ITP 30002-02 Operating System, Spring 2020

Homework 2

name : EunJong Lee,   Student ID : 21700556,   Email address : eunjong55@naver.com

1. Introduction

The solve traveling salesmen problem(tsp) effectively, parallelization is needed, because to solve tsp, very huge amount of calculation is needed. This homework demand implement parallelization program to solve tsp. To implement parallelization, the job that making several processes and managing processes is needed. I solved it by dividing roles. Main process manage child process, and child process do subtask. To communicate between processes, I used pipe API.

Another problem is that when termination signal is given, terminate program and print best solution(best path, min value), total number of checked routes up to the point. To implement this function, something similar to interrupt handling is needed. I used signal API to solve this problem.

I think that how divide problem is very important, but requirement is given. Subtask explore only 12! cases and terminate. So I used requirement.

2. Approach

I divided the program into main process part and child process part. Simply, main process make prefix (make subtask) and delegate subtask to child process and update best solution by using child process's best solution. Child process do subtask and send best solution to parent process. I'll show you how each one works in detail.

In main process

Flow

When program is started, main process open grxx.tsp file and read path length data. The data is stored in map array. then main process make prefix to delegate subtask to child process. To find next prefix, I used permutation concept to consider all number of case. To implement permutation, there are two method. One is using for loop, another is using recursion. I selected recursion method, because the count of loop is not fixed(prefix length is variable). In base case part of recursion is executed when whole city count - prefix length == 12(child process conduct remain 12! Calculation), and at the moment next subtask is decided. In base case part, check new process can be made by checking process_count variable. If new process can be made, fork new process and delegate subtask. In this case, we don't need to send main process's data to child process, because fork function copy all main process's data. If new process cannot be made(process count is full), wait until one process is terminated. Then get process id of the process. Record that one more process can be executed, and reads the data of child process corresponding to the process id through pipe API. Update parent's data. Then make new process(fork) and delegate subtask. In recursive part, make permutation by using for loop and increase index.

When interrupt signal is detected(SIGINT)

To send interrupt signal, I used signal API, and SIGINT option.

In SIGINT handler, send signal SIGTERM to all child processes which is running, and wait a child process termination. When a child process is terminated, get terminated child process's id and get data of child process(count, best path, min length) from pipe, and update main process's data. Repeat this until all child process removed. then exit program.

The method to manage process count

I used an array using_process[] to store running process's id. If the array has free space, I make a new process by fork. In order to process management work well, it was necessary to remove terminated process from the array.

The method to communicate with child process

I used pipe API to communicate with child processes. I used an array pipes[][2] to communicate with several child processes. The index of the array tells which child process to communicate with. In main process only read child processes data through pipe.

The structure of message is that the message contain 3 data about count of checked routes, min length, best path, and all data is distinguished with ' '(blank space)

In child process

Flow

do remain calculation(12!) in child process to find best path and min length. It also use recursion to check all permutation. Base part of recursion executed when index reached the total city count. In base part of recursion update best solution if it is more good solution than previous solution. In recursive part of recursion, make permutation by using for loop and increase index.

When calculation is end, send data to main process by using pipe, and exit.

When interrupt signal is detected(SIGTERM)

To send termination signal I used signal API and SIGTERM option.

When SIGTERM is detected, in SIGTERM handler, send data about best path, min length, count that checked routes up to the point to parent process through pipe, and exit. In handler, child process don't know where to send. So I added global variable sendto to tell child process where to send data.

The method to communicate with main process

I used pipe API to communicate with main processes. I used an array pipes[][2] to communicate with main child processes. The index of the array tells which main process to communicate with. In child process only send child processes data through pipe.

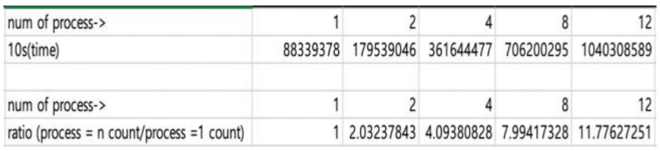
The structure of message is that the message contain 3 data about count of checked routes, min length, best path, and all data is distinguished with ' '(blank space)

3. Evaluation

1. If parallelization works well, the amount of data processed with in unit time should increase.

To check whether parallelization works well, I measured the number of checked route after 10 seconds(after 10s, press CTRL+C). To check time, I used smart phone timer.

Table1. Count measured after 10 seconds according to number of process.



num of process->	1	2	4	8	12
10s(time)	88339378	179539046	361644477	706200295	1040308589
num of process->	1	2	4	8	12
ratio (process = n count/process =1 count)	1	2.03237843	4.09380828	7.99417328	11.77627251

Of course there may be an error, but it is measured several times.

In this table, we can show that according to number of process, the number of count are increased. I evaluate parallelization works well.

Due to the time to manage the process(fork, send data and receive data through pipe), efficiency is expected to decrease slightly as the process increases.

2. To check whether processes always run as many children processes up to the given limit as possible at a time, I checked exist processes at several times.

I checked processes 30s, 1m, 1m 30s after the program starts(because in my program, I showed that one child process ends 30s after the program started) by using ps -fu s21700556. command.

I showed that always the count of process is same with limit, so I evaluate my program accomplished the given tasks.

4. Discussion

I thought that use less fork function will reduce total running time because fork task takes times. To reduce using fork function, efficient dividing of problem will be needed. For example, if process limit is 12, when we divide problem 12 parts and fork only 12 times, we can use minimal fork function. I think it will more fast than fork many times.

When is test SIGINT handling, I show all process react one SIGINT signal. At first, it is strange to me, but I understand that one signal can affect several process.

It is a little far from parallelization, During solve the tsp problem, I know that int, longlong type can not store a large number like 50!. To store large number like 50!, I implemented huge number adder by using string. adder logic is simple. for each element of string, convert character into integer and add two integer number and carry. If sum is bigger than 10 carry is 1. When use this adder, there is no limit to the length of numbers.

5. Conclusion

In this homework2, I implement parallelization to solve travel salesmen problem. I think key points of this homework is proper process creation and termination, intercommunication with child process and main process, how to send stop order and handle a process when it is ordered to stop it. To manage process creation and termination I used fork API. Fork API was very useful, because it copy all main process(parent process)'s data, so I don't need to send main process's data to child process. Also, I understand how to process terminated child process by using wait API. I can implement intercommunication between main process and child process by using pipe API. I send interrupt signal terminate signal to process by signal API and its handler. I was able to solve the problem by using these APIs, and while doing homework, I got a better understanding of how process works in multi process environment.