HARISH K M
COE18B066

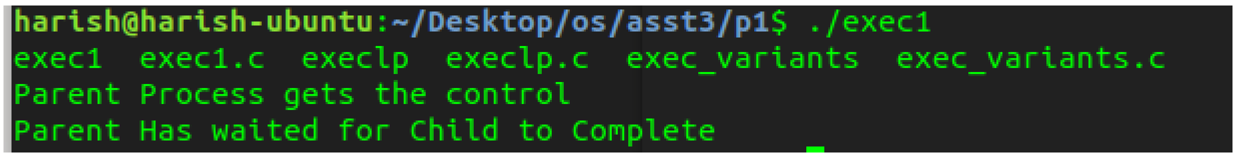
Operating System
Assignment-3

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

Execl:

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

Output:



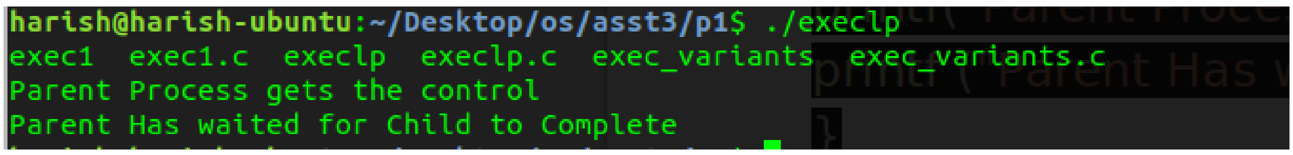
```
harish@harish-ubuntu:~/Desktop/os/asst3/p1$ ./exec1
exec1   exec1.c   execlp   execlp.c   exec_variants   exec_variants.c
Parent Process gets the control
Parent Has waited for Child to Complete
```

Execlp:

```
# include <stdio.h>
```

```
#include <sys/types.h>
#include<sys/wait.h>
#include <unistd.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");
}
}
```

Output:

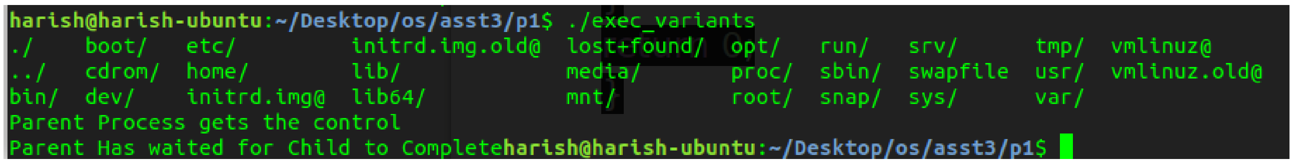


```
harish@harish-ubuntu:~/Desktop/os/asst3/p1$ ./execlp
exec1   exec1.c   execlp   execlp.c   exec_variants   exec_variants.c
Parent Process gets the control
Parent Has waited for Child to Complete
```

Exec variants:

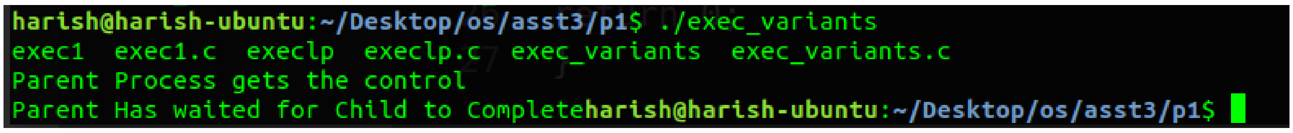
```
# include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main ()
{
pid_t pid; // this is to use the pid data type - relevant headers above
pid = fork();
if (pid == 0)
{//execlp("ls", "ls", NULL); // child image is now ls command

//char *const argv[] = {"/bin/ls","-l", NULL};
//execv(argv[0], argv);
//char *const argv[] = {"/bin/ls", NULL};
//execvp(argv[0], argv);
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");
}
return 0;
}
```

Execve output:



