CSE344 – System Programming - Signal Handling, Homework 2 Report

Ahmed Semih Özmekik, 171044039
April 12, 2020

Abstract

Demonstration of the solution of the given producer-consumer problem with inter process communications i.e. with signals.

Problem

In our problem, we first produce a data consisting of ascii bytes. Then we get a line equation for this data with a linear regression method. And with this line we create another data. The job of producing the correct equation from the data belongs to the first process. At the same time, these data must have error values according to some error metrics. Performing error scaling also represents the stage of conserving this data. And the consume operation must be done in a second process running simultaneously. So we need to solve the producer-consumer problem here through signals, which are primitive structures in inter process communication.

Solution

While solving the producer-consumer problem, we need to avoid poor design methods such as busy-waiting or sleep methods. The **SIGUSR2** signal left for users in linux can do our job here. We will see a **SIGUSR2** signal on the second process each time the producer produces, i.e when each line equation is produced. So first of all, **P2** checks whether the file is empty. If it is not empty, it continues to consume. But if it is empty, there are two situations. The first is that **P1** is already finished and no data will be produced again. The second is that **P1** is still producing.

First, let's consider the second situation. In this case, the file is empty, and **P2** needs to wait for production. While waiting for the production, he needs to suspend himself until the 2 signal comes. Of course, we will do this with the **sigsuspend** method. So if the file is empty and **P1** is still not finished, **P2** makes **sigsuspend**. And it waits until **SIGUSR1** arrives. Then he continues to consume when he wakes up.

In the first case, now we need another flag stating that **P1** is over. We will do this with the **SIGUSR1** signal. When **P1** is finished, it will send the **SIGUSR1** signal, which must be handled by **P2** (i.e. it's child). When it receives this signal it will set a flag and if this flag is set and the file is empty, it should not suspend in this case. Because now there is no **P1** left to tell him to wake up saying that he has produced it. In this case, he is not going to wait anymore.

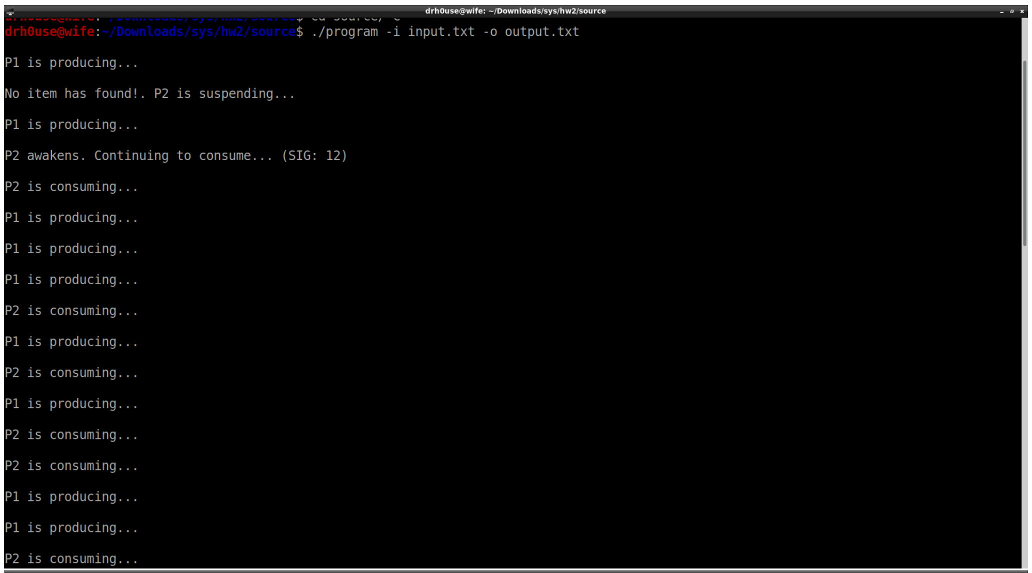
Now let's show the mentioned solution on the code. In the code, there are some parts of the producer or consumer that should not be interrupted in themselves, you can also observe them, but in this demonstration, the producer-consumer problem and signal handling are firstly mentioned.

