REPORT

- **Exercise1**

  First step of my solution is to set the handlers for the system calls *‘SIGUSR1’* and *‘SIGUSR2’* in order to make the process execute the catcher and not the default behavior.

```
void catcherSig1(){
    if(sig1 == 0){
     printf("[Father]: I cannot print because of my child!!!\n");
     sig1 = 1;
    }else{
     printf("[Father]: At last, I can print again!!!\n");
     sig1 = 0;
    }
}
```

The first catcher defines the behavior of the main process (*the father*) when receives the *‘SIGUSR1’* sys call: the main idea is to use a global variable called ***sig1*** which is set to 0 at the beginning, in order to use it inside the main process’s body to “pause”/”re-start” the printing.
The other two catchers need to send the *SIGUSR2* signal to the main process:

1. the `catcherSig2Child()`, as above, is used to set a global variable that is previously set to 0;
2. the `catcherSig2GrandChild()`, instead, sends the signal *SIGUSR2* to its own father and then terminates.

```
void catcherSig2Child(){
    printf("[Father]: Received SIGUSR2 from the child\n");
    sig2 = 1;
}

void catcherSig2GrandChild(){
    printf("[Child]: Received SIGUSR2 from the grandchild\n");
    kill(getppid(), SIGUSR2);
    exit(0);
}
```

The father performs a *fork()*, then opens a file and enters into an infinite loop in which:

1. reads each line;
2. prints the line number and the line content;
3. rewind the file.

As soon as it receives a *SIGUSR1* signal it stops printing and it will wait until it will receive another *SIGUSR1* signal. Finally, when it receives the *SIGUSR2* signal it will terminate.

```c
c = fork();
...
else if(c > 0){ //father
     fp = fopen(argv[1], "r");
     while(1){
          if(sig2 == 1){
               fclose(fp);
               exit(0);
          }else{
               while(fgets(line, sizeof(line), fp) != NULL){
                    if(sig1 == 0){
                         line[strlen(line)-1] = '\0';
                         i++;
                         printf("Row %d: %s\n", i, line);
                    }else{
                         pause();
                    }
               }
               rewind(fp);
               i = 0;
          }
     }
}
```

The child loops to the infinite and sends a *SIGUSR1* signal to the parent at random intervals between [1-10] seconds, then after 60 seconds must send a *SIGUSR2* signal to the parent and terminate. As suggested in the text, the child does another *fork()* making a new child (*the grandchild*), which sleeps 60 seconds after that sends a *SIGUSR2* signal to its father that will catch it and forward to its parent.

```c
if(c == 0){ //child
    signal(SIGUSR2, catcherSig3);
    if(fork() != 0){ //still child
          while(1){
               x = (rand() % 10) + 1;
               sleep(x);
               kill(getppid() ,SIGUSR1);
          }
    }else{ //grandchild
          sleep(60);
          kill(getppid() ,SIGUSR2);
          exit(0);
    }
}
```