ITP 30002-02 Operating System, Spring 2020

Homework 2

Park Ye Gyeom/21600277/21600277@handong.edu

1. Introduction

The objective of this project is to learn how to operate parallel child processes using fork() through the process of implementing ptsp.c and how to read/write data through unnamed pipes (IPC) between parent and child processes. Studying fork and how a parent can create multiple child processes were first required while carrying out this task. It also needed to think about how to set prefixes (re-renew them). It also needed to think about parents and children reacting differently to the same kind of signal that occurs in the parent and child processes. Next, it was necessary to study the unnamed pipe that allowed each child's process to write to the parent process the path of the peak value and the number of the child's count to the parent process and allow the parent to read the data. And in the process of implementing unnamed pipes, it was necessary to consider at what point to read/write.

2. Approach

Let's look at the workflow of the parent process, the child process. The task first defined the work flow of the parent process as follows:

1. Check the number of child process currently created and if the current number of children is less than max_process_num, use fork() as a predetermined prefix value to create a child with that prefix value. In this process, the child's prefix will be fixed and will be able to work as much as 12! without being overlapped with other children.
2. If the number of child processes is not sufficient, redo the fork as needed.
3. At the end of the child process, update the values to read and output the data sent from the signal handler to the unnamed pipe. It also reduces the current number of children by one, allowing the fork() to be performed again. In this way, the maximum number of children is kept.
4. If there is no longer a path to perform, print the best values and the total count without performing the fork(). The parent process should not end until all child processes are terminated, so the wait process in the SIGCHLD handler ensures that the parent updates the values to the last minute and outputs the best value.

The work flow of the child process is then as follows.

1. Perform the remaining 12! subtask based on the prefix set by the parent.
2. 12! Perform all assigned subtasks, or until a terminal signal occurs in the middle continue to update the optimal value and count values.
3. If you have performed 12! subtask, write down the best results and count you have found to your parents and end it.
4. If the assigned subtask is not fully performed and terminated due to the terminal signal, write the best results and count to the parents at the signal handler.

Unnamed pipe

- Use unnamed pipes for reading and writing data between parent and child processes. Parent processes and subprocesses do not share memory regions. Therefore, the child process should use unnamed pipes to update the values obtained from the parents. In this task, one parent must create 1 to 12 child processes. Therefore, the parent process generated pipes as much as the maximum_process_num previously entered, and connected unnamed pipes associated with the child and the parent process when performing the fork(). Also, children who finished writing to their parents carried out the mission of emptying pipes so that the next generation of children could be connected to their parents.

Signal

- SIGINT and SIGCHLD were the signals that needed to be caught during this task. First, SIGINT is the signal that interrupts occur while the process is running and terminates the process. Therefore, SIGINT signals were generated in the parent and child processes, and signal handlers were installed to handle them. When a SIGINT signal occurred in the parent process, the SIGCHLD signal was raised to the user, the best value, path, total count of the data from the child was printed, and the handler was installed to terminate the process. And when the SIGINT signal occurred, the child process installed a handler that wrote the best value, path and count to the parent through unnamed pipe and ended the process. Next, SIGCHLD is the signal received by the parent process. This signal is a signal to parents that they are terminated when their children are terminated. Therefore, a signal handler was installed to read data sent by children when SIGCHLD occurred.

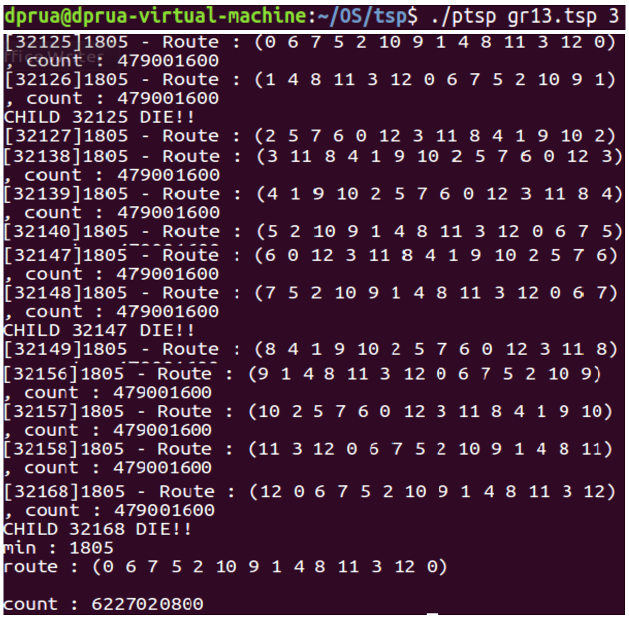
How the parent process assigns tasks to children

- The parent process performs forks only by max_process_num initially designated as input values. If the process is generated by max_process_num, the parent process no longer performs a fork and waits outside the repeating statement. The parent process reduces the current number of children by 1 to perform the fork() again when the SIGCHLD handler is activated. Therefore, the number of children will remain as specified by the max_process_num initially. Before creating a child process with a fork, the parent executes the function of obtaining the following prefixes so that the generated child can perform sub-tasks with that prefix.

