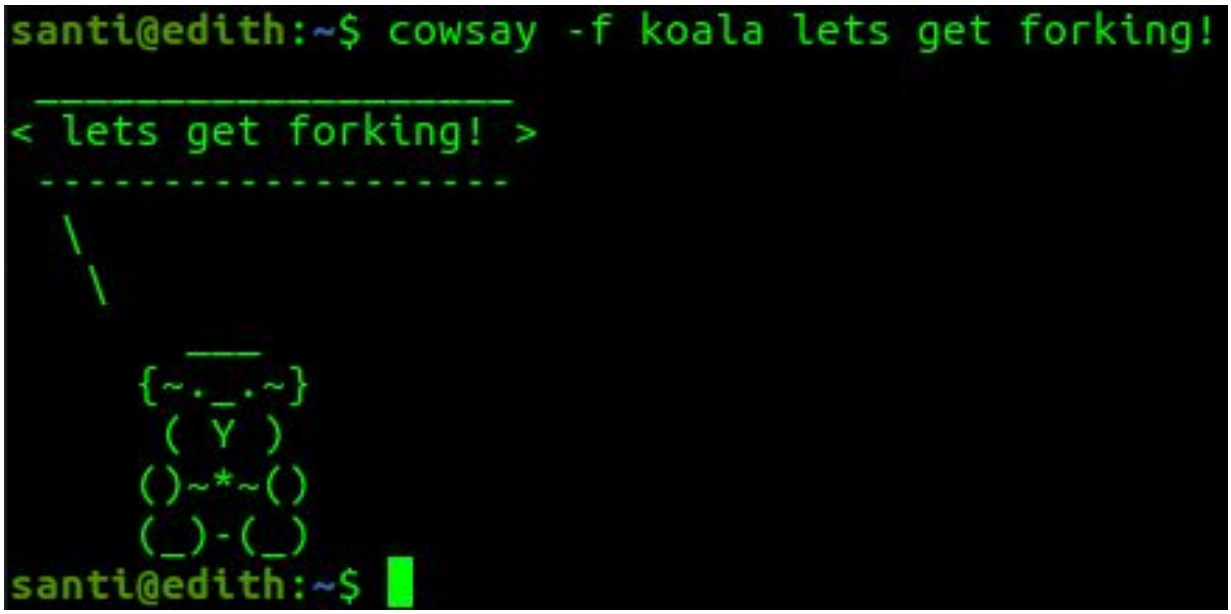

Fork Assignment - 2

COM301-P Assignment - 3

G S Santhosh Raghul
COE18B045

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

Program 1

```c
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);
   }
   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;
}
```

```
santi@edith:~/OS/L3/question_1$ ./prg1
prg1  prg1.c  prg2  prg2.c  prg3  prg3.c  prg4  prg4.c
Parent Process gets the control
Parent Has waited for Child to Complete
santi@edith:~/OS/L3/question_1$
```

Program 2

```c
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;
}
```

```
santi@edith:~/OS/L3/question_1$ ./prg2
./   bin@   cdrom/  etc/   lib@    lib64@   lost+found/  mnt/  proc/  run/   snap/  sys/  usr/
../  boot/  dev/    home/  lib32@  libx32@  media/       opt/  root/  sbin@  srv/   tmp/  var/
Parent Process gets the control
Parent Has waited for Child to Complete
santi@edith:~/OS/L3/question_1$
```

Program 3

```c
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};
       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;
}
```

```
santi@edith:~/OS/L3/question_1$ ./prg3
prg1  prg1.c  prg2  prg2.c  prg3  prg3.c  prg4  prg4.c
Parent Process gets the control
Parent Has waited for Child to Complete
santi@edith:~/OS/L3/question_1$
```

Program 4

```c
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;
}
```

```
santi@edith:~/OS/L3/question_1$ ./prg4
total 96
-rwxrwxr-x 1 santi santi 16816 Sep 20 18:18 prg1
-rw-rw-r-- 1 santi santi   472 Sep 20 18:18 prg1.c
-rwxrwxr-x 1 santi santi 16864 Sep 20 18:18 prg2
-rw-rw-r-- 1 santi santi   563 Sep 20 18:18 prg2.c
-rwxrwxr-x 1 santi santi 16864 Sep 20 18:17 prg3
-rw-rw-r-- 1 santi santi   518 Sep 20 18:14 prg3.c
-rwxrwxr-x 1 santi santi 16864 Sep 20 18:17 prg4
-rw-rw-r-- 1 santi santi   523 Sep 20 18:14 prg4.c
Parent Process gets the control
Parent Has waited for Child to Complete
santi@edith:~/OS/L3/question_1$
```