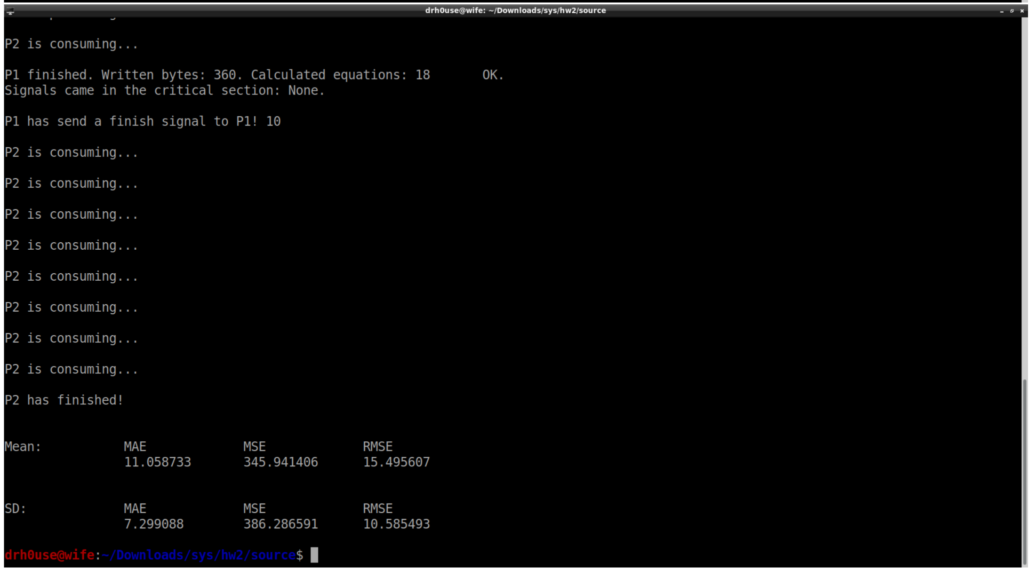
```c

void P1(int childID){
    int i = 0;
    struct line l; // no need for initialize.

    register_sighandler(SIGTERM, &P1_sigterm_handler);


    while (read20bytes(fd_in, &l) == BYTES){
        printf("\nP1 is producing...\n");
        // sleep(1);
        block_sig(SIGINT);
        block_sig(SIGSTOP);
        /* signal critical section */
        line_equation(&l);
        /* signal critical section */
        unblock_sig(SIGINT);
        unblock_sig(SIGSTOP);

        /* file critical section */
        write10points(fd_temp, &l);
        if (kill(childID, SIGUSR2) == -1) // wake up the consumer.
            xerror(__func__);
        /* end of file critical section */
        ++i;
    }

    close (fd_in);
    close (fd_temp);

    printf("\nP1 finished. Written bytes: %d. Calculated equations: %d\tOK.", i*BYTES, i);
    fflush(stdout);
    if (kill (childID, SIGUSR1) == -1) // signal to consumer that P1 is done.
        xerror(__func__);

}
```

```

void P2(){
    struct line l;

    stats.capacity = 10;
    stats.metrics = (struct metric*) xmalloc (sizeof (struct metric) * stats.capacity); //
    initial size.
    stats.idx = 0;

    register_sighandler(SIGUSR1, &handle_usr1);
    register_sighandler(SIGUSR2, &handle_usr2);
    register_sighandler(SIGTERM, &p2_sigterm_handler);

    unblock_sig(SIGUSR1); // unblock SIGUSR1, since now it has an handler.


    int fd_in_P2 = xopen(temp_name, O_RDWR);
    int i = 0;
    while (read10points(fd_in, &l)){
        printf("\nP2 is consuming...\n");
        /* end of file critical section */
        ++i;

        block_sig(SIGINT);
        block_sig(SIGSTOP);
        /* critical section */

        errors(&l, &stats);

        /* critical section */
        unblock_sig(SIGINT);
        unblock_sig(SIGSTOP);


        write10points_errors(fd_out, &l);
        /* file critical section */
    }

    printf("\nP2 has finished!\n");
    print_mean(&stats);
    print_SD(&stats);

    unlink(temp_name); // deletes the temp file.

    close (fd_in_P2});
    close (fd_out);
    free(stats.metrics);
    exit(EXIT_SUCCESS);
}

int P1_finished = FALSE;

void handle_usr1(int sig){
    printf ("\nP1 has finished! %d\n", sig);
    P1_finished = TRUE; // P1 has finished.
}

int read10points(int fd, struct line* l){
    char buf[512]; // at most 512 byte for a line
    const int N = BYTES / 2;
    int cont = FALSE;
    struct point* points = l->points;

    file_lock(fd);
    next_available(fd);
    cont = read_line(fd, buf);
    file_unlock(fd);
    if (cont) { // continue if there is input.
        char *temp = buf + 1;
        for (int i = 0; i < N; ++i)
            points[i] = buf2point(&temp);
        points[N] = buf2equation(&temp);
        file_lock(fd);
        remove_line(fd);
        file_unlock(fd);
    }
    else if (!P1_finished){ // there is no input. wait for a signal.
        printf("\nNo item has found!. P2 is suspending...\n");
        sigset_t sigusr;
        if (sigfillset (&sigusr) == -1) xerror(__func__);
        if (sigdelset (&sigusr, SIGUSR2) == -1) xerror(__func__);
        if (sigdelset (&sigusr, SIGUSR1) == -1) xerror(__func__);
        sigsuspend (&sigusr); // wait for SIGUSR2.
        block_sig(SIGUSR2); // block again.
        return read10points(fd, l);
    }
    return cont;
}
```

