

Hw1 Report

Muhammed Yasir Fidan

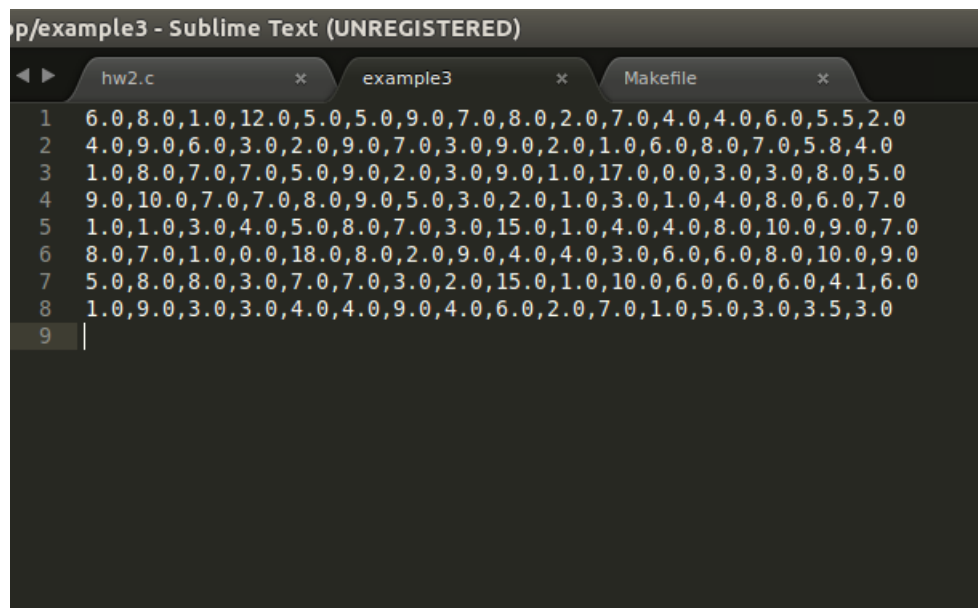
April 2021

1 Report

Compile and Run

Firstly I compile my program with `gcc -o processM hw2.c -std=gnu99 -Wall -D_BSD_SOURCE -D_POSIX_C_SOURCE=199309L -lm` in makefile, so there is no warning when using `-Wall` command. After type `make` and compile program, for running enter `./processM` filepath like in homework pdf. filepath is the path of file.

Before running program file look like this. There is a newline at end of row 8 in teacher format file so I write my code according this format. There must be a newline at row 8's end, like in teacher's example file.



```
p/example3 - Sublime Text (UNREGISTERED)
hw2.c x example3 x Makefile x
1 6.0,8.0,1.0,12.0,5.0,5.0,9.0,7.0,8.0,2.0,7.0,4.0,4.0,6.0,5.5,2.0
2 4.0,9.0,6.0,3.0,2.0,9.0,7.0,3.0,9.0,2.0,1.0,6.0,8.0,7.0,5.8,4.0
3 1.0,8.0,7.0,7.0,5.0,9.0,2.0,3.0,9.0,1.0,17.0,0.0,3.0,3.0,8.0,5.0
4 9.0,10.0,7.0,7.0,8.0,9.0,5.0,3.0,2.0,1.0,3.0,1.0,4.0,8.0,6.0,7.0
5 1.0,1.0,3.0,4.0,5.0,8.0,7.0,3.0,15.0,1.0,4.0,4.0,8.0,10.0,9.0,7.0
6 8.0,7.0,1.0,0.0,18.0,8.0,2.0,9.0,4.0,4.0,3.0,6.0,6.0,8.0,10.0,9.0
7 5.0,8.0,8.0,3.0,7.0,7.0,3.0,2.0,15.0,1.0,10.0,6.0,6.0,6.0,4.1,6.0
8 1.0,9.0,3.0,3.0,4.0,4.0,9.0,4.0,6.0,2.0,7.0,1.0,5.0,3.0,3.5,3.0
9
```

