CSE 344 HW2 REPORT

In this homework, we're asked to create two process that one of them writes to a file and the other one reads from that file. Process created by another process using fork system call. This processes must work concurrently. This means files that shared must be lock when one of the process work on them. Also, since there is a producer-consumer relationship, consumer must wait until producer produce a input for consumer. Also, second process must know process 1 is done, so it can delete files of process 1. To notify process 2 we asked to use SIGUSR1 signal. There is also a constraint that we must block SIGSTOP and SIGINT at critical sections where calculations made. We can easily block SIGINT but SIGSTOP signal cannot be blocked. SIGSTOP is a special signal used for stopping a process and managed by kernel space. From manual page:

`The signals **SIGKILL** and **SIGSTOP** cannot be caught, blocked, or ignored.`

First I created, command-line argument parsing code and I get input and output paths. If there is missing arguments and and unknown options, code will print usage message and exit using _exit system call. If everything goes fine code will create filenames for open. In there, I get file names and assigned them to char arrays but since input file must be deleted when SIGTERM is delivered, I created that array globaly to remove input file inside signal handler. I used O_CREAT option for output file so maybe it doesn't exist and need to be created. After that I created temp file in main. Purpose of that both of process will use that file so both of them will have copy of file descriptor of that temp file. Also I used unlink call whenever temp file closed or process exited gracefully, this file will be deleted.

After that I made signal dispositions. I created a sigaction struct and give handler for some signals. Also I setted SA_RESTART flag so when we do an system call if it interrupted by a signal after signal handler return, this system call will restart. For SIGTERM handling both process exit gracefully and remove input file and I used _exit syscall to exit. For SIGUSR1, I created a handler that uses a global variable *existence*. By doing that if process one is done, it will send continuously SIGUSR1 to process 2 and process to know by the value of *existence* status of parent process.

After that I created a set of masks and I initialized a mask for SIGALRM and SIGUSR1. By doing that if process 1 is send signals that required for continuous of process 2 before this block mask set, signal may lost. By doing that after relasing block, I can show which signals come to process 2. After that fork will be done.

In process 1, I created a signal mask for critical section. After that I start a endless loop and readed 20 bytes from input file. Every 20 bytes process will do a calculation. If readed bytes is less than 20, this means we reach end and there no available input for process. To print total lines at every iteration I incremented line, and added total readed byte to a variable. After that I set signal mask and enter the critical region. In there a send the readed bytes to a function with a line array that will store created line. In that function, I used readed bytes to calculate line and I putted coordinates in a given format to a line array. I putted line at the end of line with three decimal points. Before exiting critical section, to understand which signals are send to process, I used sigpending function and look for pending signals. If there is signals that blocked, I indicated that to print later.

After exiting critical section, I locked temp file to preventing any clash with child process. After write lock setted to file, I seeked file at the end. I writed new lines to end of file. By doing that with, in child process I can read from first line and I can create a FIFO like system. After writing to temp file I released lock and send SIGALRM signal to child process. Purpose of that signal is whenever child process doesn't find any consumable input and knows that process 1 is still exist, it will suspended until input provided. By that signal process 1 say "I filled file" to process 2. After processing all file and printing needed information, I created a do-while loop and wait for child to exit. I do that using waitpid and I setted WNOHANG option to send SIGUSR1 signal constantly. Purpose of doing that, not to doubt that acknowledge process 2 for process 1 is done.

In process 2, I created some metric error array for calculating errors means and standard deviations. After that I setted a wait mask, to suspend process until any input provided. If the input provided to temp file from process 1, process 2 start processing all temp file. In a endless loop, first since SIGUSR1 is blocked, I looked pending signals to understand process 1 is done or not. If it is done, I setted *existence* variable to 0 to indicate that. After that I locked temp file for preventing any clash. Since I always readed from first line I seeked temp file to beginning. Since temp file shared, offset may changed by process 1. From there we can read BUFFER_SIZE byte, and look for read byte status. If read byte status 0 this means temp file is empty. If this is the case and existence is 0, this implies that process 1 is done, there will be no input and temp file is empty, we can break loop and go to end processes. If existence is 1, this means temp file currently empty but there will be input available later. So process 2 must suspend until process 1 notify process 2 to say that temp is filled you can continue. To do that I used wait mask with sigsuspend. This provided suspending of process 2 until SIGALRM delivered. Of course, before doing that I released lock of temp file to process 1, fill the temp file. After sigsuspend, since we couldn't read any line, I passed all next steps turn back beginning of loop by using continue. If any of cases not occur, we successfully readed a line. From there may be the line consist of less number of char than BUFFER_SIZE. We need to consider a line until encounter with new line character. This place will indicate end of line. I founded that line and, seeked file after that newline char to delete the this line since it is readed. After that I send file with this offset to function. This file overwrite lines to above lines. By doing that I can write all lines to above lines. After that since last line will not be overwrited, I truncate file with writed byte number, so last line will be removed. After these are all done I released lock. Now, since we have line we can calculate error metrics, I sended this line to a function with *errorMetric* parameter that store error metric results as a char array and a struct which stores each error metric. By using this struct we have packet that stores metrics and we can save them to calculate mean and derivation. Before sending line to function, since this is a critical region I setted a mask for that place that blocks given signals. After that before ending critical region, I concatenate this newly calculated error metrics with readed line and I writed this new line to output file. At the end I saved metrics for that line to array of that metrics, If this array overflows its size, I reallocate place for new coming lines. After processing temp file. I calculated derivations and means to sending arrays to a function and printed them on to screen. At the end of process 2, I removed input file and end all the process by exiting.

To start program you type *make* to terminal. After that usage is:

./program -i [inputPath] -o [outputPath]