3. Evaluation

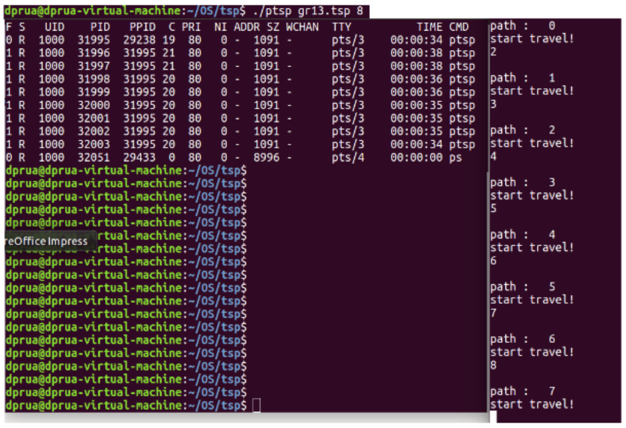
Functionalities

- The program was executed using gr13.tsp and the results were checked to ensure that the program was terminated normally when all parental work was completed.
- Ensure that the number of child process is 12! and that the prefix for each child process does not overlap.
- When the child is terminated, write the value to the parent with unnamed pipe, and the parent verifies that the value of the child has been read and printed.



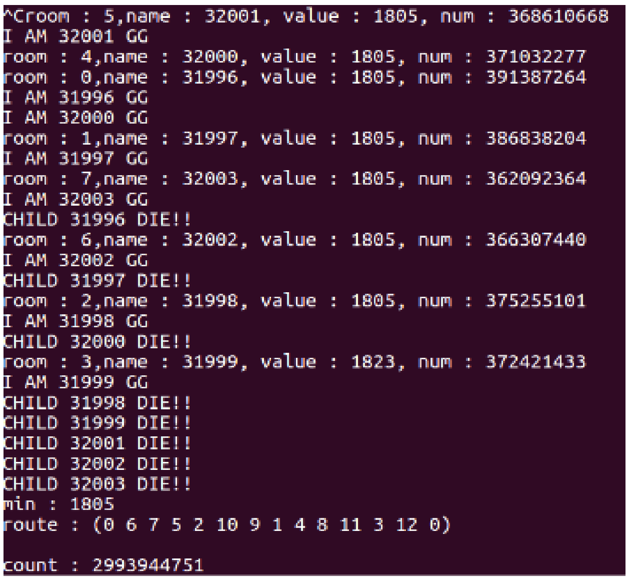
```
dprua@dprua-virtual-machine:~/OS/tsp$ ./ptsp gr13.tsp 3
[32125]1805 - Route : (0 6 7 5 2 10 9 1 4 8 11 3 12 0)
, count : 479001600
[32126]1805 - Route : (1 4 8 11 3 12 0 6 7 5 2 10 9 1)
, count : 479001600
CHILD 32125 DIE!!
[32127]1805 - Route : (2 5 7 6 0 12 3 11 8 4 1 9 10 2)
[32138]1805 - Route : (3 11 8 4 1 9 10 2 5 7 6 0 12 3)
, count : 479001600
[32139]1805 - Route : (4 1 9 10 2 5 7 6 0 12 3 11 8 4)
, count : 479001600
[32140]1805 - Route : (5 2 10 9 1 4 8 11 3 12 0 6 7 5)
[32147]1805 - Route : (6 0 12 3 11 8 4 1 9 10 2 5 7 6)
, count : 479001600
[32148]1805 - Route : (7 5 2 10 9 1 4 8 11 3 12 0 6 7)
, count : 479001600
CHILD 32147 DIE!!
[32149]1805 - Route : (8 4 1 9 10 2 5 7 6 0 12 3 11 8)
[32156]1805 - Route : (9 1 4 8 11 3 12 0 6 7 5 2 10 9)
, count : 479001600
[32157]1805 - Route : (10 2 5 7 6 0 12 3 11 8 4 1 9 10)
, count : 479001600
[32158]1805 - Route : (11 3 12 0 6 7 5 2 10 9 1 4 8 11)
, count : 479001600
[32168]1805 - Route : (12 0 6 7 5 2 10 9 1 4 8 11 3 12)
, count : 479001600
CHILD 32168 DIE!!
min : 1805
route : (0 6 7 5 2 10 9 1 4 8 11 3 12 0)

count : 6227020800
```