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)

```c
int main()
{
   pid_t pid;
   int n;
   printf("enter n value : ");
   scanf("%d",&n);
   pid=fork();
   if(pid==-1)
   {
       printf("fork failed\n");
       exit(1);
   }
   else if(pid>0) // parent block
   {
       printf("This is the parent block, odd series : ");
       for(int i=1;i<=n;i+=2) // print odd series
           printf("%d ",i);
       printf("\n");
   }
   else // child block
   {
       printf("This is the child block, even series : ");
       for(int i=0;i<=n;i+=2) // print even series
           printf("%d ",i);
       printf("\n");
   }
   return 0;
}
```

```
santi@edith:~/OS/L3$ ./prg2a
enter n value : 10
This is the parent block, odd series : 1 3 5 7 9
This is the child block, even series : 0 2 4 6 8 10
santi@edith:~/OS/L3$
```

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.

```c
int main()
{
   pid_t pid;
   int n,sum=0;
   printf("Enter no of elements : ");
   scanf("%d",&n);
   printf("Enter elements one by one : ");
   int arr[n];
   for(int i=0;i<n;i++)
       scanf("%d",&arr[i]);

   pid=fork();
   if(pid==-1)
   {
       printf("fork failed\n");
       exit(1);
   }
   else if(pid>0) // parent block
   {
   printf("This is the parent block, sum of odd terms from 0 to %d = ",n);
       for(int i=1;i<n;i+=2) // calculate odd term sum
           sum+=arr[i];
       printf("%d\n",sum);
   }
   else // child block
   {
   printf("This is the child block, sum of even terms from 0 to %d = ",n);
       for(int i=0;i<n;i+=2) // calculate even term sum
           sum+=arr[i];
       printf("%d\n",sum);
   }
   return 0;
}
```

```
santi@edith:~/OS/L3$ ./prg2b
Enter no of elements : 10
Enter elements one by one : 5 2 3 8 9 6 1 7 0 4
This is the parent block, sum of odd terms from 0 to 10 = 27
This is the child block, sum of even terms from 0 to 10 = 18
santi@edith:~/OS/L3$
```

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.

```c
int main()
{
   pid_t pid;
   int n,result ,num;

   printf("An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits.\nEnter a number n to display all the armstrong numbers until n : ");

   scanf("%d",&n);

   printf("Sum is calculated in the child block and checking if sum == number is done in the parent block.\n");

   printf("Armstrong numbers from 1 to %d are :\n",n);
   for(int num=1;num<=n;num++)
   {
       result =0;
       pid=vfork();
       if(pid == -1)
       {
           printf("fork failed \n");
           exit(1);
       }
       else if(pid>0) // parent block
       {
           if(result==num) // print the number if sum == number
               printf("%d\n",num);
       }
```

```
        else // child block
        {
            result = armstrong(num); // calculate sum and store in result
            exit(0);
        }
    }
    return 0;
}
```

Definitions of the functions used :

```
int armstrong(int x)
{
    int n = order(x);
    int temp = x, sum = 0, rem;
    while (temp)
    {
        int rem = temp % 10;
        sum += power(rem, n);
        temp = temp / 10;
    }
    return sum;
}
int power(int x, unsigned int y)
{
    if (y == 0)
        return 1;
    else if (y % 2)
        return  x * power(x, y / 2) * power(x, y / 2);
    return power(x, y / 2) * power(x, y / 2);
}
int order(int a)
{
    int n = 0;
    while (a)
    {
        n++;
        a=a/10;
    }
    return n;
}
```

```
santi@edith:~/OS/L3$ ./prg3
An Armstrong number is a number that is the sum of its own digits each raised to the power of the number of digits.
Enter a number n to display all the arstrong numbers until n : 10000
Sum is calculated in the child block and checking if sum == number is done in the parent block.
Armstrong numbers from 1 to 10000 are :
1
2
3
4
5
6
7
8
9
153
370
371
407
1634
8208
9474
santi@edith:~/OS/L3$
```