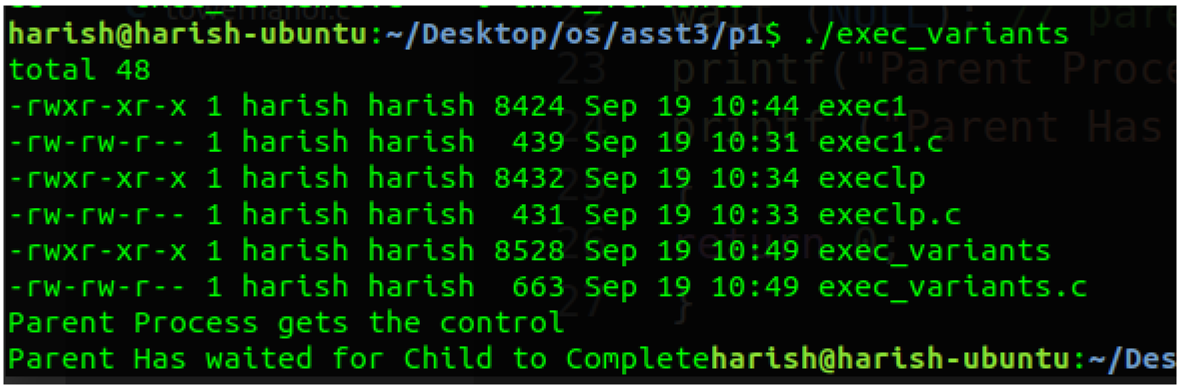
```
harish@harish-ubuntu:~/Desktop/os/asst3/p1$ ./exec_variants
./    boot/   etc/         initrd.img.old@  lost+found/  opt/   run/   srv/       tmp/   vmlinuz@
../   cdrom/  home/        lib/             media/       proc/  sbin/  swapfile   usr/   vmlinuz.old@
bin/  dev/    initrd.img@  lib64/           mnt/         root/  snap/  sys/       var/
Parent Process gets the control
Parent Has waited for Child to Completeharish@harish-ubuntu:~/Desktop/os/asst3/p1$
```

Execvp output:



```
harish@harish-ubuntu:~/Desktop/os/asst3/p1$ ./exec_variants
exec1  exec1.c  execlp  execlp.c  exec_variants  exec_variants.c
Parent Process gets the control
Parent Has waited for Child to Completeharish@harish-ubuntu:~/Desktop/os/asst3/p1$
```

Execv output:



```
harish@harish-ubuntu:~/Desktop/os/asst3/p1$ ./exec_variants
total 48
-rwxr-xr-x 1 harish harish 8424 Sep 19 10:44 exec1
-rw-rw-r-- 1 harish harish  439 Sep 19 10:31 exec1.c
-rwxr-xr-x 1 harish harish 8432 Sep 19 10:34 execlp
-rw-rw-r-- 1 harish harish  431 Sep 19 10:33 execlp.c
-rwxr-xr-x 1 harish harish 8528 Sep 19 10:49 exec_variants
-rw-rw-r-- 1 harish harish  663 Sep 19 10:49 exec_variants.c
Parent Process gets the control
Parent Has waited for Child to Completeharish@harish-ubuntu:~/Des
```

(2) 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)

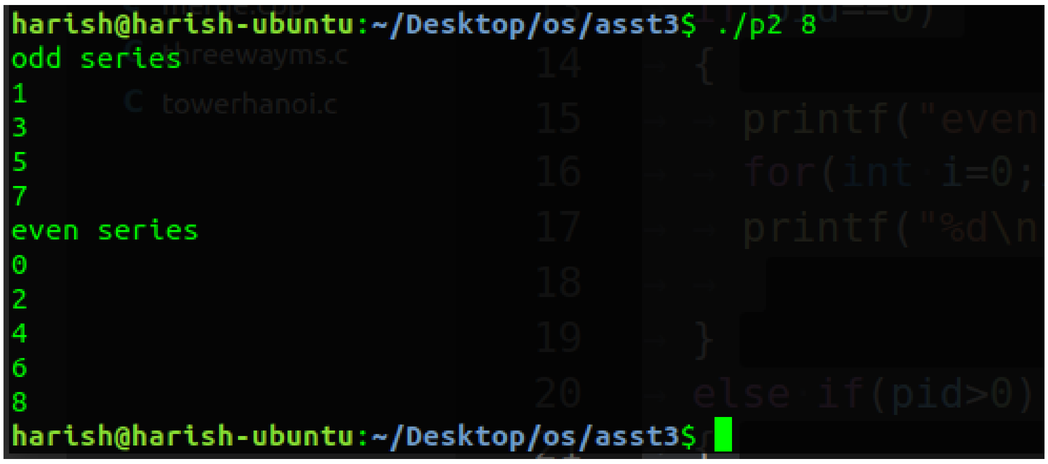
Approach:

Write codes separately for odd series and even series and insert odd series into parent block and even series into child block.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc,char *argv[])
{
if(argc!=2) {printf("invalid input\n");exit(0);}
int n=atoi(argv[1]);
pid_t pid;
pid=fork();

if(pid==0)
{
printf("even series\n");
for(int i=0;i<=n;i=i+2)
printf("%d\n",i);
}
else if(pid>0)
{
printf("odd series\n");
for(int i=1;i<=n;i=i+2)
printf("%d\n",i);
}
return 0;
}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p2 8
odd series
1
3
5
7
even series
0
2
4
6
8
harish@harish-ubuntu:~/Desktop/os/asst3$
```

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.

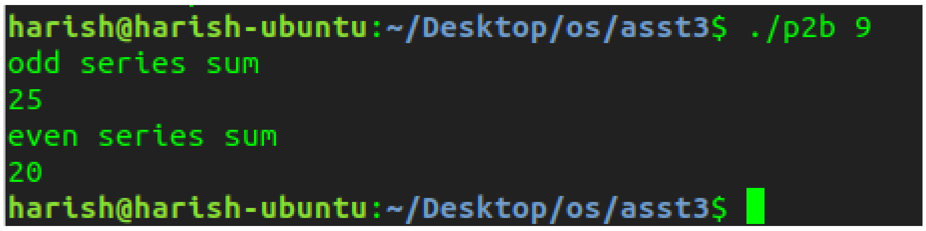
Approach:

Write codes separately for sum of odd series and even series and insert sum of odd series into parent block and sum of even series into child block.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(int argc,char *argv[])
{
if(argc!=2) {printf("invalid input\n");exit(0);}
int n=atoi(argv[1]);
pid_t pid;
pid=fork();

if(pid==0)
{
int sume=0;
printf("even series sum\n");
for(int i=0;i<=n;i=i+2)
sume+=i;
printf("%d\n",sume);
}
else if(pid>0)
{
int sumo=0;
printf("odd series sum\n");
for(int i=1;i<=n;i=i+2)
sumo+=i;
printf("%d\n",sumo);
}
return 0;
}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p2b 9
odd series sum
25
even series sum
20
harish@harish-ubuntu:~/Desktop/os/asst3$ 
```

(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.

Approach:

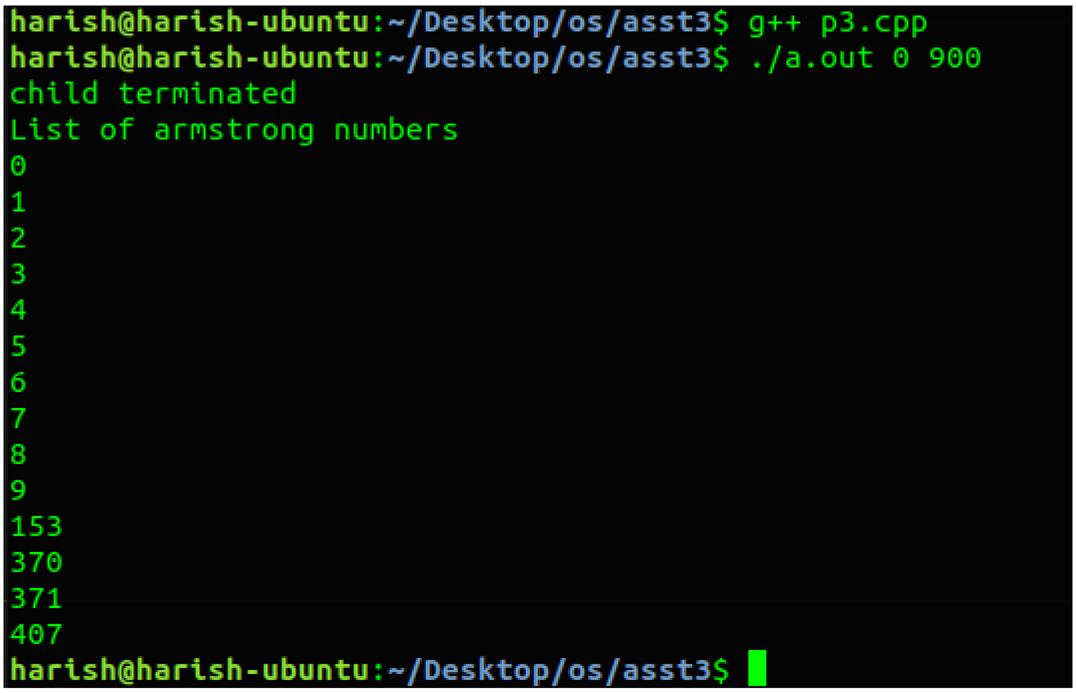
For every number in the given range the sum of its digits raised to the power of number of digits is calculated in child block and stored in an array. Once the child process gets terminated the parent starts where the calculated sum is compared with its respective number and if condition is true the number is printed.

