1. I chose 2 as my k value, hence created $2^2 = 4$ new processes inlcuding the main process forming a complete binary tree.

   Below is the program together with it's output:

   q1.c

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

int main(int argc, char* argv[]){
    int pid;
    printf(" \npid of the main process: %i\n", getpid());
    for (int i = 0; i < 2; i++)
    {
        pid = fork();
        if(pid != 0){
            printf("\nparent process's pid set to children's pid: %i\n", pid);
        }
    }
    return 0;
}
```

   make
   ./q1

```
pid of the main process: 8961

parent process's pid set to children's pid: 8962

parent process's pid set to children's pid: 8963

parent process's pid set to children's pid: 8964
```

2. Sample 10 fields in the `task_struct` definition which can be found under the Linux kernel source code inside the `sched.h` file.

   - struct thread_info thread_info
   - volatile long state
   - void *stack
   - refcount_t usage
   - unsigned int flags

- unsigned int ptrace
- int on_cpu
- unsigned int cpu
- unsigned int wakee_flips
- unsigned long wakee_flip_decay_ts

3. Total of 10 processes created throughout the program execution via the `fork` systemcall. The children that are created in the iteration with even number are terminated by program.

4. Following is the output of the program to the console. The program prints the number 100 and then three consecutive 250 on the console. Due to asynchronous nature of the program, the execution order is susceptible to change. Additionally you can see the execution of the `ls` command in the output in between printing numbers.

```
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw2/q4$ ./q4

100
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw2/q4$ total 4,0K
drwxr-xr-x 19 zcankara zcankara 4,0K Şub 15 14:42 zcankara

250

250

250
```

5. The q5.c program which being used to create 2 child processes and execute "ps aux" and "ls -al" programs while the parent of the children waiting children to process.

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

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

    pid_t child1, child2, wpid;
    int status = 0;;
    char *binaryPath = "/bin/ls";
    char *arg1 = "-al";
    child1 = fork();
    if (child1 == 0) {
        execlp("ps", "ps", "aux", NULL);
        exit(0);
    } else {
        child2 = fork();
        if (child2 == 0) {
            execlp(binaryPath, binaryPath, arg1, NULL);
            exit(0);
        } else {
            /* parent */
        }
    }
    while ((wpid = wait(&status)) > 0);
    return 0;
}
```

6. Creates two children. One child sends the message “I hear and I forget. I see and I remember. I do and I understand.” string to the other child over a messaging queue. The other child receives the string and prints to the console.

   Below is the C program together with it’s output:

   q6.c

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

// structure for message queue
struct mesg_buffer {
        long mesg_type;
        char mesg_text[100];
} message;

int main(int argc, char* argv[]){
    key_t key;
        int msgid;

        // ftok generates an unique key
        key = ftok("progfile", 65);

        // creates a message queue and return the identifier
        msgid = msgget(key, 0666 | IPC_CREAT);

    pid_t child1, child2;
    child1 = fork();
    if (child1 == 0) {
        message.mesg_type = 1;
```

```c
        char * line = "I hear and I forget. I see and I remember. I do and I understand.";
        // write data to the messaging queue
        for (int i = 0; i < 66; i++)
        {
            message.mesg_text[i] = line[i];
        }
        // msgsnd to send message
        msgsnd(msgid, &message, sizeof(message), 0);

        // display the message
        printf("Message Sent: %s \n", message.mesg_text);
        exit(0);
    } else {
        child2 = fork();
        if (child2 == 0) {
            // msgrcv to receive message
            msgrcv(msgid, &message, sizeof(message), 1, 0);

            // display the message
            printf("Message Received : %s \n", message.mesg_text);
            exit(0);
        } else {
            wait(NULL);
        }
    }
    // wait the child processes to complete
    wait(NULL);
    // destroy the message queue
        msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

make
./q6

```
zcankara@zcankara-VirtualBox:~/Desktop/cs342/hw2/q6$ ./q6
Message Sent: I hear and I forget. I see and I remember. I do and I understand.
Message Received : I hear and I forget. I see and I remember. I do and I understand.
```

7. q7.c

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
int main(int argc, char *argv[])
 {
    int f1, f2;
    char buff[100];
    long int n;
    // open the file to read from which passed as the first argument to the program
    f1 = open(argv[1], O_RDONLY);
    // open the file to write to which passed as the second argument to the program
    f2 = open(argv[2], O_CREAT | O_WRONLY | O_TRUNC, 0700);
    // read the input file
    n = read(f1, buff, 100);
    if(n>0){
        // process byte by byte
        for (int i = 0; i < n; i++)
        {
            char c = buff[i];
            // write the same character twice to the output file
            write(f2, &c, sizeof(c));
            write(f2, &c, sizeof(c));
        }
        close(f2);
        exit(0);
    }
 }
```