Figure 1: File before running

After executing program file will be like this

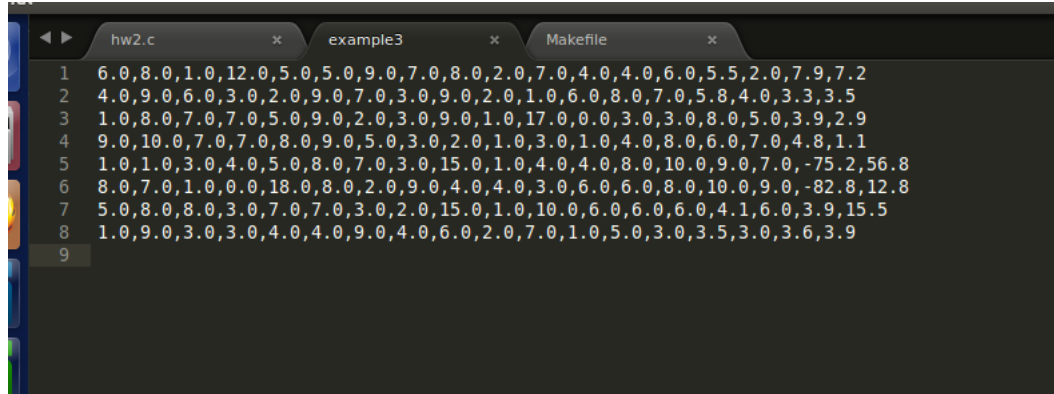


Figure 2: File after running

As you can see, after program run each row has 2 more point at the end of the row. New first number is Lagrange of x7 according to first 6 coordinate and second new number at the end of row is Lagrange of x7 according to first 7 coordinate. All row's Lagrange calculated by one child process. For example, first child process calculate first row lagrange for first 6 coordinates and found 7.9 then write it, Then in second round it uses first 7 coordinates and found 7.2 for lagrange of x7 and write it file. Every 8 process calculate and write their results to file like this and so on.

Also After running program, Parent process write error for first round and second round in terminal, Also write each polynomial coefficients.

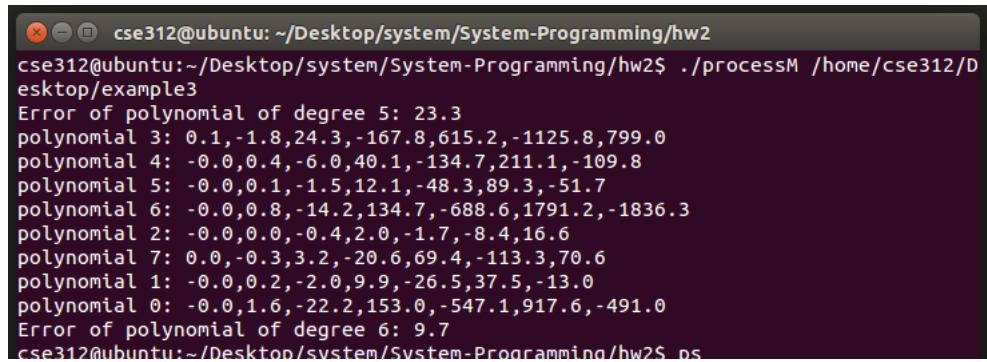


Figure 3: Terminal after program run

How did I solve file synchronization

There is 8 process that read and write file, So there must be a synchronization for reading and writing. I use fcntl for locking file as suggested in homework pdf, so each child process can read and write file. If there hadn't a file locking system, for example multiple process would read file and change offset or may write at the same time and cause corrupt file. Locking file solve these problems.