Code:

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

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

```
if(argc!=3) {printf("invalid input\n");exit(0);}
int lnum=atoi(argv[1]),hnum=atoi(argv[2]);
int sum[1000]={0};
pid_t pid;
pid=vfork();

if(pid==0)
{
for(int i=lnum;i<hnum;++i)
{
int k,n=i;
int count=0;
while(n>0)
{
n/=10;
count++;
}
n=i;
while(n>0)
{
k=n%10;
n=n/10;
sum[i-lnum]+=pow(k,count);
}
}
exit(0);
}
else if(pid>0)
{
wait(NULL);
printf("child terminated\nList of armstrong numbers\n");
for(int i=lnum;i<hnum;++i)
if(sum[i-lnum]==i)
printf("%d\n",i);
}
return 0;
}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ g++ p3.cpp
harish@harish-ubuntu:~/Desktop/os/asst3$ ./a.out 0 900
child terminated
List of armstrong numbers
0
1
2
3
4
5
6
7
8
9
153
370
371
407
harish@harish-ubuntu:~/Desktop/os/asst3$ 
```

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

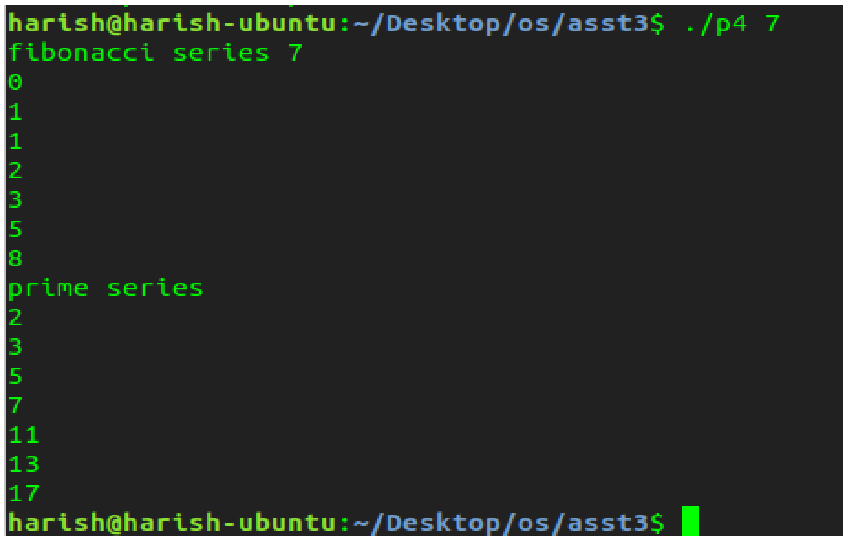
Approach:

Write codes separately for fibonacci series and prime series and insert fibonacci series into child block and prime series into parent block.

