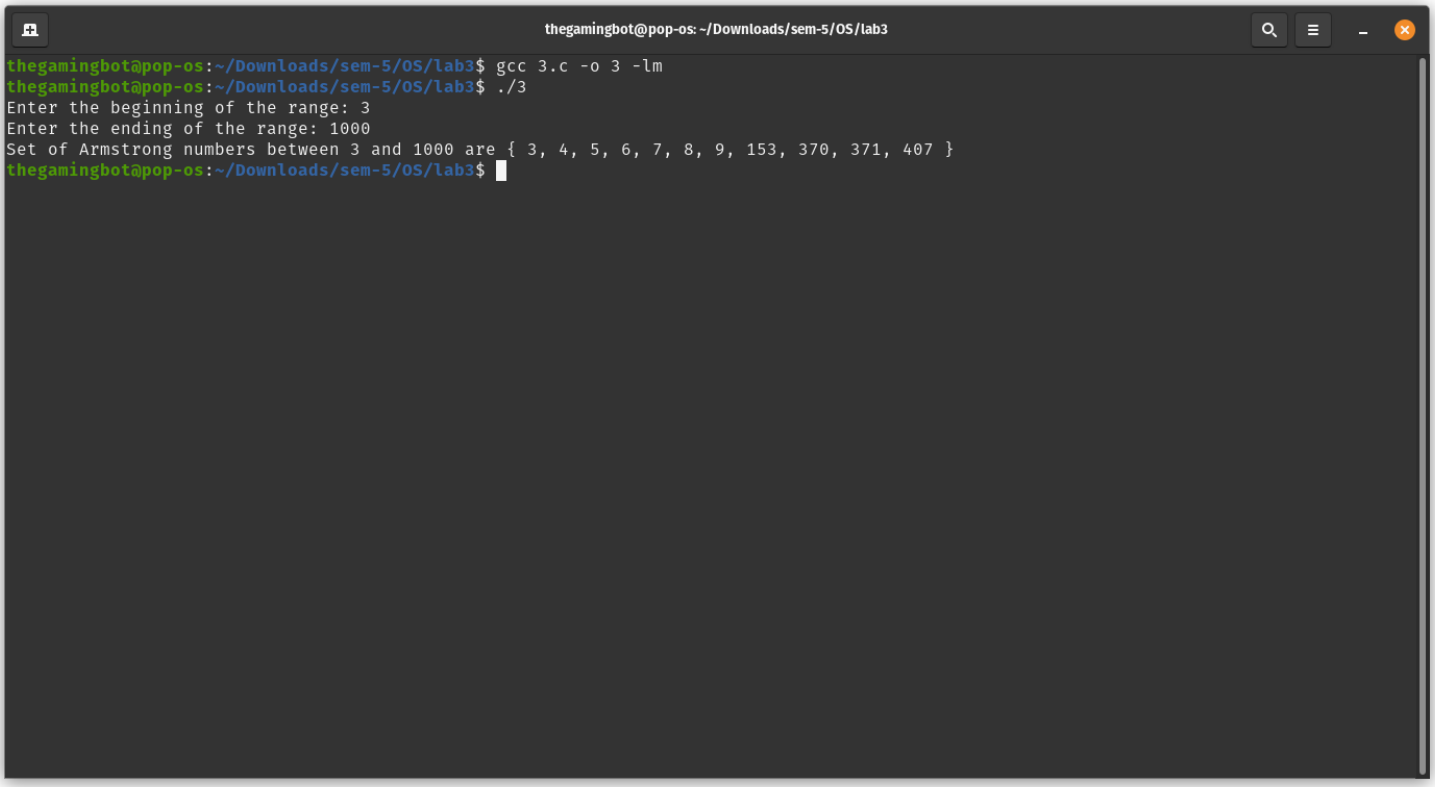
3. **Armstrong number generation within a range. The digit extraction, cubing can be the responsibility of the child while the checking for sum == no can happen in the child and the output list in the child.**

Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Main driver function for the program
int main(){
   int start, end;
   // Scan the start and end of the range
   printf("Enter the beginning of the range: ");
   fflush(stdin);
   scanf("%d", &start);
   printf("Enter the ending of the range: ");
   fflush(stdin);
   scanf("%d", &end);
   // Initialize the array to 0
   int n = end - start + 1, temp, count = 0, digit;
   int arr[n];
   for(int i = 0; i < n; i++)
       arr[i] = 0;
   // Create a new process
   pid_t pid = vfork();
   // If the fork fails
   if(pid < 0) printf("Fork failed.\n");
   // Child process
   else if(pid == 0){
       // Loop through the range
       for(int i = start; i < end + 1; i++){
           temp = i;
           // Get the count of digits
           while(temp != 0){
               temp /= 10;
               count++;
           }
           temp = i;
           // Loop through each digit
           while(temp != 0){
               digit = temp % 10;
```

```
                temp /= 10;
                // Compute the digit power count
                arr[i - start] += pow(digit, count);
            }
            count = 0;
        }
    }
    // Parent process
    else {
        // Wait for the child process to complete
        wait(NULL);
        printf("Set of Armstrong numbers between %d and %d are { ", start, end);
        // Loop through the range
        for(int i = start; i < end + 1; i++)
            // If the computer number is equal to the original number
            if(arr[i - start] == i)
                // It is an armstrong number
                printf("%d, ", i);
        printf("\b\b }\n");
    }
    exit(0);
}
```

Output:



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

Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <math.h>

// Function declarations
void primeNumberGenerator(int n);
int fib(int n);
void fibonacciSeriesGenerator(int n);

// Main driver function for the program
int main(){
   // Scan the number of elements to be generated
   int n;
   printf("Enter the number of terms to be generated: ");
   fflush(stdin);
   scanf("%d", &n);
   // Create a new process
   pid_t pid = fork();
   // If the fork fails
   if(pid < 0) printf("Fork failed.\n");
   // Child process: Generates the fibonacci series
   else if(pid == 0) fibonacciSeriesGenerator(n);
   else {
       // Wait for the child to complete
       wait(NULL);
       // Generate prime numbers less than n
       primeNumberGenerator(n);
   }
   exit(0);
}

// A function to generate prime numbers
void primeNumberGenerator(int n){
   int flag;
   printf("The set of first 'n' prime numbers are { ");
   // Loop through 2 to n
   for (int i = 2; i <= n; i++){
```

```
        flag = 1;
        // Check if there are any factors from 2 to square root of i
        for (int j = 2; j <= sqrt(i); ++j){
            // If there is a factor
            if (i % j == 0){
                // Break the loop
                flag = 0;
                break;
            }
        }
        // If flag is 1 print the number
        if (flag == 1)
            printf("%d, ", i);
    }
    printf("\b\b }\n");
}

// A function that calculates fibonacci number recursively
int fib(int n){
    // Initial numbers are 0 and 1
    if (n <= 1)
        return n;
    // Return the sum of fibonacci of n-1 and n-2
    return fib(n-1) + fib(n-2);
}

// A function to generate fibonacci series
void fibonacciSeriesGenerator(int n){
    printf("The set of first 'n' fibonacci series numbers are { ");
    // Loop through the range
    for(int i = 0; i < n; i++)
        // Print the sequence
        printf("%d, ", fib(i + 1));
    printf("\b\b }\n");
}
```

Output:


thegamingbot@pop-os: ~/Downloads/sem-5/OS/lab3
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ gcc 4.c -o 4 -lm
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ ./4
Enter the number of terms to be generated: 20
The set of first 'n' fibonacci series numbers are { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765 }
The set of first 'n' prime numbers are { 2, 3, 5, 7, 11, 13, 17, 19 }
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$