How did I solve parent-child synchronization

In this homework, after child write their first calculated value to the file, they must wait and parent must continue, then parent wait and children processes continue. So parent and child must synchronize by using signals. I use only SIGUSR1 signal and sigsuspend for synchronization between parent and child processes. In first part parent use sigsuspend 8 times for waiting SIGUSR1 in his 8 child processes. After child processes calculate their lagrange and write it to the file they send SIGUSR1 by using kill system call to the parent. But there is very tricky problem. What if 2 or more child process send SIGUSR1 signal to parent. In this case because there is no signal pending for more than 1 signal, parent process will execute sigsuspend for only 1 signal, so it lose some children signal and wait infinitely. For solving this problem, I send signal from child before unlocking file and after parent receive signal it send SIGUSR1 to this child to let them know parent receive their signal so they can continue. So with this design parent process can take all 8 signals and then it can continue and children processes once again execute sigsuspend for waiting parent. So all of the children now wait parent to calculate error of degree 5. After parent calculate error of degree 5 it prints out terminal and send SIGUSR1 all children to make them continue. Now all children process once again continue and calculate new lagrange for their row. After that like in first round, they send signals to parent and wait. After parent continue and print error of polynomial of degree 6 it sends SIGUSR1 to all children to make them continue. Then each child calculate coefficient of each polynomial and print them to screen and terminate.

```
//Waiting 8 signal from childrens with sigsuspend
for(i = 0; i<8; i++){
    sigsuspend(&sigset);
    kill(sender_pid,SIGUSR1);
}
```

Figure 4: Parent wait 8 SIGUSR1 from children

Like above screenshot, Parent process execute 8 sigsuspend to collect all 8 child signal before continue and after it take a signal it send a SIGUSR1 to child process that send SIGUSR1. So if child receive SIGUSR1 it means parent take his signal without any problem.

```

}
//send sigusr1 to parent
//printf("Singal sending %d\n",i );;
kill(getppid(),SIGUSR1);
sigsuspend(&sigset);

if(fcntl(fd,F_SETLK,&lock) == -1){
    perror("fcntl error");
    exit(1);
}

//Wait signal from parent to continue
sigsuspend(&sigset);

```

Figure 5: Child process send SIGUSR1 to parent

And above screenshot shows that, child process send SIGUSR1 to parent process then suspend for coming signal from parent. After parent send signal it means, parent take his signal without any problem, so child process unlock file and execute sigsuspend once again. So with this design all child process send their signal one by one so parent process can't losing any signal. After parent take all 8 signal it will continue and all child process will be in second sigsuspend for waiting parent.

```

411
412     parent_calculate_error(fd,7);
413
414
415     //Sending sigusr1 to children
416     //So they can continue
417     for(int i=0; i<8; i++) kill(child_process[i],SIGUSR1);
418
419

```

Figure 6: Parent calculate error and send signals to children to make them continue

Now parent calculate error for degree 5 and print it to terminal meanwhile all 8 children processes wait parent in sigsuspend. So after parent done it using kill system call to send all 8 children SIGUSR1 signal so know they can continue for second part.

Second part again work like this. Again children read and write file for their row's lagrange also they print coefficient of polynomials too. Then parent continue and print error degree of 6.

How did I handle zombie process

As mentioned in homework pdf, when a child process terminated a SIGCHLD signal send automatically. So in handler I use wait like we learn in lecture to clean zombie processes.

```
44  */
45  static void handler(int sig, siginfo_t *siginfo, void *context){
46      switch(sig){
47          case SIGUSR1:
48              sender_pid = siginfo->si_pid;
49              break;
50          case SIGCHLD: ;
51
52              int status;
53              wait(&status);
54              child_exit_status = status;
55              break;
56      }
57  }
58
```

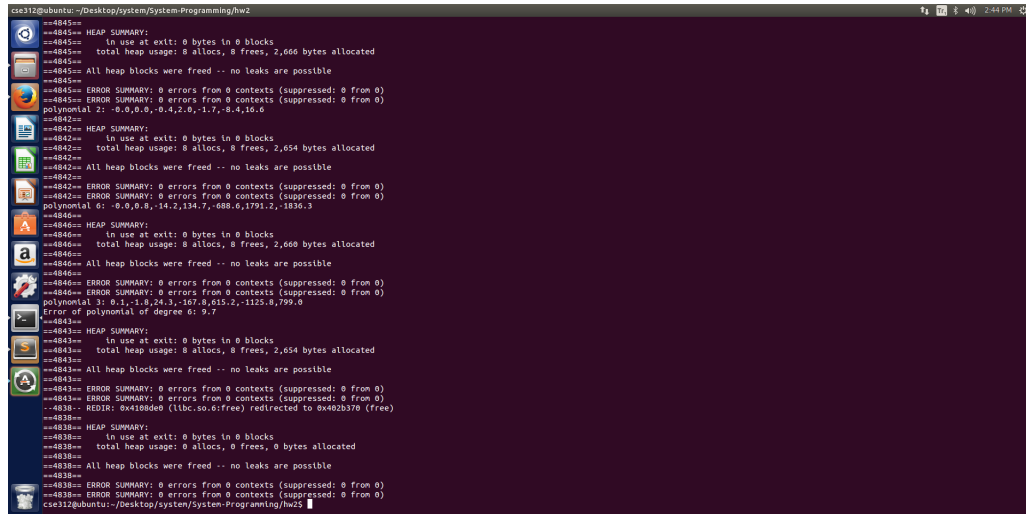
Figure 7: Signal handler call wait for SIGCHLD signal