Code:

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


int main(int argc,char *argv[])
{
if(argc!=2) {printf("invalid input\n");exit(0);}
int n=atoi(argv[1]);
pid_t pid;
pid=fork();

if(pid==0)
{
printf("fibonacci series %d\n",n);
int a=0,b=1,c=0;
for(int i=0;i<n;i++)
{printf("%d\n",a);
c=a+b;
a=b;
b=c;
}
}
else if(pid>0)
{
wait(NULL);
printf("prime series\n");
int count=0,i=2;
while(count<n)
{
int flag=1;
for(int j=2;j<i;++j)
{if(i%j==0)
flag=0;
}
if(flag==1)
{
printf("%d\n",i);
count++;
}
i++;
}
}
return 0;
}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p4 7
fibonacci series 7
0
1
1
2
3
5
8
prime series
2
3
5
7
11
13
17
harish@harish-ubuntu:~/Desktop/os/asst3$
```

(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)

Approach:

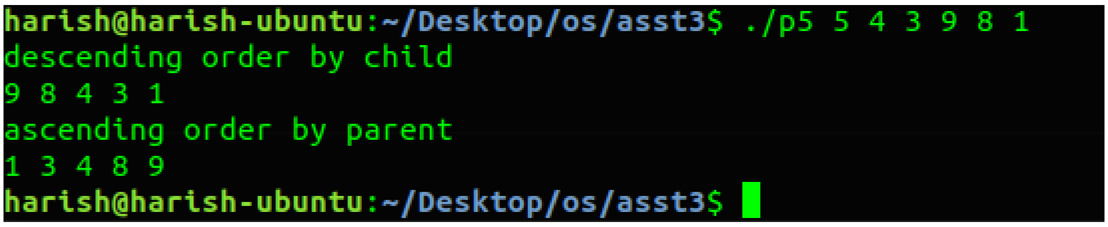
Descending sort function is done in child and ascending sort is done in parent and the obtained array is printed.

Code:

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

int main(int argc,char *argv[])
{
int num=atoi(argv[1]);
if(argc!=num+2) {printf("invalid input\n");exit(0);}
int arr[20];
for(int i=0;i<num;++i)
{
```

```
arr[i]=atoi(argv[i+2]);
}

pid_t pid;
pid=fork();

if(pid<0){printf("fork failed\n");exit(0);}

if(pid==0)
{
for (int i = 1; i < num; i++)
{
int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j]<key)
{
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
printf("descending order by child\n");
for(int i=0;i<num;++i)
printf("%d ",arr[i]);
printf("\n");
}

else if(pid>0)
{
wait(NULL);
for (int i = 1; i < num; i++)
{
int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j]>key)
{
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
printf("ascending order by parent\n");
for(int i=0;i<num;++i)
printf("%d ",arr[i]);
printf("\n");
}

return 0;

}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p5 5 4 3 9 8 1
descending order by child
9 8 4 3 1
ascending order by parent
1 3 4 8 9
harish@harish-ubuntu:~/Desktop/os/asst3$
```

(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)

Approach:

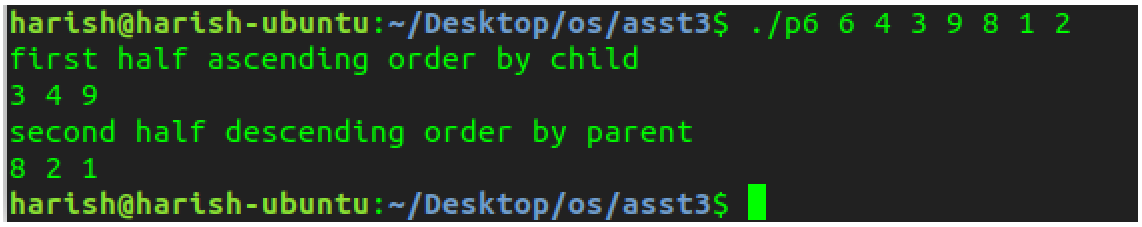
Descending order sort of first n/2 elements done in child and the ascending order sort of rest of the element done in parent and is printed.

