ITP 30002-02 Operating System, Spring 2020

Homework 2

Jeon, Yeo Hun / 21500630 / 21500630@handong.edu

1. Introduction

In this assignment, I built a multiprocessing program that solves a TSP (Traveling Salesman Problem). TSP is famous NP-complete problem, which means it needs to be checked all the possible solutions to get the best solution. To check the possible solutions, this program utilizes given numbers of its child processes to solve subtasks in the different region of the solution area.

User gives the program a TSP data recorded with distance weight from each city to others. With the data, user also gives a limit number of child process. The program's job is to distribute the problem to the number of its child processes, so they can check all the possible routes and get the optimal distance weight in the given subtask. To distribute the problem, the main process will make prefix of the subtasks so that child process always checks permutation of 12 cities with no duplicates. In addition, due to the NP-complete problem's computation time to get ultimate solution, it is necessary to get the best solution at the point when user wants to terminate the program.

In this program, DFS algorithm with backtracking strategy is used to make permutation of prefix and subtasks. Since the subtask always needs to be with 12 cities, the prefix length is set as *total number of cities-12.* To create only limited number of child process, the main process counts the number of child process and block its behavior by wait() when the number touches the limit. All the solutions made by child process needed to be checked by the main process to update the best solution. For this, the main process and its child process use unnamed pipe for reading and writing their solutions.

This program also has two signal handler functions. One is for SIGINT, which invokes when user put <control><z> to terminate process. When this signal invokes, the main processor waits for all child processes to be terminated. After it checks that all child processes are terminated, it prints the best solution it has and terminates itself. The other signal handler is for SIGCHLD, which invokes to parent process whenever any of its child process is terminated. Since the child process writes its optimal solution into pipe right before it is terminated, the main process starts read the pipe.

2. Approach

One of the jobs of the main process are to make 'prefix'. This prefix will make each subtask be unique and set the number cities needed to be traversed by child process to 12. It is created by making permutations of *total number of cities-12* number of cities with no duplicates. When the length of the permutation reaches to the number, the main process creates a child process and assign the prefix to the child process. The 'assignment' of prefix does not explicitly be done with function or statement. This is because when child process is forked, it inherits all the data and even program counter from its parent process. Therefore, the child process already has the prefix in its route-recording variable. After created, the child process starts to add cities into its route to make permutation with the remaining 12 cities. Making the permutation of both prefix and traversing routes is done by DFS algorithm with backtracking strategy. The visited cities are marked, and the process recursively add unmarked cities to the route of the prefix or subtask.

The main process and the child process work concurrently. The fork function returns the pid of the created child process to main process and 0 to the child process. Therefore, we can distinguish jobs of each type of process with the pid. When the child process starts its job, the main process backtracks the permutation and add new city to make new prefix. If the prefix successfully made, the main process makes new child process and assigned the prefix. Otherwise, if there is no more possible prefix, the main process calls wait() to wait for all child processes finish their work and terminate. The waiting continues infinitely until the wait function return -1, which means error, no more child process to wait.

When the number of created child process reaches to the limit number given by user, the main process stops for space to make new child process even though it has a prefix. To manage this behavior, the main process has a counter variable to track the number of child process, and call wait() to wait for any terminated child process. Whenever one of the child processes is terminated, the main process resumes its work, create new child process with prefix it held before wait behavior.

Since the all child process should be forked from single main process, the child process has exactly same data of main process at the point fork. However, to update the optimal solution to main process, it is necessary to make communication system between main and child process. Pipes are used for this situation. When program begins and main process starts, it allocates integer array with size 2. This array will be a pipe by using pipe function in unstd.h. By this function, each element of the array is initialized as read and write file descriptor. The read file descriptor is used for the main process and the write file descriptor is used by child process. When child process is created, it inherits all data from its parent process. With the data, the file descriptors of the pipes are also inherited. Therefore, all the child process can write into pipe which is connected to the main process. The communication is made by two functions that writes the optimal data into write pipe and read the data from pipe and update the best solution.

Whenever the child process checks all the possible routes and there is no more possible permutation, it calls function to write the optimal solution of the subtask into the pipe. Then it terminates itself by exit(). When child process terminates, signal invokes in main process name SIGCHLD. The signal is sent from child process to its parent process. When the main process gets this signal, it will call its own predefined function. This can be done by using signal() function to designate a function as signal handler. In this program, the signal handler for SIGCHLD is designed to call read data from pipe. After reading the data in main process, it updates the value of number routes checked, the min distance value and the routes of the min distance and reducing the number of running child process.

To allow user to stop program and get best solution include optimal solution from currently working child process, this program has another signal handler. When user input <control><z> during the process running, SIGINT signal invokes for all main process and child process. Therefore, the signal handler function defined in the program distinguishes either the process is main or child by checking pid that saved when fork occurs. If the process is main process, it needs to wait for all child process is terminated. The strategy of waiting is same as that of waiting after making all possible prefix in main process. If the process is child process, the process should not be terminated and go back to its work to finish the assigned subtask.

3. Evaluation

To evaluate the program, I made questions below.

1. Does the program find solution successfully?
2. Does the program utilize child processes as many as possible?
3. Does the program keep running until all the child processes is terminated when user try to terminate the program?
4. Does the program pass data through pipes successfully?

To check if the program finds expected solution, I set a test data with 13 cities, the minimum size of input of this program. Since the subtask is always with 12 cities, the main process needs to make 13 prefixes. I gave program 3 as the limit number child process. As the result, it found the best solution I expected and the total number of checked routes as output was 6,227,020,800, which means the program conducted 12! tasks for 13 times.

The number of running process is not given by the program output. Therefore, I used 'watch "ps"' command to monitor currently running process. In this experiment, I used input data with 17 cities and 5 limit number of child process. As the result, I could check that 6 processes are running. This means that one main process and 5 child processes are working. I also could observe that the pid of child processes are keeps changing since when one child process terminates, the main process created new child process.

With same input condition of above, I input <control><z> few seconds after the execution. The program did not terminate immediately. I could observe that 6 processes are still running in ps command. After few minutes later, I could get an output with solution, and 2,395,008,000 checked routes, which means the program checked for 5 different prefix and each subtask for the prefix has 12! possible routes.

In the program, there is no any availability to bring data from child process to main process without using pipes. The initial variable to store the optimal path and the minimum distance is initialized with 0 and -1. Therefore, the pipe communication worked well between the main process and the child process. In addition, the best path as output of the program is different while the minimum distance value is same when I terminate the program with SIGINT signal. Each child process takes different time to finishes its work. In Addition, the program does not update value if newly read solution from child is equal or less than the current best value. Therefore, if there are child processes with same optimal distance value, the first one who writes data into pipe and terminates will be the answer.

4. Discussion

In this program, multiprocessing strategy is used to make parallel computation to solve given problem. Each process should have a single thread. Therefore, this program's performance is highly depending on the number of cores that can be used to execute the process at the same time. The processes concurrently execute by context switching in the given number cores.

The issue in multiprocessing oriented program is that it requires extra memory space for the child process. When a child process is created by fork(), it needs a memory space to inherit exactly same data in the main process. More on to this, it is relatively more complicated to make communication between parent and child processes. The pipe does not allow process to write and read with single pipe. It allows only a single direction communication. If it is required to have bi-direction communication among parent and child processes, it is not easy to design and build the system.

Using multithreading strategy might be much better to increase performance and implement aspect. In multithreading, the true parallel computation can be conducted even in a single core system. Moreover, the context switching between threads is more cost efficient than that between processes because the threads have shared memory within the process. Since thread share the data, it will be also much easier to manage the best solution of the problem.

The best strategy to use will be using mixed strategy of multiprocessing and multithreading. The main process manages the subtask and create child, and the child creates multiple threads to compute the best solution in its subtask. In this way, the main process can have advantage of multiprocessing to manage all the child processes, and the child process can have fast computation with multithreading in its own subtask.

In addition, since the problem condition of this program requires visiting all the given cities and the last route is always needed to be returning from the last city to the first city, the solution forms a complete graph. In complete graph, the starting point of does not need to be considered because the best solution will be same in any starting point. Therefore, it could make better performance when the program makes prefix from the second city. For instance, the best solution of input data with 13 cities can be found with checking *12! * 1* combination instead of checking all the *12! * 13* combination.

5. Conclusion

In this assignment, I built parallel TSP solver program. This program needs to have distinct behavior between the main process and child processes created by the main process. The main process creates a prefix to set unique subtask and make the size of the subtask to 12 cities. With the prefix, the child processes test all the possible routes and find best solution within the subtask. The obtained optimal solution of the subtask is transferred to the main process by pipes, and the main process updates the global optimal solution by checking the data passed from the child processes.

To make all possible combination of prefix and routes, DFS with backtracking strategy is used. The visited cities are marked as visited, and the combination is made to keep adding possible cities until the length there is no more possible combination.

The main process manages the number of child process by counting the number when fork() occurs. If the number of child process reaches to the limit, the main process wait infinitely until any child process is terminated. To getting optimal solution from child process, the main process catches SIGCHLD signal and immediately read the pipe to get data passed from the child process. If the optimal solution from the child is better solution than it currently has, the main process updates the global solution.

Since NP-complete problem takes enormous computation time to get final solution, it is needed to get the best solution at the point when user wants to stop the program. To implement this, the program will catch SIGINT signal and make two different behavior for main and the child processes. The main process will be wait for all child processes are terminated and the child processes ignore the termination signal and resume the work to find best solution of the given subtask.

I set some questions to evaluate the functionalities of the program and conducted experiments in different execution scenario. As the result, I could get the expected output of number of checked routes and final solution with smallest data. I also could observe the management of the child processes by main process with 'ps' command in system.