ITP 30002-01 Operating System, Spring 2020

**Homework 2**

**Name** : Min Jae Yi (이민재) **Student number** : 21800511 **Email Address** : 21800511@handong.edu

## 1. Introduction

The main part of the hw2 can divided into three parts making "main process", making "child process" and connecting "main process" and "child processes" by IPC mechanisms. And in the IPC mechanisms "unnamed pipes" and "signal" was used.

The requirement or function for the c program can be shown as below.

| |
|---|
| *Main process* |
| 1-1. Create child processes by the given number. |
| 1-2. Maintain number of child processes until there is no other task. |
| 1-3. Give next subtask to the child process by giving the next Prefix. |
| 1-4. If the child process is terminated received the best solution for the child's subtask. |
| 1-5. Store the best solution and total number of the checked routes. |
| 1-6. If the user quit the program during the execution quit the program and find the best solution among the check routes. |
| *Child process* |
| 2-1. Find the best solution for the given prefix. |
| 2-2. If the program is ended or there is a SIGTERM signal give the best solution to the main |
| 2-3. Count the number of the execution. |
| *Communication* |
| 3-1. Deliver the best solution of child process to the main process |
| 3-2. Main process waits for deallocation |
| of child process and allocate new child process by signal handler. |

Table 1 : problem define

Brief approach for each problem.

| |
|---|
| a. 1-1 and 1-2 can be handled by managing "child_pid_used[]" array in main process. |
| b. 1-3 and 2-1 can be handled by give_next_Prefix() function. |
| c. 1-4 and 3-1 can be handled by Listen() function which listens to the pipe and get solution from child process. |
| d. 1-6, 2-2 and 3-2 can be handled by signal handler. |
| e. 2-3 and 1-5 can handled by program design |

Table 2 : problem approach

## 2. Approach

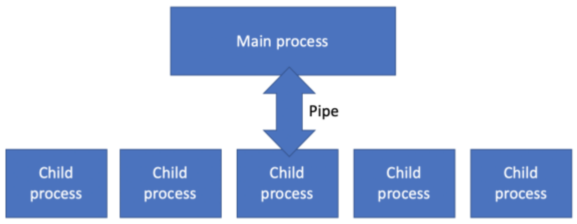
To explain more detail about the approach of the solution I will follow the Table 2.

a. To manage child processes "child_pid_used[]" array was used. The value of child_pid_used[i] is determined by the currently running state of the child_pid[i]. If the child_pid[i] is running child_pid_used[i] becomes 1 and if child_pid[i] terminate child_pid_used[i] becomes 0. This makes the main process to check how many child process is running and can allocate new child process to the empty child_pid[]. Main process checks the state of child_pid_used[] by "for loop" until there is no more subtask.

b. give_next_prefix() function gives next prefix by looking at the current prefix. It uses recursive function and generate prefix by decreasing value from right to left.

For example, if the current value is "9 5 3 2", because the right most value can be "1" it gives "9 5 3 1". And if the value is "9 5 3 0" it decreases the second right value because "0" cannot decrease any more. So. it gives "9 5 2 8" (If the maxim value of element is 9). It continues until the first value should be decreased to give the next prefix. So, the last value would be "9 0 1 2". I fixed the first value by giving the assumption that starting point should be start from the highest value. There needs two version of give_next_prefix(). Main process generate prefix from 0 to length-13 and child process length-12 to length -1.

c.



Pic 2 : Pipe connection

Listen() function is activated when child process terminates and SIG_CHLD occurs. Child process sends a string that contains best solution for the subtask through the pipe. Right after the child process terminates, Listen() function reads the pipe and decode the string. If the sum of all route is smaller than the previous solutions it changes the best solution for main process. Child processes and Main process shares one pipe so main process should clear the pipe before another child process writes on the pipe.

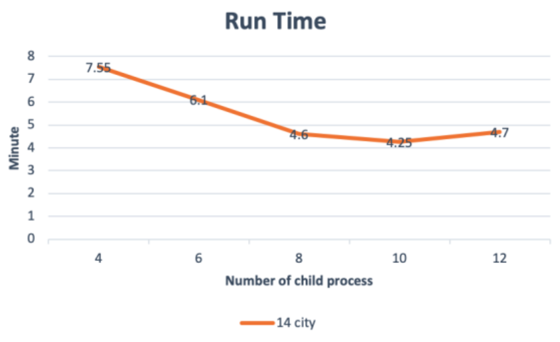
d. There are four signal handlers.

handler_ctr_c() and handler() executes when user pressed the crtl-c button. handler_ctr_c() kills all the remaining child processes. handler_ctr_c() uses sigchld_handler() and handler2() to find the best solution among the running child process before terminating. If all of the child process is terminated handler_ctr_c() terminates the main process. Handler() does nothing it waits until the child process terminates.

sigchld_handler() and handler2() executes when child process terminates. Sigchld_handler() executes in main process memory and handler2() executes in child process memory. handler2() gives the best solution of the subtask to the sigchld_handler() by using the pipe. Sigchld_handler() reads the pipe and give it to the main process.