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)**

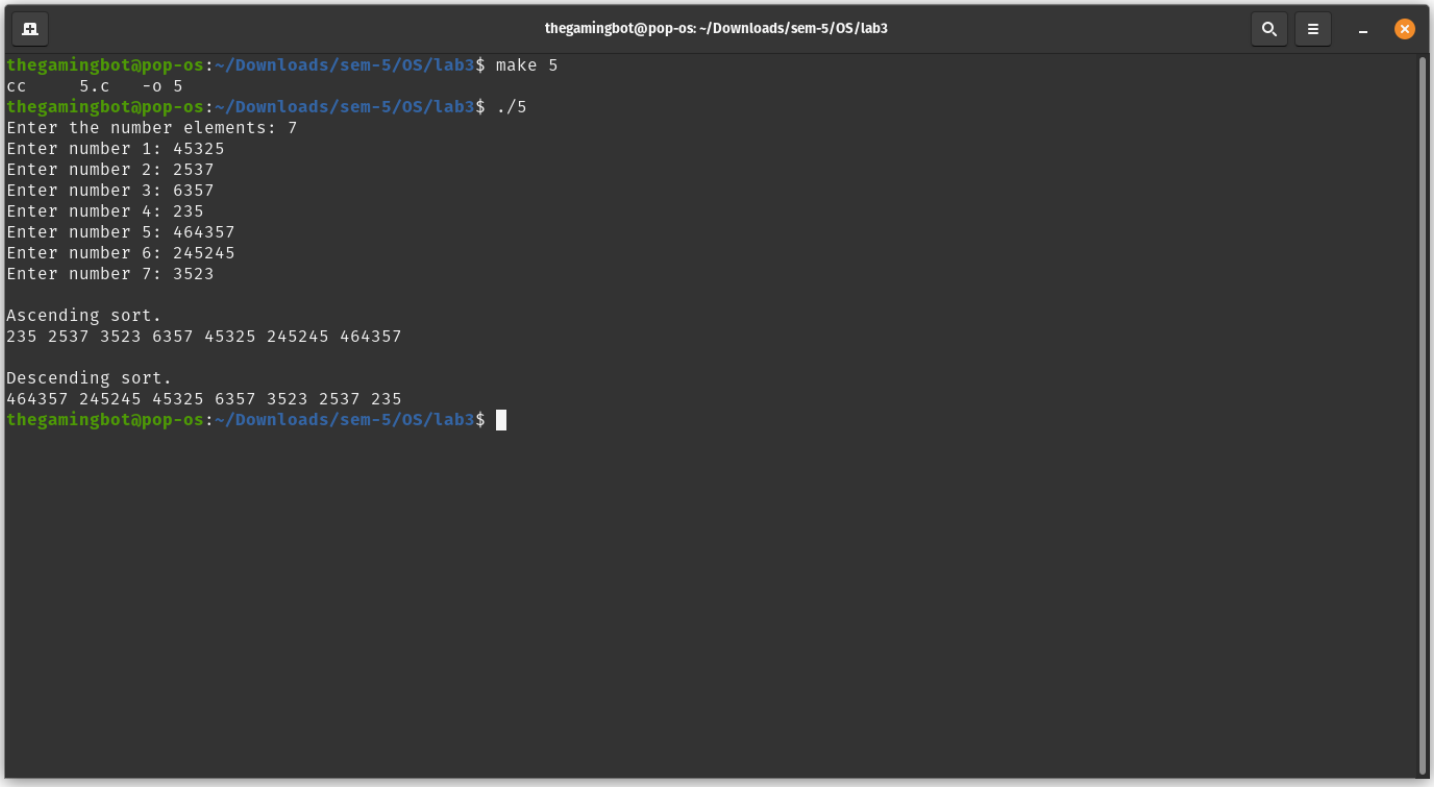
Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdbool.h>

// Function declarations
void bubbleSort(int* arr, int n, bool fun(const void*, const void*));
void swap(int* a, int* b);
void print(int* arr, int n);
bool asc(const void* a, const void* b);
bool desc(const void* a, const void* b);

// Main driver function for the program
int main(){
   // Scan the number of elements
   int n;
   printf("Enter the number elements: ");
   fflush(stdin);
   scanf("%d", &n);
   int arr[n];
   // Scan the array elements
   for(int i = 0; i < n; i++){
       printf("Enter number %d: ", i + 1);
       fflush(stdin);
       scanf("%d", &arr[i]);
   }
   // Create a new process
   pid_t pid = fork();
   // If the fork failed
   if(pid < 0) printf("Fork failed.\n");
   // Child process
   else if(pid == 0){
       printf("\nAscending sort.\n");
       // Create a new process
       pid_t child = vfork();
       // If the fork failed
```

