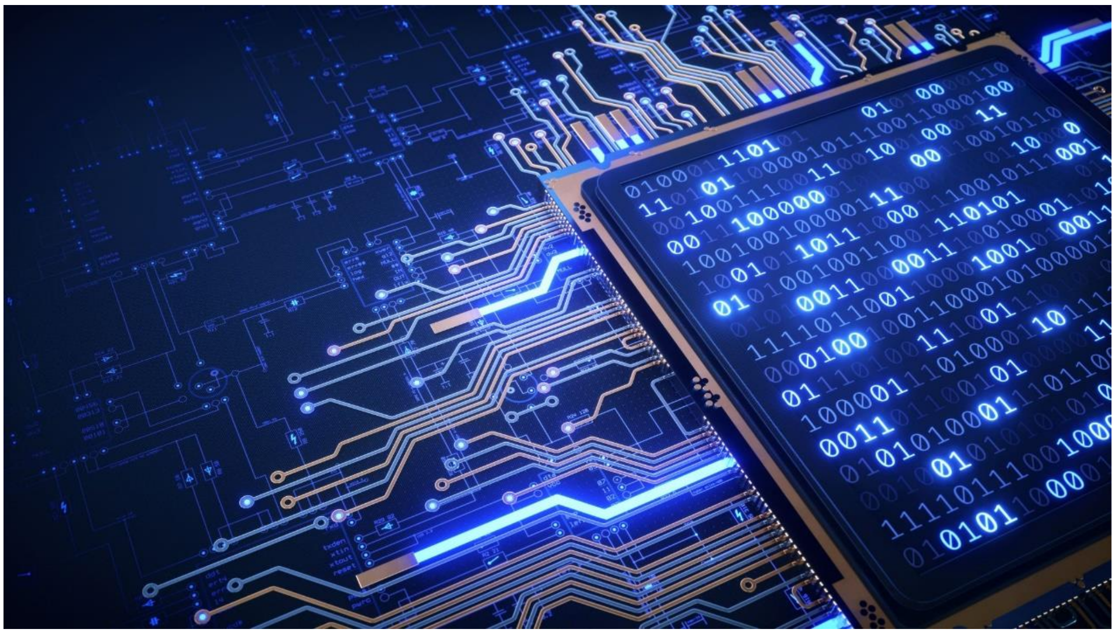
ADVANCED UNIX PROGRAMMING ASSIGNMENT REPORT

ASSIGNMENT 8

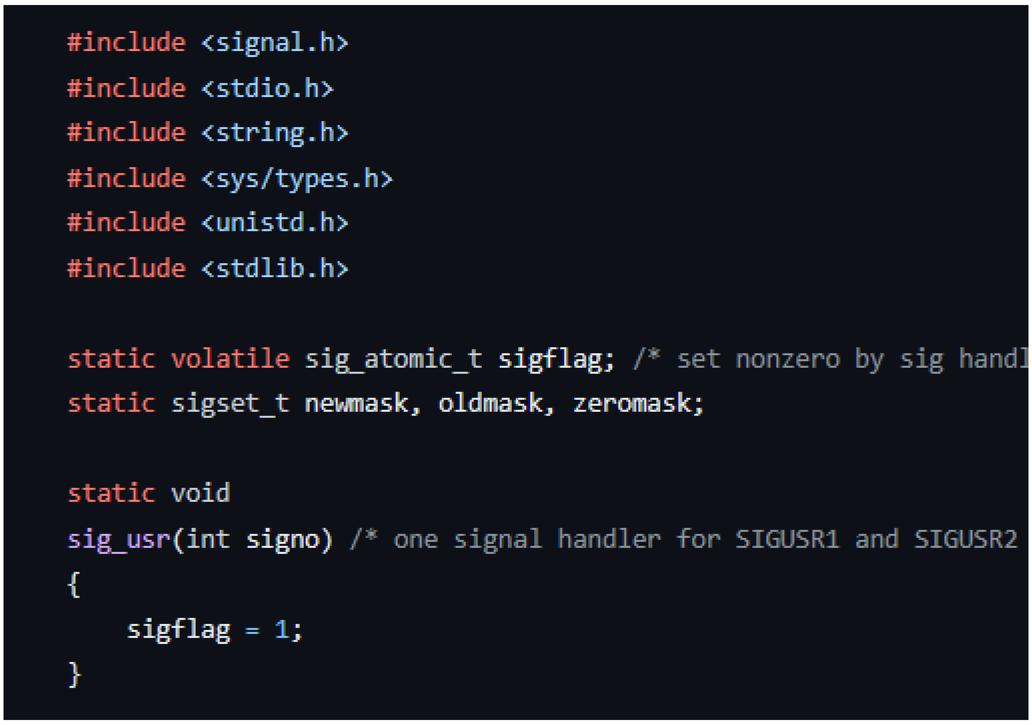


TEAM 9 — 林禾堃、馬毓昇、陳曦

Nov 2023

1. Code Implementation

Declare a flag for alternating the parent and the child, the signals masks, and set up the signal handler for SIGUSR's:

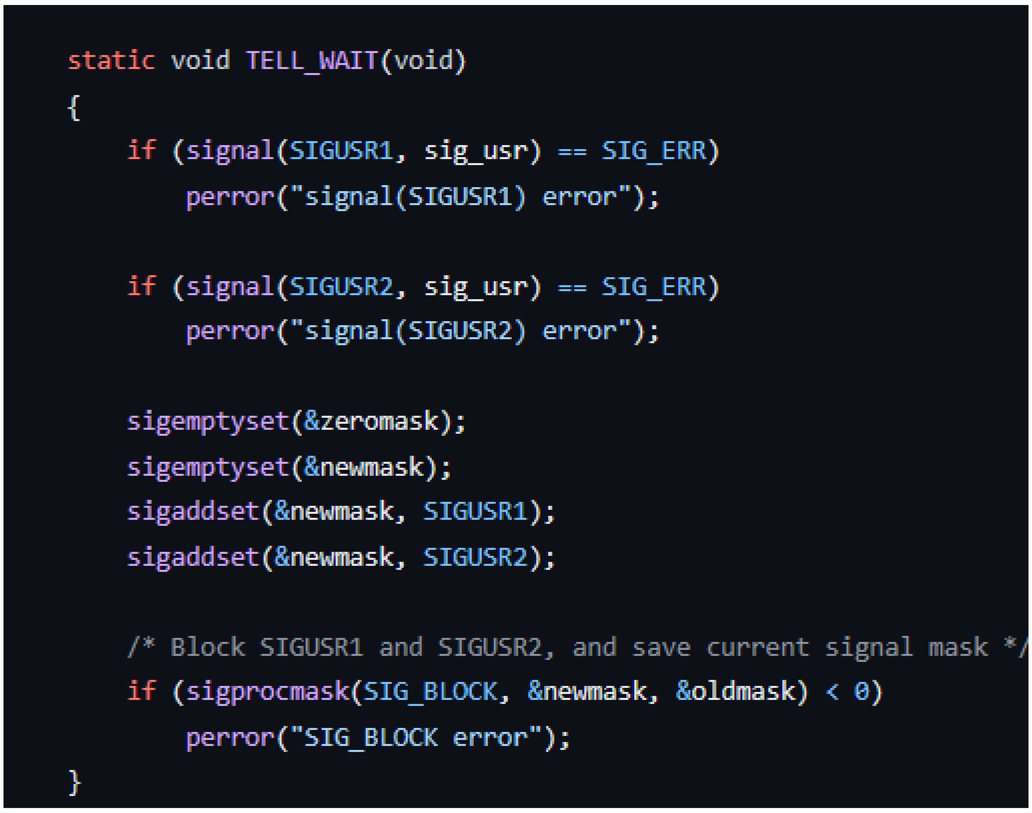


```c
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

static volatile sig_atomic_t sigflag; /* set nonzero by sig handl
static sigset_t newmask, oldmask, zeromask;

static void
sig_usr(int signo) /* one signal handler for SIGUSR1 and SIGUSR2
{
    sigflag = 1;
}
```

TELL_WAIT: set up the signals and the masks, then block the SIGUSR's.



```c
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");
}
```

TELL_PARENT and TELL_CHILD handle the termination notifications and WAIT_PARENT and WAIT_CHILD handle the alternating blocking of the processes.