Calculation of polynomial's coefficients

I use gauss elimination method for finding 7 coefficients. Each child process find their corresponding row coefficients by using gauss elimination method.

Valgrind example

When I execute my program with valgrind leak-check=full -v the output like this:



```

cs312@ubuntu: ~/Desktop/system/System-Programming/hw2
==4845== HEAP SUMMARY:
==4845==    in use at exit: 0 bytes in 0 blocks
==4845==   total heap usage: 8 allocs, 8 frees, 2,666 bytes allocated
==4845== All heap blocks were freed -- no leaks are possible
==4845== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==4845== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
polynomial 2i -0.0,0.0,-0.4,2.8,-1.7,0.4,16.0
==4842==
==4842== HEAP SUMMARY:
==4842==    in use at exit: 0 bytes in 0 blocks
==4842==   total heap usage: 8 allocs, 8 frees, 2,654 bytes allocated
==4842== All heap blocks were freed -- no leaks are possible
==4842== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==4842== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
polynomial 2i -0.0,0.0,-0.4,2.8,-1.7,0.4,16.0
==4846==
==4846== HEAP SUMMARY:
==4846==    in use at exit: 0 bytes in 0 blocks
==4846==   total heap usage: 8 allocs, 8 frees, 2,660 bytes allocated
==4846== All heap blocks were freed -- no leaks are possible
==4846== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==4846== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
polynomial 2i -0.1,1.0,24.0,-167.0,615.2,-1125.0,799.0
Error of polynomial of degree 6: 9.7
==4843==
==4843== HEAP SUMMARY:
==4843==    in use at exit: 0 bytes in 0 blocks
==4843==   total heap usage: 8 allocs, 8 frees, 2,654 bytes allocated
==4843== All heap blocks were freed -- no leaks are possible
==4843== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==4843== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
..4838.. REDIR: 0x4188de0 (libc.so.6:free) redirected to 0x402b370 (free)
==4838==
==4838== HEAP SUMMARY:
==4838==    in use at exit: 0 bytes in 0 blocks
==4838==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==4838== All heap blocks were freed -- no leaks are possible
==4838== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
==4838== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cs312@ubuntu: ~/Desktop/system/System-Programming/hw2

```

Figure 8: Valgrind leak-check=full -v example

As you can see there is no leak possible

Also I only lock files when reading and writing, for example when calculating lagrange formula for both round 1 and round 2 there is no lock so multiple process can calculate their row lagrange at the same time.

```

//printf("%s",buffer );

lock.l_type = F_UNLCK;
if(fcntl(fd,F_SETLK,lock) == -1){
    perror("fcntl error");
    exit(1);
}

char* useStrForToken = calloc(strlen(buffer)+1, sizeof(char));
strcpy(useStrForToken, buffer);

char* strNewRow = calloc(strlen(buffer)+100, sizeof(char));
strcpy(strNewRow, buffer);

//Arrays for first 6 coordinate
double x_first_6[6];
double y_first_6[6];
double x_7;
//Take 6 point
take_first_6_point(useStrForToken,x_first_6,y_first_6,6);
x_7 = take_x_7(buffer);
double res_6 = calculate_lagrange(x_first_6,y_first_6,6,x_7);
create_new_row(strNewRow,res_6);

```

Figure 9: No lock when calculating lagrange