## 3. Evaluation

a. Check Run Time



Graph 3 : Run time for 14 city.

The above graph 1 shows how does run time are affected by the number of child process. You can see that the first 3 element 4,6,8 shows a decreasing in linear and 8,10,12 are similar to each other. There are 14 subtasks to solve the 14 city tsp problem. So, if the child process number is 4 it takes 4 time rotation (14 % 4 = 4).

| Number of child process | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|
| Rotation number | 4 | 3 | 2 | 2 | 2 |
| Runtime rotation / number | 1.9 | 2.0 | 2.3 | 2.1 | 2.4 |

Table 3 : Compare "Runtime rotation / number" between the number of child process

If you look at the Table 3 we could find that the result of "runtime rotation / number" is almost around 2.0. This delivers that the subtask of each child process is well distributed.

b. Check correctness.

```
0 1 2 3 4 5 6 7 8 9 0 2 2 3
1 0 4 4 4 4 4 4 4 4 4 4 4 4
2 4 0 4 4 4 4 4 4 4 4 4 4 4
3 4 4 0 4 4 4 4 4 4 4 4 4 4
4 4 4 4 0 4 4 4 4 4 4 4 4 4
5 4 4 4 4 0 4 4 4 4 4 4 4 4
6 4 4 4 4 4 0 4 4 4 4 4 4 4
7 4 4 4 4 4 4 0 4 4 4 4 4 4
8 4 4 4 4 4 4 4 0 4 4 4 4 4
9 4 4 4 4 4 4 4 4 0 4 4 4 4
0 4 4 4 4 4 4 4 4 4 0 4 4 4
2 4 4 4 4 4 4 4 4 4 4 0 4 4
2 4 4 4 4 4 4 4 4 4 4 4 0 4
3 4 4 4 4 4 4 4 4 4 4 4 4 0
```

Pic 2 : check.tsp

By making a new input we can briefly check the correctness. If you look at the Pic 2 we can find that city 0,10 should be connected and 0,1 should be connected.

```
Path : 13 6 12 11 10 0 1 9 8 7 4 5 3 2
The length of shortest path is 49
```

Table 4 : Result of check.tsp

We can find the best solution for check.tsp consists "10 0 1" from Table 4.

Also, we should test if the results are well delivered even though the termination during running program.

```
Parent: 3306 16 15 14 13 12 11 10 9 1 4 2 5 7 6 0 3 8
Parent: 3404 16 15 14 13 11 12 10 9 1 4 2 5 7 6 0 3 8
Parent: 3242 16 15 14 13 10 12 11 8 4 1 9 2 5 7 6 0 3
```

Table 5 : Output from child process ended during exection.

```
Path : 16 15 14 13 10 12 11 8 4 1 9 2 5 7 6 0 3
The length of shortest path is 3242
```

Table 6 : Result of gr17.tsp

Table 5 shows the best solution for each child process among the checked route and Table 6 shows the best solution among the checked route. We can see that the best solution in Table 5 and Table 6 matches.

## 4. Discussion

During the programming ptsp.c the most difficult thing was checking the correctness of the program, because there was no answer of the input and it took a long time to run a program. To handle with the problems, I changed some part of the given material. First, I made 14*14 matrix to meet the end of running program. Also, I change the subtask of child process 12! into 8!.

One of way I used to reduce the execution time is fixing the starting point of the prefix. Every route should pass every city so fixing the starting point can reduce the overlap of computation.

Before knowing the signal of SIGTERM it made the program design much more complicated. The main process should check the child process in two different ways so the code length was twice larged. However, after using the signal SIGTERM I could reduce the length of code and don't have to care about asynchronization.

There are some problems of my program. First, I used a lot of global variable which makes the program unsafe. Second, multiple child process shares one pipe which can make missing information by the wrong asynchronization. Also, main process should continuously run even though it is just waiting for the end of child process.

I think first problem can solved by using a lot of parameter in the functions. I think second problem can be solved by allocating each pipe to the child process by using the pipe array. I still don't know the solution for the last problem. I think sleep function and interrupt can be used.

## 5. Conclusion

This homework teaches the way of using multiple child process. And gives method such as signal and pipe to communicate multiple child process with main process. By using the multiple process, we can decrease the runtime by dividing subtask and parallelizing the child process.