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

```c
int main()
{
   pid_t pid;
   int n;
   printf("enter n value : ");
   scanf("%d",&n);
   pid=fork();
   if(pid==-1)
   {
        printf("fork failed\n");
        exit(1);
   }
   else if(pid>0) // parent block
   {
        int x=0, y=1, temp;

        printf("This is parent block. Fibonacci sereis generation is done here\n");
```

```
      for (int i=1;i<=n;i++)
       {
           printf("%d ",x);
           temp=x+y;
           x=y;
           y=temp;
       }
       printf("\n");
   }
   else // child block
   {
       int x=3,count=1,flag,i;
       printf("This is child block. Prime sereis generation is done
here\n");
       if(n>0)
           printf("2 ");
       while(count<n)
       {
           flag=1;
           for(i=2;i<=sqrt(x);i++)
               if(x%i==0)
               {
                   flag = 0;
                   break;
               }
           if(flag)
           {
               printf("%d ",x);
               count++;
           }
           x++;
       }
       printf("\n");
   }
   return 0;
}
```

```
santi@edith:~/OS/L3$ ./prg4
enter n value : 15
This is parent block. Fibonacci sereis generation is done here
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
This is child block. Prime sereis generation is done here
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
santi@edith:~/OS/L3$
```

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)

```
int main()
{
   pid_t pid;
   int n;
   printf("Enter no of elements : ");
   scanf("%d",&n);
   printf("Enter elements one by one : ");
   int arr[n];
   for(int i=0;i<n;i++)
       scanf("%d",&arr[i]);
   pid=fork();
   if(pid==-1)
   {
       printf("fork failed\n");
       exit(1);
   }
   else if(pid>0) // parent block
   {
       printf("This is parent block. Sorting in ascending order is done
here\n");
       asc_sort(arr,0,n-1); // sort in ascending order
       printf("Array after sorting : ");
       for(int i=0;i<n;i++)
           printf("%d ",arr[i]);
```

```
        printf("\n");
    }
    else // child block
    {
        printf("This is child block. Sorting second half in descending
order is done here\n");
        desc_sort(arr,0,n-1);  // sort in descending order
        printf("Array after sorting : ");
        for(int i=0;i<n;i++)  printf("%d ",arr[i]);
        printf("\n");
    }
    return 0;
}
```

Definitions of the functions used :

```
void asc_sort(int a[],int start,int end)
{
    int temp;
    for(int i=start;i<=end;i++)
        for(int j=start;j<(end-i+start);j++)
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
}
void desc_sort(int a[],int start,int end)
{
    int temp;
    for(int i=start;i<=end;i++)
        for(int j=start;j<(end-i+start);j++)
            if(a[j]<a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
}
```

```
santi@edith:~/OS/L3$ ./prg5
Enter no of elements : 10
Enter elements one by one : 5 2 3 8 9 6 1 7 0 4
This is parent block. Sorting in ascending order is done here
Array after sorting : 0 1 2 3 4 5 6 7 8 9
This is child block. Sorting second half in descending order is done here
Array after sorting : 9 8 7 6 5 4 3 2 1 0
santi@edith:~/OS/L3$
```

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)

```
int main()
{
   pid_t pid;
   int n;
   printf("Enter no of elements : ");
   scanf("%d",&n);
   printf("Enter elements one by one : ");
   int arr[n];
   for(int i=0;i<n;i++)
       scanf("%d",&arr[i]);

   pid=vfork();
   if(pid==-1)
   {
       printf("fork failed\n");
       exit(1);
   }
   else if(pid>0) // parent block
   {
       printf("This is parent block. Sorting first half in ascending order
is done here\n");
       asc_sort(arr,0,n/2-1);
   }
```

```
    else // child block
    {
        printf("This is child block. Sorting second half in descending
order is done here\n");
        desc_sort(arr,n/2,n-1);
        exit(0);
    }
    printf("Array after sorting : ");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");
    return 0;
}
```

Definitions of the functions - same as previous program.

```
santi@edith:~/OS/L3$ ./prg6
Enter no of elements : 10
Enter elements one by one : 5 2 3 8 9 6 1 7 0 4
This is child block. Sorting second half in descending order is done here
This is parent block. Sorting first half in ascending order is done here
Array after sorting : 2 3 5 8 9 7 6 4 1 0
santi@edith:~/OS/L3$
```