- Use the Signal Handler to ensure that the parent process does not terminate before the child process, and that the values are printed before parent procees terminated.
- If a signal is generated through crtl+c during a process run, compare the values stored by each child to see if the best value and store it. And print that value and total count.



```
dprua@dprua-virtual-machine:~/OS/tsp$ ./ptsp gr13.tsp 8
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R  1000 31995 29238 19  80   0 -  1091 -      pts/3    00:00:34 ptsp
1 R  1000 31996 31995 21  80   0 -  1091 -      pts/3    00:00:38 ptsp
1 R  1000 31997 31995 21  80   0 -  1091 -      pts/3    00:00:38 ptsp
1 R  1000 31998 31995 20  80   0 -  1091 -      pts/3    00:00:36 ptsp
1 R  1000 31999 31995 20  80   0 -  1091 -      pts/3    00:00:36 ptsp
1 R  1000 32000 31995 20  80   0 -  1091 -      pts/3    00:00:35 ptsp
1 R  1000 32001 31995 20  80   0 -  1091 -      pts/3    00:00:35 ptsp
1 R  1000 32002 31995 20  80   0 -  1091 -      pts/3    00:00:35 ptsp
1 R  1000 32003 31995 20  80   0 -  1091 -      pts/3    00:00:34 ptsp
0 R  1000 32051 29433  0  80   0 -  8996 -      pts/4    00:00:00 ps
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
reOffice Impress rtual-machine:~/OS/tsp$
                 rtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
dprua@dprua-virtual-machine:~/OS/tsp$
```

```
path :   0
start travel!
2

path :   1
start travel!
3

path :   2
start travel!
4

path :   3
start travel!
5

path :   4
start travel!
6

path :   5
start travel!
7

path :   6
start travel!
8

path :   7
start travel!
```



```
^Croom : 5,name : 32001, value : 1805, num : 368610668
I AM 32001 GG
room : 4,name : 32000, value : 1805, num : 371032277
room : 0,name : 31996, value : 1805, num : 391387264
I AM 31996 GG
I AM 32000 GG
room : 1,name : 31997, value : 1805, num : 386838204
I AM 31997 GG
room : 7,name : 32003, value : 1805, num : 362092364
I AM 32003 GG
CHILD 31996 DIE!!
room : 6,name : 32002, value : 1805, num : 366307440
I AM 32002 GG
CHILD 31997 DIE!!
room : 2,name : 31998, value : 1805, num : 375255101
I AM 31998 GG
CHILD 32000 DIE!!
room : 3,name : 31999, value : 1823, num : 372421433
I AM 31999 GG
CHILD 31998 DIE!!
CHILD 31999 DIE!!
CHILD 32001 DIE!!
CHILD 32002 DIE!!
CHILD 32003 DIE!!
min : 1805
route : (0 6 7 5 2 10 9 1 4 8 11 3 12 0)

count : 2993944751
```

Performance measurement of ptsp.c

- Performance measurement methods will be used as a performance evaluation index for counting counts per unit hour.
- Set the number of child processes, run the program for a set period of time, and compare the performance by looking at the count under each condition.

process num	count
1	125000054
2	256046036
3	366448555
4	488080683
5	553586710
6	632259875
7	747794025
8	800795571
9	878718796
10	968652886
11	1087882909
12	1161690635

Run the program with gr17.tsp. After 30 seconds, the program was terminated using ctrl+c and the total count was compared. As a result, the process increased one by one, we could see that about 100 million more routes were being explored. It is expected that if you shorten the process creation time, we will be able to navigate more paths each time the number of processes increases.

4. Discussion

The process of writing and reading values through a pipe seemed complicated and inefficient because parents and children did not share memory. If the same program is implemented using thread, the threads will share the memory area, so they will be able to process the values much more efficiently and faster than the pipes.

5. Conclusion

The project implemented a method of reading and writing data between parent and children using unnamed pipes and signals so that parent and children can know each other's condition and act accordingly. To facilitate the exchange of information between parents and children, unnamed pipes were made as many as the number of children. And we were able to implement signal handlers that catch the SIGINT signal and SIGCHLD signal to fulfil the conditions necessary for this project. And through the process of creating multiple child processes through fork(), I was able to experience the efficiency and speed up of work by dividing one main problem into sub problems.