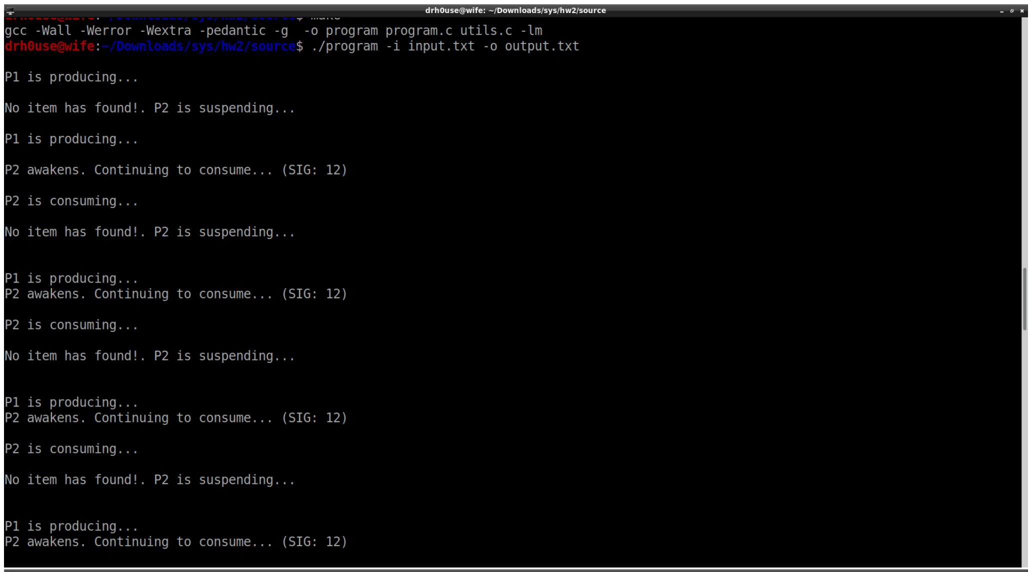
As can be seen, the **P1-finished** variable is the flag we use. The **read-line** function returns the variable **cont** to whether the file is empty or not. Then, if the file is empty, we check if **P1** has finished again. If **P1** has finished, **P2** now also ends. But if he's not done, he makes himself suspend.

Generally, there is an item to be consumed for **P2** in the temporary file, but of course not necessarily. In this case, the consumer, **P2**, (again, this is observation does not claim that this is the precise case.) Does not suspend himself. In the output below, we see that the flow of produce and consume continues as we watch it, one by one, in an order like producer produces and consumer consumes right after it.