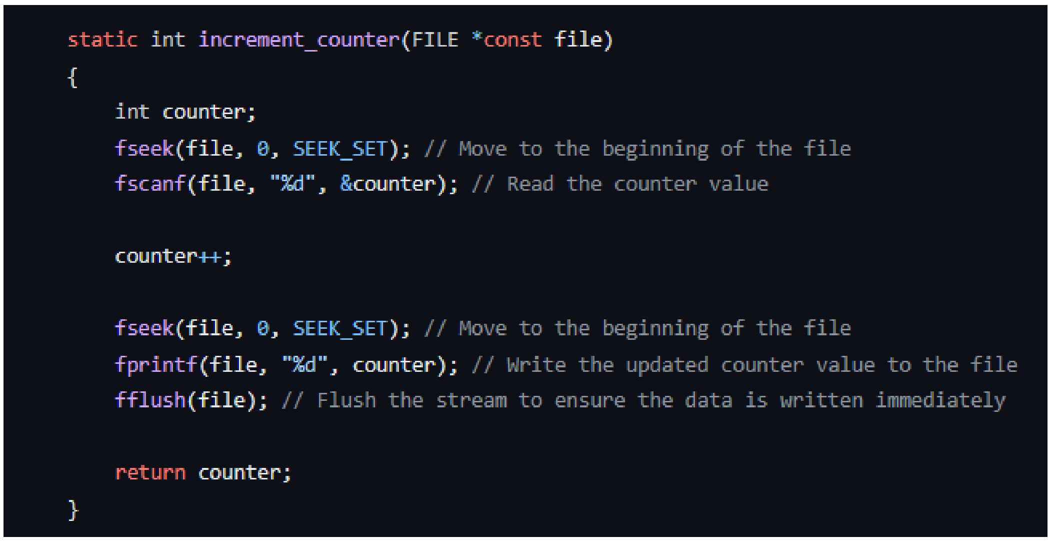
```c
static void TELL_PARENT(void)
{
    kill(getppid(), SIGUSR2); /* tell parent we’re done */
}

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");
}
```

Implement a counter incrementing function that reads the previously stored counter value, add it by one and store it back.



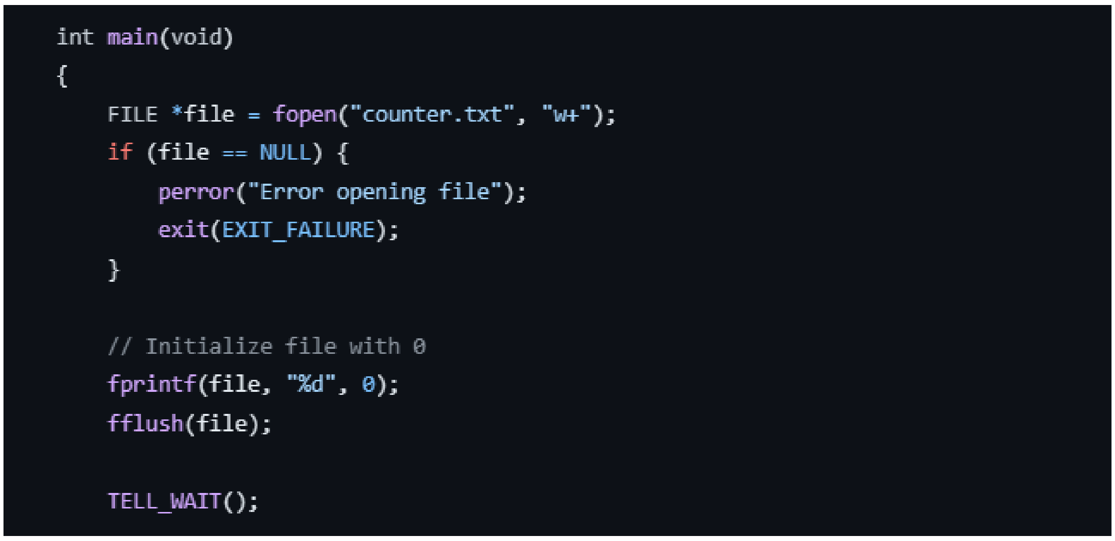
```c
static int increment_counter(FILE *const file)
{
    int counter;
    fseek(file, 0, SEEK_SET); // Move to the beginning of the file
    fscanf(file, "%d", &counter); // Read the counter value

    counter++;

    fseek(file, 0, SEEK_SET); // Move to the beginning of the file
    fprintf(file, "%d", counter); // Write the updated counter value to the file
    fflush(file); // Flush the stream to ensure the data is written immediately

    return counter;
}
```

Finally, in main function, initialize everything with a call of TELL_WAIT():