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)

```
int main()
{
   pid_t pid;
   int n,x,count;
   printf("Implementation of binary search.\nEnter no of elements : ");
   scanf("%d",&n);
   printf("Enter elements one by one : ");
   int arr[n];
   for(int i=0;i<n;i++)
       scanf("%d",&arr[i]);
   printf("Enter element to be searched : ");
   scanf("%d",&x);

   pid=vfork();
   if(pid==-1)
   {
       printf("fork failed\n");
       exit(1);
   }
   else if(pid>0) // parent block
   {
       printf("\nThis is parent block. Searching in first half is done
here.\n");
       sort(arr,0,n/2-1);
       printf("Sorting in 1st half completed! Now starting search.\n");
       int pos[2];
       bin_search(arr,x,0,n/2-1,pos);
       if(*pos==-1)
           printf("%d was not found in the 1st half.\n",x);
       else if(*pos==*(pos+1))
           printf("%d occurs once in the 1st half, at index
%d.\n",x,*pos);
       else
           printf("%d occurs %d times in the 1st half, from index %d to
%d.\n",x,*(pos+1)-*pos+1,*pos,*(pos+1));
   }
```

```
    else // child block
    {
        printf("\nThis is child block. Searching in second half is done
here.\n");
        sort(arr,n/2,n-1);
        printf("Sorting in 2nd half completed! Now starting search.\n");
        int pos[2];
        bin_search(arr,x,n/2,n-1,pos);
        if(*pos==-1)
            printf("%d was not found in the 2nd half.\n",x);
        else if(*pos==*(pos+1))
            printf("%d occurs once in the 2nd half, at index %d (realtive
index %d).\n",x,*pos,*pos-n/2);
        else
            printf("%d occurs %d times in the 2nd half, from index %d to
%d.\n",x,*(pos+1)-*pos+1,*pos,*(pos+1));
        exit(0);
    }
    printf("\nArray after sorting 2 halves separately: ");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");
    return 0;
}
```

Definitions of the functions - `sort()` same as `asc_sort()` in previous program.

```
void bin_search(int a[], int x, int start, int end,int* pos)
{
    // pos={-1,-1};
    *pos=bin_search_first_occ(a,x,start,end);
    // returns the starting position of the search key
    if(pos[0]!=-1)
    {
        pos[1]=pos[0];
        while(a[pos[1]]==x)
            pos[1]++;       // find ending postion of search key in pos[1]
        pos[1]--;
    }
}
```

```c
int bin_search_first_occ(int a[], int x, int left, int right)
{
   if(left==right)
       return left;
   int mid = (right+left)/2;
   if(a[mid]<x) // right half
       return bin_search_first_occ(a, x, mid+1, right);
   else // left half
       return bin_search_first_occ(a, x, left, mid);
}
```

```
santi@edith:~/OS/L3$ ./prg7
Implementation of binary search.
Enter no of elements : 10
Enter elements one by one : 5 3 1 3 6 0 1 5 3 7
Enter element to be searched : 3

This is child block. Searching in second half is done here.
Sorting in 2nd half completed! Now starting search.
3 occurs once in the 2nd half, at index 7 (realtive index 2).

This is parent block. Searching in first half is done here.
Sorting in 1st half completed! Now starting search.
3 occurs 2 times in the 1st half, from index 1 to 2.

Array after sorting 2 halves separately: 1 3 3 5 6 0 1 3 5 7
santi@edith:~/OS/L3$
```

8) * 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)

Here, the question talks about a “faster version” of fibonacci series generation using parallel processing. **I claim that there is no “faster version” than the dynamic programming approach of generating fibonacci series which incurs a time complexity of O(n).** Here is my argument :

The formal definition of the fibonacci numbers is as follows:

Fibonacci numbers form a sequence called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1.

(i.e) $F_0 = 0$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n > 1$.

Here, it is very clear that we need the value of F_{n-1} and F_{n-2} in order to find the value of F_n . So, for an arbitrary number k, to display Fibonacci numbers until k, we need to calculate F_{k-1} , F_{k-2} , F_{k-3} , ... , $F_{k-(k-1)} = F_1$, $F_{k-k} = F_0$. So, we have to make k+1 operations in total and each operation is of O(1). So, the total time complexity is O(k).

One might ask what about parallel processing. To answer this question, we have to note that each fibonacci value except F_0 and F_1 depend entirely on the previous terms. It is impossible to calculate the k^{th} term unless we have already calculated the $(k-1)^{th}$ term and the $(k-2)^{th}$ term. So, applying parallel processes to calculate different fibonacci numbers will only result in more number of processes which wait for other processes and there won’t truly be any parallel processing.

There are other approaches for finding the n^{th} fib term in O(logn) time but the question talks about fibonacci series generation and not n^{th} term generation and more over, those algorithms don’t use parallel processing to optimize the n^{th} term generation so I have not mentioned about those in this assignment.

So, my conclusion is that there is no way of parallelizing the generation of fibonacci series to make it faster since every term depends on the previous terms. Because of what I’ve mentioned above, I have not attached any program with this question.