```
        if(child < 0) printf("Fork failed.\n");
        // Child process
        else if(child == 0)
            // Bubble sort the array
            bubbleSort(arr, n, asc);
        // Parent process
        else{
            // Wait for the child process
            wait(NULL);
            // Print the sorted array
            print(arr, n);
        }
    }
    // Parent process
    else{
        // Wait for the child to complete
        wait(NULL);
        printf("\nDescending sort.\n");
        // Create a new process
        pid_t child = vfork();
        // If the fork failed
        if(child < 0) printf("Fork failed.\n");
        // Child process
        else if(child == 0)
            // Bubble sort the array
            bubbleSort(arr, n, desc);
        // Parent process
        else{
            // Wait for the child process
            wait(NULL);
            // Print the sorted array
            print(arr, n);
        }
    }
    exit(0);
}

// A function to bubble sort the array with a function pointer
void bubbleSort(int* arr, int n, bool fun(const void*, const void*)){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            // Check the output of the function pointer
            if (fun(&arr[j], &arr[j+1]))
                swap(&arr[j], &arr[j+1]);
}
```

```c
// A function to print the array
void print(int* arr, int n){
   for(int i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}

// A function that simulates the ascending sort condition
bool asc(const void* a, const void* b){
   return *(int*)a > *(int*)b;
}

// A function that simulates the descending sort condition
bool desc(const void* a, const void* b){
   return *(int*)a < *(int*)b;
}