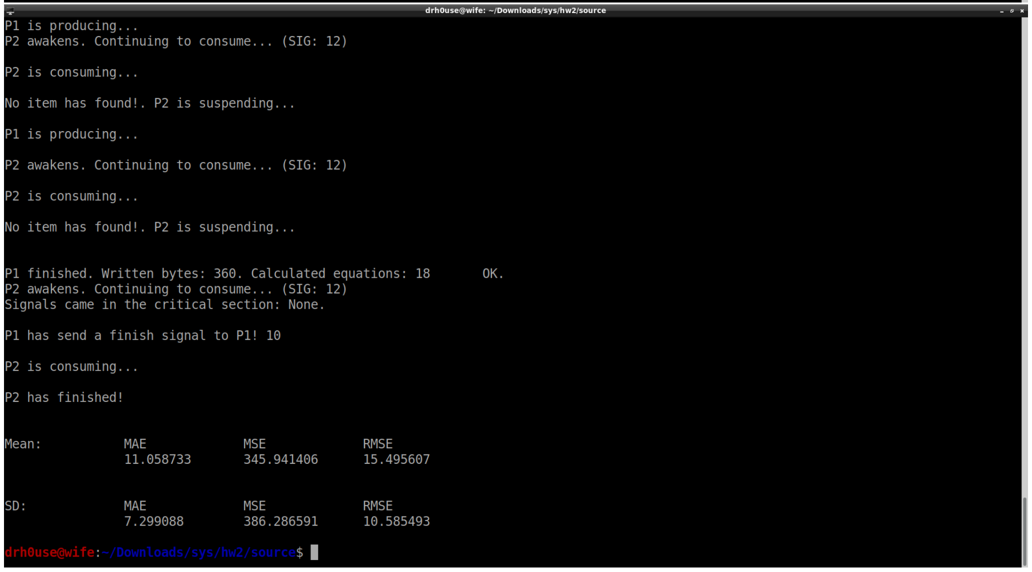

```
drh0use@wife:~/Downloads/sys/hw2/source$ ./program -i input.txt -o output.txt

P1 is producing...

No item has found!. P2 is suspending...

P1 is producing...

P2 awakens. Continuing to consume... (SIG: 12)

P2 is consuming...

P1 is producing...

P1 is producing...

P1 is producing...

P2 is consuming...

P1 is producing...

P2 is consuming...

P1 is producing...

P2 is consuming...

P2 is consuming...

P1 is producing...

P1 is producing...

P2 is consuming...
```


```
P2 is consuming...

P1 finished. Written bytes: 360. Calculated equations: 18       OK.
Signals came in the critical section: None.

P1 has send a finish signal to P1! 10

P2 is consuming...

P2 is consuming...

P2 is consuming...

P2 is consuming...

P2 is consuming...

P2 is consuming...

P2 is consuming...

P2 is consuming...

P2 has finished!


Mean:           MAE             MSE             RMSE
                11.058733       345.941406      15.495607


SD:             MAE             MSE             RMSE
                7.299088        386.286591      10.585493

drh0use@wife:~/Downloads/sys/hw2/source$ 
```

If we slow down the producer, that is, if we uncomment the **sleep** method in the **P1** code above, we observe that the consumer **P2** cannot find the item to consume in the file. In this case, it suspends itself, and when the producer reproduces, he is awakened and continues to consume. This situation was observed in the example below, and it was shown that inter-process communication was successfully provided.

