ASSESSMENT-3
17BIT0028
SIDDHI SINGH

QUES 1 : The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm: n = n/2, if n is even n = 3 × n + 1, if n is odd The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if n = 35, the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1.Write a C program using the fork () system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the Command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the wait () call to wait for the child process to complete before exiting the program

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

int main()
{
    printf("\n\n Solution to Collatz Conjecture by Sid-
dhi, 17BIT0028\n");

    int n=0;
    int k=0;

    pid_t pid;

        do
        {
```

```
            printf("Please enter a number greater than 0
to run the Collatz Conjecture.\n");
            scanf("%d", &k);
        }while (k <= 0);

        pid = fork();

        if (pid == 0)
        {
            printf("Child is working...\n");
            printf("%d\n",k);
            while (k!=1)
            {
                if (k%2 == 0)
                {
                    k = k/2;
                }
                else if (k%2 == 1)
                {
                    k = 3 * (k) + 1;
                }

                printf("%d\n",k);
            }

            printf("Child process is done.\n");
        }
        else
        {
            printf("Parents is waiting on child
process...\n");
            wait();
            printf("Parent process is done.\n");
        }
    return 0;
}
```

```
17bit0028@sjtg18site009: ~
17bit0028@sjtg18site009:~$ gcc prog1.c
17bit0028@sjtg18site009:~$ ./a.out


 Solution to Collatz Conjecture by Siddhi, 17BIT0028
Please enter a number greater than 0 to run the Collatz Conjecture.
35
Parents is waiting on child process...
Child is working...
35
106
53
160
80
40
20
10
5
16
8
4
2
1
Child process is done.
Parent process is done.
17bit0028@sjtg18site009:~$
```

QUES 2 : Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85 The program will report The average value is 82 The minimum value is 72 The maximum value is 95 The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited.

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <errno.h>



#define handle_error_en(en, msg) \

        do { errno = en; perror(msg); exit(EXIT_FAILURE);
} while (0)


volatile int running_threads = 0;


pthread_t thread[3];


int numOfElements;


struct Results{
```

```
    int min;
    int max;
    int average;

}Results;


void *findMin(void *array_ptr){

    int i;

    int *elements = (int*)array_ptr;

    Results.min = elements[0];

    for(i = 0; i < numOfElements; i++){
if(elements[i] < Results.min)
{Results.min = elements[i];}

    }

    running_threads -= 1;

return NULL;

}
```

```
void *findMax(void *array_ptr){

    int i;
    int *elements = (int*)array_ptr; /

    for(i = 0; i < numOfElements; i++){

        if(elements[i] > Results.max){
            Results.max = elements[i];

        }
    }

    running_threads -= 1;

return NULL;

}


void *findAverage(void *array_ptr){

    int i;

    int *elements = (int*)array_ptr;
    for(i = 0; i < numOfElements; i++){
```

```
        Results.average += elements[i];
    }

    Results.average = Results.average/numOfElements;
    running_threads -= 1;
return NULL;

}



int getArrayInput(int n, int *array_ptr){

        int input;

        int numberOfElements = 0;

    printf("Creating Dynamic Array...\n-\n");

        for(;;){

        printf("Enter a positive value:\nNegative Number
to Stop\n-\n");



            if (scanf("%d",&input) != 1){

                printf("\nOops that wasn't an
Integer\nlets try filling the array again\nRemember
```

```
INTEGERS only!\n");

                exit(EXIT_FAILURE);

            }

        if (input >= 0){

                if (numberOfElements == n){

                  n += 1;

                    array_ptr  =  realloc(array_ptr,  n  *
sizeof(int));

                }

            array_ptr[numberOfElements++] = input;

        } else {

               printf("\nNumber   of   Integers:   %d\n",
numberOfElements);

             break;

                }
```

```
            }

    return numberOfElements;

        }


void joinThreads(int numberOfThreads){

    int i;

    int s;

    while(numberOfThreads >= 0){

        s = pthread_join(thread[numberOfThreads], NULL);


         if (s != 0){

            handle_error_en(s, "pthread_create");

         }

         numberOfThreads--;

    }
```

```
}


void createThreads(int *array_ptr){

    int s;


    s = pthread_create(&thread[0], NULL, findMin, (void
*)array_ptr);

     if (s != 0){

            handle_error_en(s, "pthread_create");

         }
            running_threads += 1;


     s = pthread_create(&thread[1], NULL, findMax, (void
*)array_ptr);

         if (s != 0){

           handle_error_en(s, "pthread_create");

         }
         running_threads += 1;
```

```
      s  =  pthread_create(&thread[2],  NULL,  findAverage,
(void *)array_ptr);

         if (s != 0){

           handle_error_en(s, "pthread_create");

         }

            running_threads += 1;

}


int main(){

    int n = 1;

    int *array_ptr = malloc(n * sizeof(int));


         numOfElements = getArrayInput(n, array_ptr);

         createThreads(array_ptr);

        while(running_threads>0){
```

```
                sleep(1);

            }

        joinThreads(2);


        printf("\nThe average is %d\nThe maximum is %d\nThe minimum is %d\n",Results.average, Results.max, Results.min);

    return(0);

}
```