Brief Introduction for Mergesort

***THOUGHT IN MIND:**

I felt that this homework is the most tough homework throughout the whole semester. Maybe, it is because that I am really not familiar with signals. My code is a little ugly this time. I believe that I have spent more than fifteen hours on it in three days!!!

***FEATURE:**

1.I use the function "epoll" instead of the combination of "select" or "pipe2 + sleep + alarm", because that I felt the signals is quite not stable and may lead some exceptions happened.

2. I implement "non-blocking" with "epoll + fcntl(fd, F_SETFL, O_NONBLOCK)".

3. I simulate "multi-alarm" with epoll()'s timer.

***ROUTINE:**

4. I first analyze the parameter, and create a file named "data" for the sack of storing temp information, such as all the input.

5. Second, I will pre-calculate the quantity of data that each blocks' size.

6. Then, build up two times of NUM_PROCESS pipes, and then create NUM_PROCESS child processes, each of them exec() the sorting file. After that, I register the event of epoll.

7. Now, I formally begin dealing with the data. Everytime, I will wait for a ready event.
   1) If the event is a "EPOLLOUT" event, I will read some data from the temp file and give them to the target child. Here are two conditions, one is that the block of the data has been totally read, then I will register the "EPOLLIN" event, or, I will keep registering the "EPOLLOUT" event.
   2) Else if the event is a "EPOLLIN", I will read the sorted data from the target child, and overwrite the same data block in the temp file. The two conditions thereby is quite similar as the former one, which conducts just adversely.

8. If there are some exceptions conditions occurred, I will judge the invalid signals as SIGPIPE, and then print out the message. After that, I will restart the certain child.

9. If there are some processes timeout, I will also restart these children after killing them.

10. After all the blocks of data has been overwritten, I will begin to "mergesort".
   1) My method is that I will hold a priority_queue with the size of the number of the blocks and record the flags for all the blocks of data. Thus, the space-complexity of my heap is O(NUM_BLOCKS).
   2) Everytime, I will pop the minimum item from the heap, and keep doing so for "k" times. When I pop out an element, I will check where it comes from, then move the flag and add a new item inside the heap if it is a valid operation. (which means that it won't beyond the boundary)

11. Finally, I will print out a number, which indicates the "k-th" number throughout all the data.

***ERROR HANDLER:**

12. If the number of input parameter is incorrect, you will receive “ARGUMENT_NUM ERROR”, and if there is invalid argument, you will receive “ARGUMENT ERROR”. The result after the error is not guaranteed.

13. If the “k” you request is too big, then, it will be judged illegal requirement, and you will receive “Your target is too far”.

14. “SIGALRM” will be print if some child is halting for a too long time, and “SIGPIPE” will be print if SIGPIPE do occur, or some illegal message is received, for example, “EPOLLHUP”.

15. There might be other error messages, which is printed by the function “perror()”.

I appreciated the TAs' and Teacher's efforts throughout the whole semester;

I did learn a lot from you, thanks!!!