// A function to swap two numbers
void swap(int* a, int* b){
   int c = *a;
   *a = *b;
   *b = c;
}
```

Output:



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)**

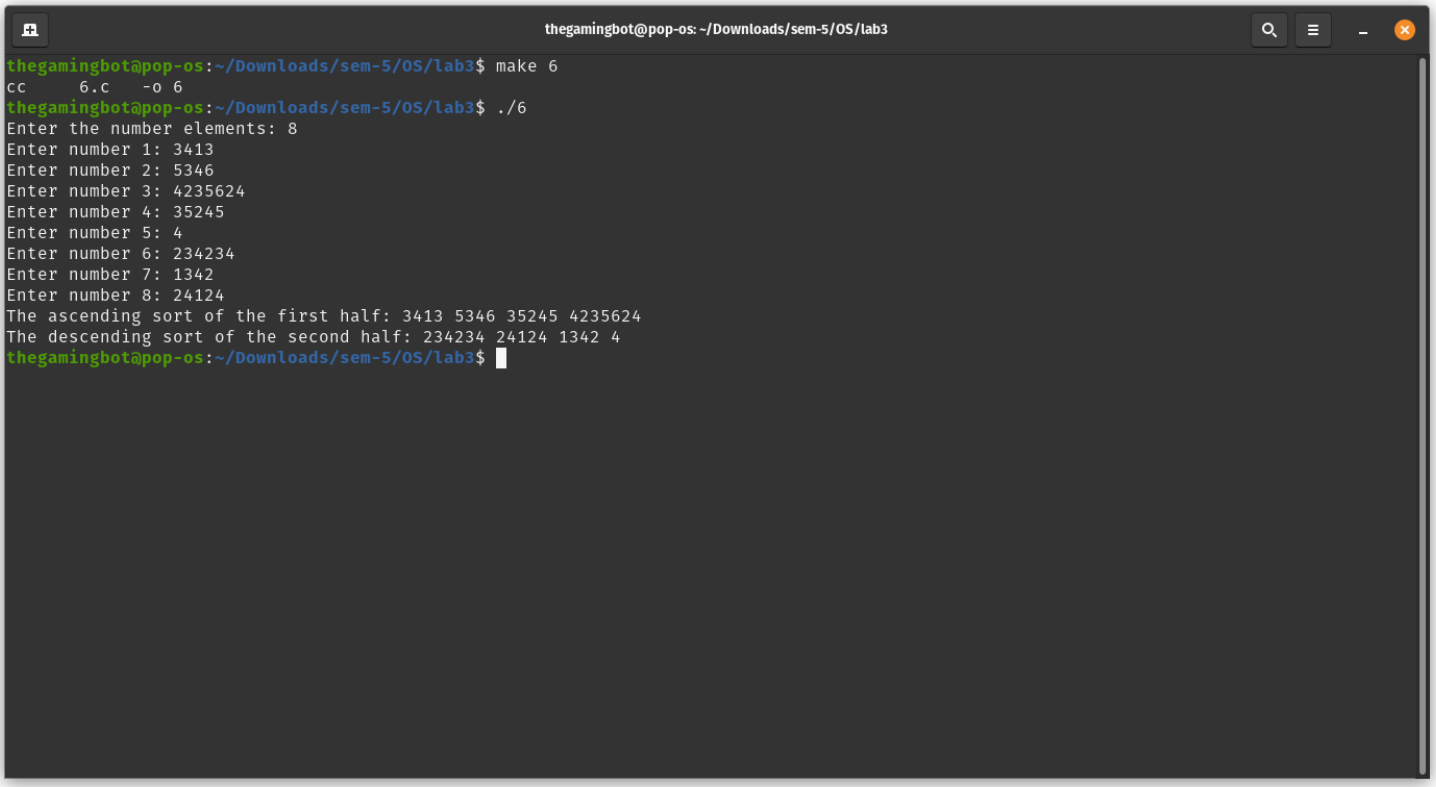
Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdbool.h>

// Function declarations
void bubbleSort(int* arr, int n, bool fun(const void*, const void*));
void swap(int* a, int* b);
void print(int* arr, int n);
bool asc(const void* a, const void* b);
bool desc(const void* a, const void* b);

// Main driver function for the program
int main(){
   int n, mid, x = 0;
   // Scan the number of elements
   printf("Enter the number elements: ");
   fflush(stdin);
   scanf("%d", &n);
   mid = n / 2;
   int arr1[mid], arr2[n - mid];
   // Scan the array elements
   for(int i = 0; i < mid; i++){
       printf("Enter number %d: ", ++x);
       fflush(stdin);
       scanf("%d", &arr1[i]);
   }
   for(int i = 0; i < n - mid; i++){
       printf("Enter number %d: ", ++x);
       fflush(stdin);
       scanf("%d", &arr2[i]);
   }
   // Create a new process
   pid_t pid = fork();
   // If the fork fails
   if(pid < 0) printf("Fork failed.\n");
   // Child process
```

```
    else if(pid == 0){
        printf("The ascending sort of the first half: ");
        // Create a new process
        pid_t child = vfork();
        // If the fork fails
        if(child < 0) printf("Fork failed.\n");
        // Child process
        else if(child == 0)
            // Bubble sort the first half of the array
            bubbleSort(arr1, mid, asc);
        // Parent process
        else{
            // Wait for the child to complete
            wait(NULL);
            // Print the array
            print(arr1, mid);
        }
    }
    // Parent process
    else{
        wait(NULL);
        printf("The descending sort of the second half: ");
        // Create a new process
        pid_t parent = vfork();
        // If the fork fails
        if(parent < 0) printf("Fork failed.\n");
        // Child process
        else if(parent == 0)
            // Bubble sort the second half of the array
            bubbleSort(arr2, n - mid, desc);
        // Parent process
        else{
            // Wait for the child to complete
            wait(NULL);
            // Print the array
            print(arr2, n - mid);
        }
    }
    exit(0);
}

// A function to bubble sort the array with a function pointer
void bubbleSort(int* arr, int n, bool fun(const void*, const void*)){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
```

```
            // Check the output of the function pointer
            if (fun(&arr[j], &arr[j+1]))
                swap(&arr[j], &arr[j+1]);
}

// A function to print the array
void print(int* arr, int n){
   for(int i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}

// A function that simulates the ascending sort condition
bool asc(const void* a, const void* b){
   return *(int*)a > *(int*)b;
}

// A function that simulates the descending sort condition
bool desc(const void* a, const void* b){
   return *(int*)a < *(int*)b;
}

// A function to swap two numbers
void swap(int* a, int* b){
   int c = *a;
   *a = *b;
   *b = c;
}
```

Output:


thegamingbot@pop-os: ~/Downloads/sem-5/OS/lab3
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ make 6
cc   6.c   -o 6
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ ./6
Enter the number elements: 8
Enter number 1: 3413
Enter number 2: 5346
Enter number 3: 4235624
Enter number 4: 35245
Enter number 5: 4
Enter number 6: 234234
Enter number 7: 1342
Enter number 8: 24124
The ascending sort of the first half: 3413 5346 35245 4235624
The descending sort of the second half: 234234 24124 1342 4
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$

