Assignment 8

- Code

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

static volatile sig_atomic_t sigflag; /* set nonzero by sig handler */
static sigset_t newmask, oldmask, zeromask;
FILE* fp;
pid_t pid;

static void sig_usr(int signo) { /* one signal handler for SIGUSR1 and S
    sigflag = 1;
}

static void TELL_WAIT(void) {
    if (signal(SIGUSR1, sig_usr) == SIG_ERR)
        perror("signal(SIGUSR1) error");

    if (signal(SIGUSR2, sig_usr) == SIG_ERR)
        perror("signal(SIGUSR2) error");

    sigemptyset(&zeromask);
    sigemptyset(&newmask);
    sigaddset(&newmask, SIGUSR1);
    sigaddset(&newmask, SIGUSR2);

    /* Block SIGUSR1 and SIGUSR2, and save current signal mask */
    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0)
        perror("SIG_BLOCK error");
}

static void TELL_PARENT(void) {
    kill(getppid(), SIGUSR2); /* tell parent we're done */
}

static void WAIT_PARENT(void) {
    while (sigflag == 0)
        sigsuspend(&zeromask); /* and wait for parent */
    sigflag = 0;
    /* Reset signal mask to original value */
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
        perror("SIG_SETMASK error");
}

static void TELL_CHILD(pid_t pid) {
    kill(pid, SIGUSR1); /* tell child we're done */
}

static void WAIT_CHILD(void) {
    while (sigflag == 0)
        sigsuspend(&zeromask); /* and wait for child */
    sigflag = 0;
    /* Reset signal mask to original value */
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
        perror("SIG_SETMASK error");
}

static int increment_counter(FILE *const file) {
    /* TODO */
    int val;
    fscanf(file, "%d", &val);
    ++val;
    fseek(file, 0L, SEEK_SET);
    fprintf(file, "%d", val);
    fclose(fp);
    return val;
}

int main(void) {
```

```
71      /* TODO */
72      fp = fopen("result.txt", "w");
73      fprintf(fp, "%d", 0);
74      fclose(fp);
75
76      pid = fork();
77      TELL_WAIT();
78
79      while(!pid) {
80          fp = fopen("result.txt", "r+");
81          int cur = increment_counter(fp);
82          if (cur > 100) {
83              TELL_PARENT();
84              break;
85          }
86          printf("Child incrementing, value: %d\n", cur);
87          TELL_PARENT();
88          WAIT_PARENT();
89      }
90
91      while(pid) {
92          WAIT_CHILD();
93          fp = fopen("result.txt", "r+");
94          int cur = increment_counter(fp);
95          if (cur > 100) {
96              TELL_CHILD(pid);
97              break;
98          }
99          printf("Parent incrementing, value: %d\n", cur);
100         TELL_CHILD(pid);
101     }
102
103     return 0;
104 }
```

Implementation

INCREMENT_COUNTER

- line 62: read the value in file into val.
- line 63: increment.
- line 65: write the updated value back to file(replace).
- line 66 & 67: close the file, return the counter value.

MAIN

- line 72-74: create “result.txt”, write a 0 into it(initialization).
- line 76: fork a child process.
- line 77: initialize the synchronization process(TELL_WAIT).
- line 79: while(pid == 0) (child)
- line 91: while(pid > 0) (parent)
- line 80-81, 93-94: increment the counter.
- line 82-85, 95-98: if counter > 100, tell child/parent, then break.

- line 86, 99: print message.
- line 92: since child have to do first, we wait child here.