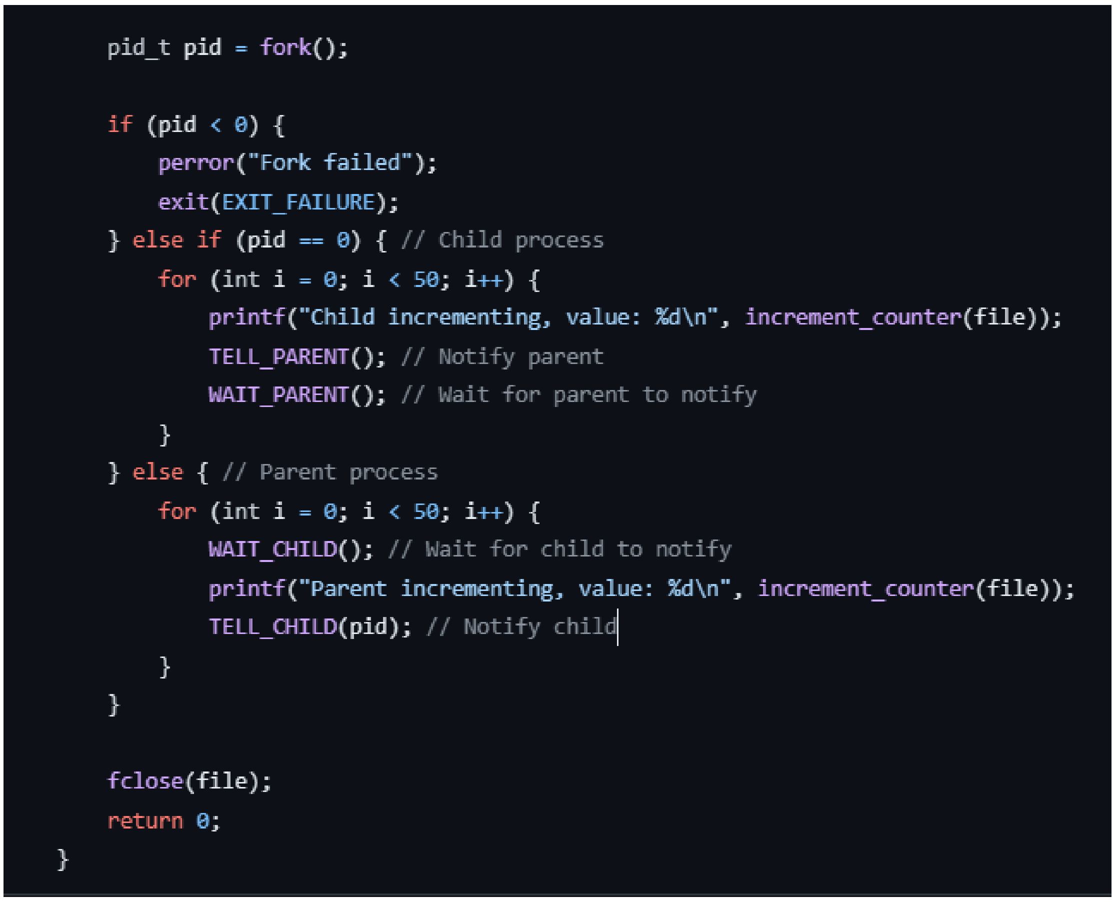


```c
int main(void)
{
    FILE *file = fopen("counter.txt", "w+");
    if (file == NULL) {
        perror("Error opening file");
        exit(EXIT_FAILURE);
    }

    // Initialize file with 0
    fprintf(file, "%d", 0);
    fflush(file);

    TELL_WAIT();
```

Then, fork the processes and utilize the aforementioned functions to let the two processes take turn to print the required output. The child will first increment and print the counter value, then notify the waiting parent to do so while entering waiting state. On the other hand, the parent will do the similar things after being notified, which will then build two alternating loop, letting the child and the parent increase and print the counter value alternatively.



```
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) { // Child process
        for (int i = 0; i < 50; i++) {
            printf("Child incrementing, value: %d\n", increment_counter(file));
            TELL_PARENT(); // Notify parent
            WAIT_PARENT(); // Wait for parent to notify
        }
    } else { // Parent process
        for (int i = 0; i < 50; i++) {
            WAIT_CHILD(); // Wait for child to notify
            printf("Parent incrementing, value: %d\n", increment_counter(file));
            TELL_CHILD(pid); // Notify child
        }
    }

    fclose(file);
    return 0;
}
```

Lastly, close the file and exit.