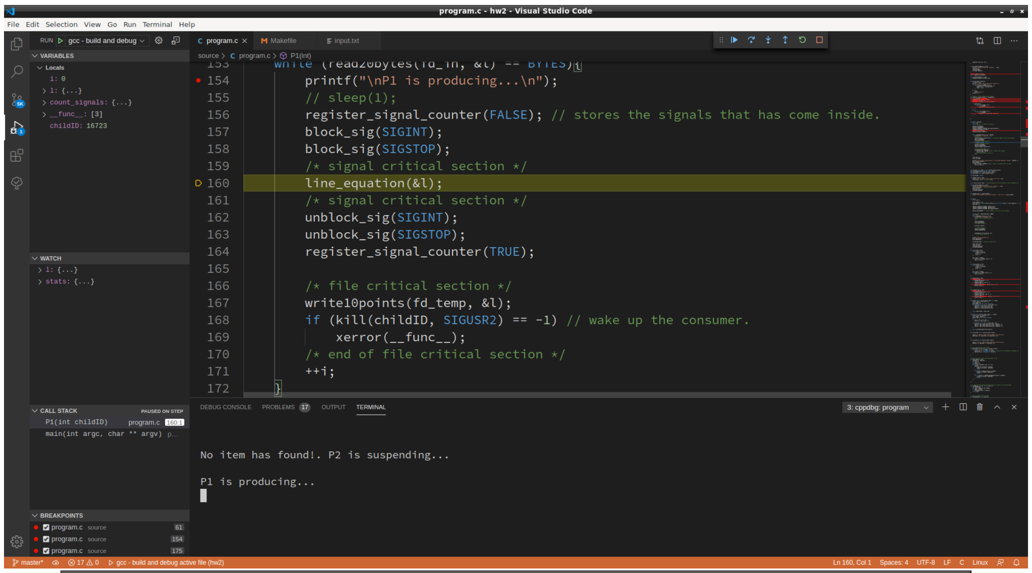


```
gcc -Wall -Werror -Wextra -pedantic -g  -o program program.c utils.c -lm
drh0use@wife:~/Downloads/sys/hw2/source$ ./program -i input.txt -o output.txt

P1 is producing...

No item has found!. P2 is suspending...

P1 is producing...

P2 awakens. Continuing to consume... (SIG: 12)

P2 is consuming...

No item has found!. P2 is suspending...


P1 is producing...
P2 awakens. Continuing to consume... (SIG: 12)

P2 is consuming...

No item has found!. P2 is suspending...


P1 is producing...
P2 awakens. Continuing to consume... (SIG: 12)

P2 is consuming...

No item has found!. P2 is suspending...


P1 is producing...
P2 awakens. Continuing to consume... (SIG: 12)
```



```
P1 is producing...
P2 awakens. Continuing to consume... (SIG: 12)

P2 is consuming...

No item has found!. P2 is suspending...

P1 is producing...

P2 awakens. Continuing to consume... (SIG: 12)

P2 is consuming...

No item has found!. P2 is suspending...


P1 finished. Written bytes: 360. Calculated equations: 18       OK.
P2 awakens. Continuing to consume... (SIG: 12)
Signals came in the critical section: None.

P1 has send a finish signal to P1! 10

P2 is consuming...

P2 has finished!


Mean:           MAE             MSE             RMSE
                11.058733       345.941406      15.495607


SD:             MAE             MSE             RMSE
                7.299088        386.286591      10.585493

drh0use@wife:~/Downloads/sys/hw2/source$
```

Another important task is the task of recording the signals received by the **P1** in the critical region when calculating line equations. In order to test this, we first waited in the critical region in the debugger, and then we sent 2 signals from the shell to this program. Then, as can be seen in blow, the two signals we sent were recorded and then printed out.


program.c - hw2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN gcc - build and debug
program.c Makefile input.txt
source > program.c > P1(int)
VARIABLES
Locals
WATCH
CALL STACK
PAUSED ON STEP
BREAKPOINTS
DEBUG CONSOLE PROBLEMS 17 OUTPUT TERMINAL
3: cppdbg: program
master* 17 0 gcc - build and debug active file (hw2)
Ln 160, Col 1 Spaces: 4 UTF-8 LF C Linux

```
i: 0
> l: {...}
> count_signals: {...}
> __func__: [3]
childID: 16723
```