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)**

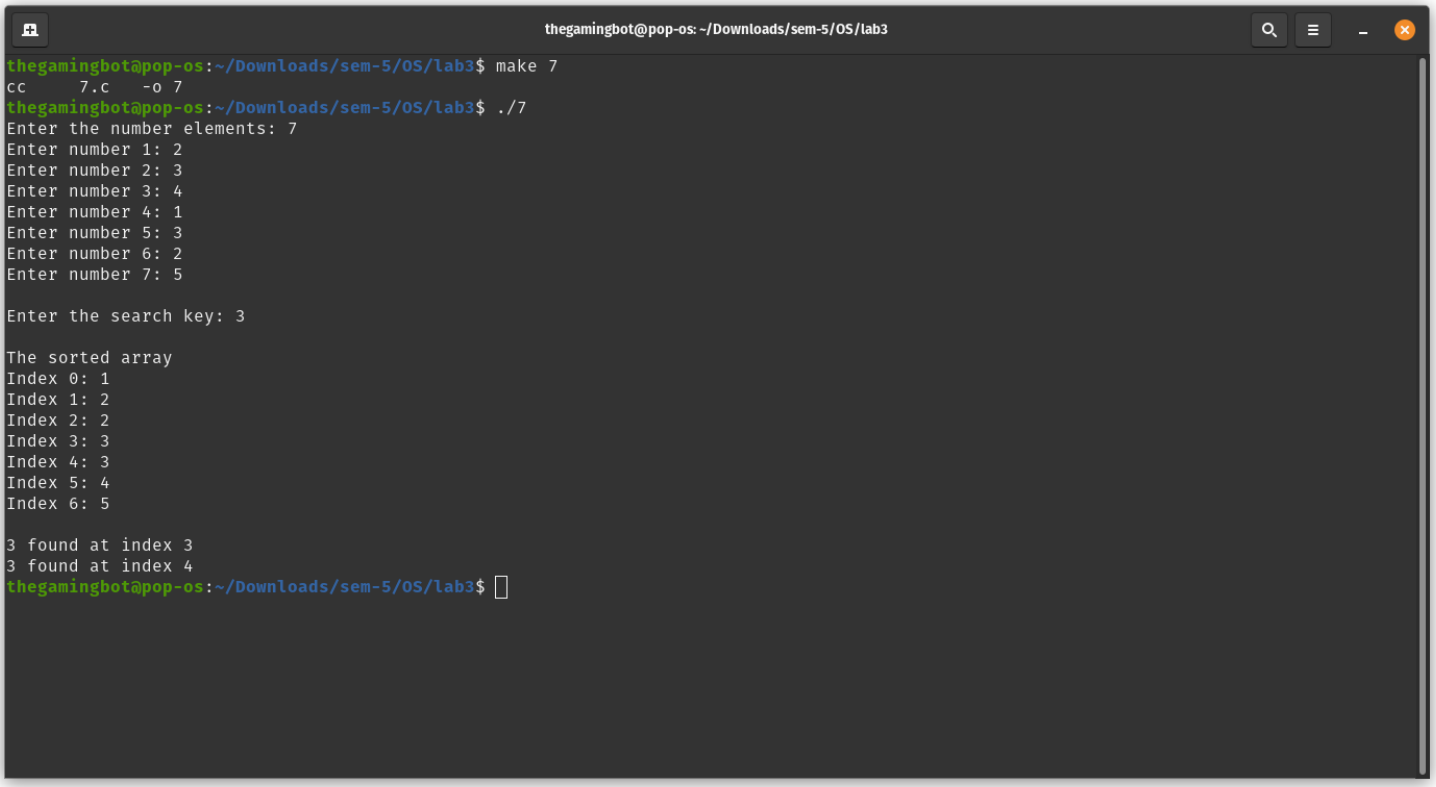
Code:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdbool.h>

// Function declarations
int binarysearch(int *a, int n, int x);
void bubbleSort(int *arr, int n);
void swap(int *a, int *b);
void print(int *arr, int n);