Code:

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

```
int main(int argc,char *argv[])
{
int num=atoi(argv[1]);
if(argc!=num+2) {printf("invalid input\n");exit(0);}
//int sum=0;
int arr[20];
for(int i=0;i<num;++i)
{
arr[i]=atoi(argv[i+2]);
}

pid_t pid;
pid=fork();

if(pid<0){printf("fork failed\n");exit(0);}

if(pid==0)
{
for (int i = 1; i < num/2; i++)
{
int key = arr[i];
int j = i - 1;
while (j >= 0 && arr[j]>key)
{
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
printf("first half ascending order by child\n");
for(int i=0;i<num/2;++i)
printf("%d ",arr[i]);
printf("\n");
}
else if(pid>0)
{
wait(NULL);
for (int i = num/2+1; i < num; i++)
{
int key = arr[i];
int j = i - 1;
while (j >= num/2 && arr[j]<key)
{
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
printf("second half descending order by parent\n");
for(int i=num/2;i<num;++i)
printf("%d ",arr[i]);
printf("\n");
}
return 0;
}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p6 6 4 3 9 8 1 2
first half ascending order by child
3 4 9
second half descending order by parent
8 2 1
harish@harish-ubuntu:~/Desktop/os/asst3$
```

(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)

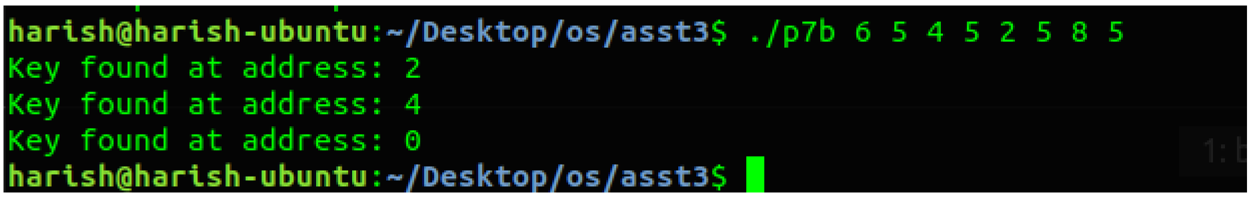
Approach:

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.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
#include<math.h>
void binary_search (int arr[], int start, int end, int key);
int main(int argc,char *argv[])
{
int num=atoi(argv[1]);
if(argc!=num+3) {printf("invalid input\n");exit(0);}
int arr[20];
for(int i=0;i<num;++i)
{
arr[i]=atoi(argv[i+2]);
}
int x=atoi(argv[num+2]);
binary_search(arr,0,num,x);
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:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p7b 6 5 4 5 2 5 8 5
Key found at address: 2
Key found at address: 4
Key found at address: 0
harish@harish-ubuntu:~/Desktop/os/asst3$
```

(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)

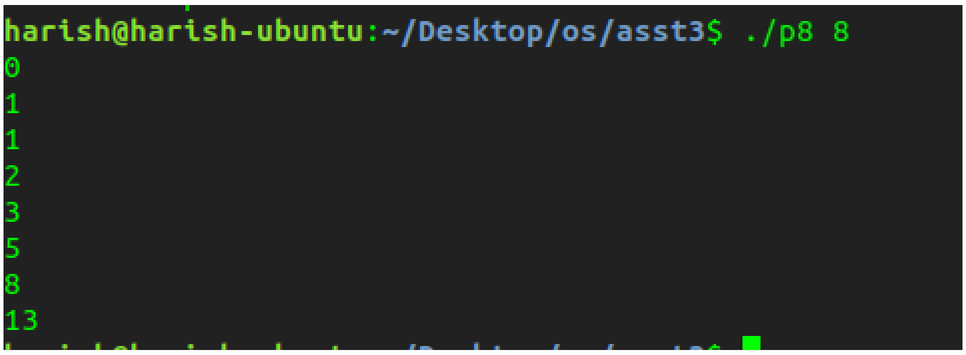
Approach:
To calculate the nth element, its previous elements fib(n-1) and fib(n-2) have to be calculated. So inorder to make this parallel, we calculate fib(n-1) and fib(n-1) using child processes and add them both to get nth fibonacci number in the parent process.

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 (int argc,char *argv[])
{
if(argc!=2) {printf("invalid input\n");exit(0);}
int n=atoi(argv[1]);

for(int i=1;i<=n;++i)
printf("%d\n", parl_fib(i));
return 0;
}
```

Output:



```
harish@harish-ubuntu:~/Desktop/os/asst3$ ./p8 8
0
1
1
2
3
5
8
13
```