South China University of Technology

《Operating System》Experiment Report

Experiment Title： Session 1: Process and Thread Creation

Name： 谭演锋 Student ID： 202130100456

Class： 计算机全英联合班 Group：

Collaborator：

Teacher： 钟竞辉

|  |
| --- |
| **Description** |
| 【Objective and Requirement】  **Objective:** Be familiar with the creation of process and thread.  **Requirement:**  **Task 1:** Create a console application, “child”, which keeps printing out “The child is talking at [system time]” (in a loop, one per 1s).  **Task 2:** Create another console application, “parent”. It creates a child **process** to execute “child”. At the same time, the “parent” process keeps printing out “The parent is talking at [system time]”. (one per 1s). Execute “parent” and explain the output you see.  **Task 3: C**reate a child thread in the “mainThread” program. Both the main thread and the child thread keep printing out “[ThreadID] + [System time]”.  **Task 4**: Create a console application, which contains a shared integer shared\_var. The initial value of shared\_var is 0. The application will create a child **thread** after it starts. The main thread keeps increasing the value of shared\_var by 1, while the child thread keeps decreasing the value of shared\_var by 1. Explain the observed results.  【Environment】  Operating System：Linux CentOS7 |
| **Content** |
| 【Procedure】  **Task 1**  First we include the <chrono> library provides a set of templates and functions to perform time-related operations with a high degree of precision. The <thread> library provides functionality for managing threads, such as sleeping, creating, and synchronizing threads. The <ctime> library is used to convert time values to human-readable strings.  In the code I first use infinite loop to keep the console keep printing. Then in order to get current time, we use the std::chrono::system\_clock to obtain the current time point and stores it in the variable now. The auto keyword is used to automatically deduce the type of the variable. Then, we use the to\_time\_t function to convert the time point (now) to a time\_t type, which is a representation of time used in C. We print the “the child is talking at [system time]”. The **std::ctime** function is used to convert the time\_t value to a C-style string representing the time, and it is then printed to the console.  Finally, in order to print the result for each second, we use the code **std::this\_thread::sleep\_for(std::chrono::seconds(1))**; std::this\_thread::sleep\_for: It is used to make the current thread sleep or pause for a specified duration. std::chrono::seconds(1): Creating a duration of 1 second. The std::chrono::seconds is a duration type representing seconds. The 1 inside the parentheses is the duration value, indicating 1 second delay.    Fig. 1 Detail codes of task1  In order to run on Linux, we first compile the code by the command “g++ -o child child.cpp” to call the compiler “g++” to generate an executable file named "child".  Then use the command “./child” to run the file.  截屏2023-12-13 14.46.53  Fig. 2 Running result of task 1  **Task 2**  In order to finish the task, first we include necessary C++ standard libraries (<iostream>, <chrono>, <ctime>, <thread>) and POSIX libraries (<unistd.h> for fork() and execv()).  In the **main** function, the program first checks if **fork()** is equal to 0. The fork() function is used to create a new process by duplicating the existing process. After a successful fork(), two processes are created: the parent process and the child process. The child process is an almost exact copy of the parent, and both processes continue to execute from the point of the fork() call. In the parent process, fork() returns the process ID (PID) of the child process. In the child process, fork() returns 0.  If it's the child process, it executes a new program using execv() with parameter "/root/os/lab1/child" to call the child process The char \*v[] array is used to pass arguments to the new program, and in this case, it is set to NULL (no arguments). The exec() family of functions is used to replace the current process image with a new one. There are several variants of the exec() function, such as execl, execv, execle, execve, etc. They differ in how command-line arguments are passed to the new program and in how environment variables are handled. If exec() is successful, it does not return; the new program is executed, and the old process image is replaced. If exec() fails, it returns -1.  If it's the parent process, it enters an infinite loop, printing the current time every second, similar to the child process.    Fig. 3 Detail codes of task2  Following is the output, we can see that the parent process and the child process alternately output their respective results on the console, child and parent process print the same time. It is a correct answer, since the time slots for each process is much smaller then 1 second. Each process sleeps for 1 second after printing which will lead to another process be called and print the result immediately.    Fig. 2 Detail codes of task3  **Task 3**  First we include the necessary libraries. In the code, I first create a function called “print\_message” which is similar codes in the task1. The only difference is we get the thread id by using the function **std::this\_thread::get\_id()** to print the result.  In the **main** function, I first call the function **std::thread child\_thread(print\_message)**; to create a child thread and associates it with the **print\_messag**e function. The constructor of **std::thread** takes a function (in this case, print\_message) as an argument. It means that the new thread will execute the code defined in the print\_message function. The child thread starts executing concurrently with the main thread.  Then call the function print\_message(), Since the main thread also calls the print\_message function, so both the main thread and the child thread are executing this function concurrently.  Finally the **child\_thread.join()**; This line ensures that the main thread waits for the child thread to finish before proceeding. The join function is used to synchronize the threads. The main thread will pause at this line and wait for the child\_thread to finish its execution before moving on. This ensures that the main thread does not proceed beyond this point until the child\_thread has completed its task. The main thread will pause at this line and wait for the child\_thread to finish its execution before moving on. This ensures that the main thread does not proceed beyond this point until the child\_thread has completed its task    Fig. 5 Running result and source code of task 3  Finally, we can get the output, which is the two threads “548460490752” and “ 548456296928” alternately output their respective results on the console, which is similar to the task2.    Fig. 6 Running result and source code of task 3  **Task 4**  Here we first include the library <mutex> for the use of a mutex (**std::mutex**) to synchronize access to a shared variable (shared\_var) between the main thread and a child thread. Then we will use the fucntion The **std::lock\_guard** will be employed to automatically lock and unlock the mutex in a scoped manner.  std::lock\_guard is a C++ standard library class template defined in the <mutex> header. The primary purpose of std::lock\_guard is to ensure that a mutex is locked during the execution of a critical section and automatically releases the lock when the std::lock\_guard goes out of scope.  The **main** function structure is similar to the task3 instead of the mutex lock. So we do not talk about it any more.    Fig.7 Source code of task 4  If the sleep duration of main thread is equal to the child thread, the shared variable doesn’t have increasing or decreasing trend, it just oscillates back and forth between 0 and 1. Here is the example of main thread sleep for 0.5 second and the child thread sleep for 0.5 second.    Fig.8 Running result of task 4  Here is the example of main thread sleep for 0.1 second and the child thread sleep for 0.5 second. If the sleep duration of parent thread is less than the child thread, the shared variable has increasing trend. Also we can see for a round, the main thread called 5 times and the child called 1 time.    Fig.9 Running result of task 4 (II)  Here is the example of main thread sleep for 0.5 second and the child thread sleep for 0.1 second. If the sleep duration of parent thread is larger than the child thread, the shared variable has decreasing trend. Also we can see for a round, the main thread called 1 times and the child called 5 time.    Fig.10 Running result of task 4 (III) |
| **Conclusion** |
| In the experiment, I met difficulty in get the current time first. Then int task2, I also met diffculty in calling other process in the codes. In the task3 and task4 I met the difficulty in creating thread.  From the task1, I learn how to get the current time by the library <chrono> and <ctime> including using the function chrono::system\_clock::now to get time and translate time by the function chrono::system\_clock::to\_time\_t() and ctime() for a readable string. Then I learn the usage of library <thread> to block the process. Then from the task, I learn the using of POSIX libraries (<unistd.h> for fork() and execv()). I learn how to use fork() function to create a new child process and separating them from child and parent by return value. I also learn how tocall the executable file image by the function execv() by adding the path as parameter. In the task3, I learn how to create a thread by using the std::thread and use a function as the parameter to execute the thread. And I learn the function this\_thread::get\_id() to get the current thread ID. Finally, I learn the usage of function thread.join() for the thread synchronize. Finally, in the task4 I learn the use of library <mutex> and the function std::lock\_guard<std::mutex> lock(mutex) to protect the mutex accessing of a code block scope. And I learn influence the different sleeping time for a thread and its shared\_variable. In conclusion, I learn a lot coding knowledge of the thread and the process from the 4 task in the lab, which give me a deeply understanding of process and thread in practical. This process meaningfully enhances and impresses my understanding of the principle and management of process and thread in the computer operating system. |
| **Teacher’s Comments and Score** |
| Comment：  Score：           Signature：                                                 Date： |