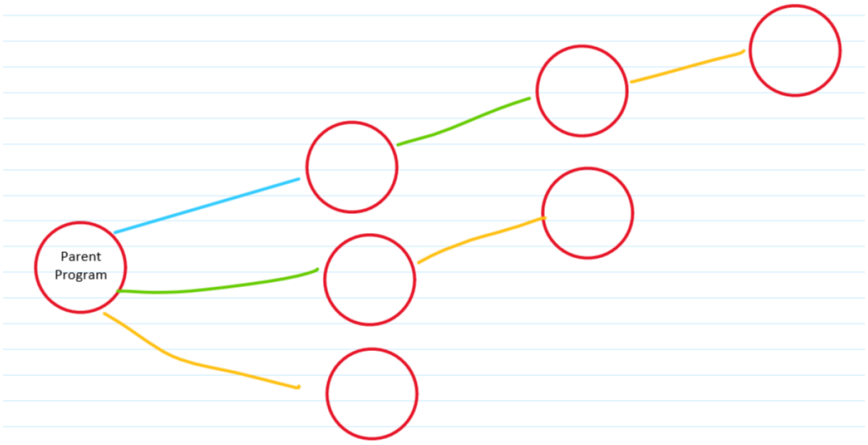
Operating System, hw 1

B07702011, NTU ACCT, Jack Chen

1.

There are 8 processes created by this program. Because once create a process, the parent process and the child process will both continue execute the instruction after fork(). Thus the parent process will create 3 child processes, and the first child process will create 2 child processes, the second child process will create 1 child process. Shown by the following figure, blue for first fork(), green for second, yellow for the last one.



2.

First, use scanf(), so we can get integer from command line. And perform some check to make sure we don't get a negative number.

```
int input;
scanf("%d",&input);

while(input <= 0){
    printf("Please enter a positive integer\n");
    scanf("%d",&input);
}
```

```
jacklinux@jacklinux-VirtualBox:~/hw$ ./1-2
-10
Please enter a positive integer
-15
Please enter a positive integer
35
35,106,53,160,80,40,20,10,5,16,8,4,2,1
Child process is done
jacklinux@jacklinux-VirtualBox:~/hw$
```

Code and results of checking positivity

Then just follow the structure from slides. Use fork() to create a child process, and use a if-else, to tell parent and child what should they do. The pid of child process is 0, and child process is responsible for calculation. Use a while loop, while the input number does not equals to 1, perform the algorithm and print out the result. The pid of parent process is not 0, and for parent process, all it need to do is wait() until the child process to finish.

```c
pid_t pid;
pid = fork();

if (pid < 0){
    printf("Fork failed");
    return 1;
}
else if (pid == 0){
    //child process
    printf("%d",input);
    while (input != 1){
        printf(",");
        if(input % 2 == 0){
            input = input / 2;
            printf("%d",input);
        }
        else{
            input = input *3 + 1;
            printf("%d",input);
        }
    }
}
else{
    //parent process
    wait(NULL);
    printf("\nChild process is done\n");
}
```

Code for primary process

Results:

```
jacklinux@jacklinux-VirtualBox:~/hw$ gcc 1-2.c -o 1-2
jacklinux@jacklinux-VirtualBox:~/hw$ ./1-2
35
35,106,53,160,80,40,20,10,5,16,8,4,2,1
Child process is done
jacklinux@jacklinux-VirtualBox:~/hw$
```

3.

Just like the previous question, but for this one we need to use share memory to implement it. For the both process, we open share memory, one for write and another for read, then we use fork() to create a process. For child process, it still does the calculation part, and every time it has done the algorithm, we write the result into share memory. For parent process, waits for child process to complete first, then through the share memory it reads the results, then prints out.

```c
pid_t pid;
pid = fork();

const int SIZE = 4096;
const char *smptr = "shared";
int shm_fd;
void *ptr;
char *com = ",";


if(pid < 0){
    printf("Fork failed");
    return 1;
}
else if (pid == 0){
    //child process
    shm_fd = shm_open(smptr,O_CREAT|O_RDWR,0666);
    ftruncate(shm_fd,SIZE);
    ptr = mmap(0,SIZE,PROT_WRITE,MAP_SHARED,shm_fd,0);
    //calculating
    while(input != 1){
        ptr += sprintf(ptr,"%d",input);
        sprintf(ptr,"%s",com);
        ptr += strlen(com);
        if(input % 2 == 0){
            input = input / 2;
        }
        else{
            input = input * 3 + 1;
        }
    }
    ptr += sprintf(ptr,"%d",input);

}
else{
    //parent process
    wait(NULL);
    //start printing
    shm_fd = shm_open(smptr,O_RDONLY,0666);
    ptr = mmap(0,SIZE,PROT_READ,MAP_SHARED,shm_fd,0);
    printf("%s\nProcess complete\n",(char *)ptr);
    shm_unlink(smptr);
}
```

Code for primary process

Results:

```
jacklinux@jacklinux-VirtualBox:~/hw$ gcc 1-3.c -lrt -o 1-3
jacklinux@jacklinux-VirtualBox:~/hw$ ./1-3
35
35,106,53,160,80,40,20,10,5,16,8,4,2,1
Process complete
jacklinux@jacklinux-VirtualBox:~/hw$
```