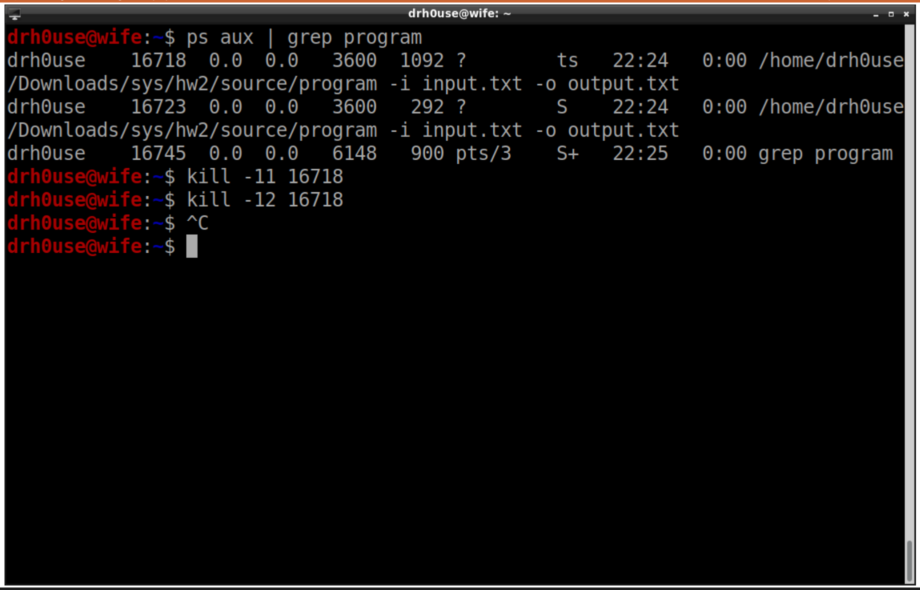
```
> l: {...}
> stats: {...}
```

```
P1(int childID)    program.c 160:1
main(int argc, char ** argv) p...
```

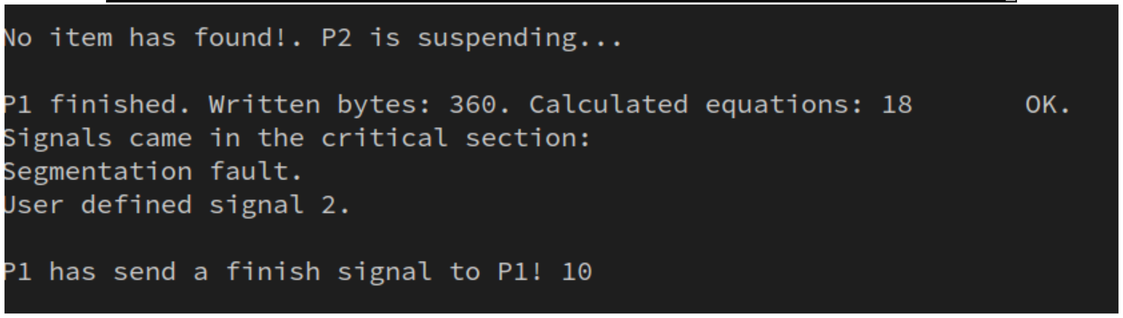
```
program.c source 61
program.c source 154
program.c source 175
```

```c
while (read20bytes(fd_in, &l) == BYTES){
    printf("\nP1 is producing...\n");
    // sleep(1);
    register_signal_counter(FALSE); // stores the signals that has come inside.
    block_sig(SIGINT);
    block_sig(SIGSTOP);
    /* signal critical section */
    line_equation(&l);
    /* signal critical section */
    unblock_sig(SIGINT);
    unblock_sig(SIGSTOP);
    register_signal_counter(TRUE);

    /* file critical section */
    write10points(fd_temp, &l);
    if (kill(childID, SIGUSR2) == -1) // wake up the consumer.
        xerror(__func__);
    /* end of file critical section */
    ++i;
}
```

```
No item has found!. P2 is suspending...

P1 is producing...
```



```
drh0use@wife:~$ ps aux | grep program
drh0use    16718  0.0  0.0   3600  1092 ?        ts   22:24   0:00 /home/drh0use
/Downloads/sys/hw2/source/program -i input.txt -o output.txt
drh0use    16723  0.0  0.0   3600   292 ?        S    22:24   0:00 /home/drh0use
/Downloads/sys/hw2/source/program -i input.txt -o output.txt
drh0use    16745  0.0  0.0   6148   900 pts/3    S+   22:25   0:00 grep program
drh0use@wife:~$ kill -11 16718
drh0use@wife:~$ kill -12 16718
drh0use@wife:~$ ^C
drh0use@wife:~$ 
```



```
No item has found!. P2 is suspending...

P1 finished. Written bytes: 360. Calculated equations: 18        OK.
Signals came in the critical section:
Segmentation fault.
User defined signal 2.

P1 has send a finish signal to P1! 10
```