// Main driver function for the program
int main(){
   // Scan the number of elements
   int n, mid, x, count = 0;
   printf("Enter the number elements: ");
   fflush(stdin);
   scanf("%d", &n);
   int arr[n];
   // Scan the array elements
   for (int i = 0; i < n; i++){
       printf("Enter number %d: ", i + 1);
       fflush(stdin);
       scanf("%d", &arr[i]);
   }
   // Scan the search element
   printf("\nEnter the search key: ");
   fflush(stdin);
   scanf("%d", &x);
   // Sort the given array
   bubbleSort(arr, n);
   printf("\nThe sorted array\n");
   fflush(stdin);
   print(arr, n);
```

```c
    // Create a new process
    pid_t pid = vfork();
    // If the fork fails
    if (pid < 0) printf("Fork failed.\n");
    // Child process
    else if (pid == 0){
        // If the search key is present in the array
        if ((mid = binarysearch(arr, n, x)) == -1){
            printf("%d is not found in the array\n", x);
            exit(1);
        }
        // Id the search key is absent in the array
        else
            printf("\n%d found at index %d\n", x, mid);
        exit(0);
    }
    // Parent process
    else{
        // Wait for the child process
        wait(NULL);
        int i = mid - 1;
        // Loop through the left of the search element to find the duplicates
        while (i > -1 && arr[i] == x){
            printf("%d found at index %d\n", x, i);
            i--;
        }
        i = mid + 1;
        // Loop through the right of the search element to find the duplicates
        while (i < n && arr[i] == x){
            printf("%d found at index %d\n", x, i);
            i++;
        }
    }
    exit(0);
}

// A function to implement binary search
int binarysearch(int *a, int n, int x){
    int l = 0, r = n - 1;
    // If there exists a number in the range
    while (l <= r){
        // Get the nid of the range
        int m = l + (r - l) / 2;
        // If the mid index is the key
        if (a[m] == x)
```

```
            return m;
        // If the array element is less than key
        else if (a[m] < x)
            // Update the lower range
            l = m + 1;
        // If the array element is greater than key
        else
            // Update the higher range
            r = m - 1;
    }
    return -1;
}

// A function to sort the array
void bubbleSort(int *arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}

// A function to swap two numbers
void swap(int *a, int *b){
    int c = *a;
    *a = *b;
    *b = c;
}

// A function to print the array
void print(int *arr, int n){
    for (int i = 0; i < n; i++)
        printf("Index %d: %d\n", i, arr[i]);
}
```

Output:


thegamingbot@pop-os: ~/Downloads/sem-5/OS/lab3
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ make 7
cc   7.c   -o 7
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$ ./7
Enter the number elements: 7
Enter number 1: 2
Enter number 2: 3
Enter number 3: 4
Enter number 4: 1
Enter number 5: 3
Enter number 6: 2
Enter number 7: 5
Enter the search key: 3
The sorted array
Index 0: 1
Index 1: 2
Index 2: 2
Index 3: 3
Index 4: 3
Index 5: 4
Index 6: 5
3 found at index 3
3 found at index 4
thegamingbot@pop-os:~/Downloads/sem-5/OS/lab3$

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:

```c
// Include the required libraries
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>

// Function declarations
int cache[100];
int fib(int n);

// Main driver function for the program
int main (){
   // Scan the number of elements to be generated
   int n;
   printf("Enter the number of terms to be generated: ");
   fflush(stdin);
   scanf("%d", &n);
   memset(cache, -1, 100);
   // Calculate fib of n
   fib(n);
   // Print the cache i.e. the fibonacci sequence
   printf("The set of first 'n' fibonacci series numbers are { ");
   for(int i = 0; i < n; i++)
       printf("%d, ", cache[i + 1]);
   printf("\b\b }\n");
   exit(0);
}

// A function to generate fibonacci of n
int fib(int n){
   int temp = 0;
   // If the fib of n is not yet computed
   if (cache[n] == -1){
       // Generate the initial numbers, i.e. 0 and 1
       if (n <= 1)
           cache[n] = n;
       else{
```

```
            // Create a new process
            pid_t pid = vfork();
            // If the fork fails
            if(pid < 0) printf("Fork failed.\n");
            // Child process
            else if(pid == 0){
                // Compute fib of n-1
                temp += fib(n - 1);
                exit(0);
            }
            // Parent process
            else {
                // Wait for the child to execute
                wait(NULL);
                // Compute fib of n-1
                temp += fib(n - 2);
            }
            // Update the cache
            cache[n] = temp;
        }
    }
    return cache[n];
}
```

Output: