South China University of Technology

《Operating System》Experiment Report

Experiment Title： Session 1: Process and Thread Creation

Name： 向天翼 Student ID： 201836020389

Class： 18计算机联合班 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：Ubuntu 18.04.4 LTS |
| **Content** |
| 【Procedure】  **Task 1**  Here a function ‘get\_time’ is defined to return current time in string format. And we use an infinite loop to let the program keeps printing out “The child is talking at [system time]”. ‘sleep’ function is also used to control the frequent of outputting. The code is showed in Appendix and the result is showed in Fig.1.    Fig. 1 Running result of task 1  **Task 2**  To create child process, we use ‘fork’ function and check its return value to distinguish child process and parent process. If the process is child process, we use ‘execv’ function to run the child program we have compiled in task 1, otherwise the parent process will output ‘The parent is talking at’ and current time. From the running results which is showed in Fig.2, we can see that the parent process and the child process alternately output their respective results on the console.      Fig. 2 Running result and source code of task 2  **Task 3**  In this task we use ‘pthread\_create’ function to create new thread in a process. If creating is succeed, we continue running the process, otherwise the process terminates. Here, the function ‘child\_thread’ become a parameter of the ‘pthread\_create’ function, because we set it to be the entrance of the child thread. Also, ‘pthread\_self’ function is used here to get the thread id. The child thread will execute the codes in ‘child\_thread’ function and the parent thread will execute the remaining code in main function. From the result showed in Fig.3 we can see that two threads execute their own code, and alternately output in the console.      Fig. 3 Running result and source code of task 3  **Task 4**  Here we need to create two thread and let them to operate a single variable. We use the same method in Task 3 to create thread and let them execute different functions. And we try to adjust different value of sleep function and to see the changing trend of the shared variable.    Fig.4 Source code of task 4  If the sleep duration of parent 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.    Fig.5 Running result of task 4 (I)  If the sleep duration of parent thread is less than the child thread, the shared variable has increasing trend.    Fig.6 Running result of task 4 (II)  If the sleep duration of parent thread is larger than the child thread, the shared variable has decreasing trend.    Fig.7 Running result of task 4 (III)  **Appendix: Source code for every task.**  //Task 1 Child.cpp  #include<stdio.h>  #include<stdlib.h>  #include<time.h>  #include<unistd.h>  char\* get\_time(){  time\_t tt = time(NULL);  tm\* t=localtime(&tt);  char \*buffer = (char\*)calloc(80,sizeof(char));  strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", t);  return buffer;  }  int main(){  while(true){  printf("The child is talking at %s.\n",get\_time());  sleep(1);  }  }  //Task 2 Parent.cpp  #include<stdio.h>  #include<time.h>  #include<unistd.h>  #include<stdlib.h>  char\* get\_time(){  time\_t tt = time(NULL);  tm\* t=localtime(&tt);  char \*buffer = (char\*)calloc(80,sizeof(char));  strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", t);  return buffer;  }  int main(){  if(fork() == 0){  char \*v[] = { (char \*) 0 };  execv("./child.out",v);  }  else{  while(true){  printf("The parent is talking at %s.\n",get\_time());  sleep(1);  }  }    }  //Task 3 mainThread.cpp  #include<stdio.h>  #include<time.h>  #include<unistd.h>  #include<pthread.h>  #include<stdlib.h>  pthread\_t thread1;  char\* get\_time(){  time\_t tt = time(NULL);  tm\* t=localtime(&tt);  char \*buffer = (char\*)calloc(80,sizeof(char));  strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", t);  return buffer;  }  void \*child\_thread(void\* arg){  while(true){  printf("%lld %s\n",pthread\_self(),get\_time());  sleep(1);  }  }  int main(){  int i = pthread\_create(&thread1,NULL,child\_thread,NULL);  if(i == 0){  while(true){  printf("%lld %s\n",pthread\_self(),get\_time());  sleep(1);  }  }  }  //Task 4 sharedData.cpp  #include<stdio.h>  #include<time.h>  #include<unistd.h>  #include<pthread.h>  pthread\_t thread1;  int shared\_var = 0;  void \*child\_thread(void\* arg){  while(true){  printf("%d ",--shared\_var);  fflush(stdout);  sleep(2);  }  }  int main(){  int i = pthread\_create(&thread1,NULL,child\_thread,NULL);  if(i == 0){  while(true){  printf("%d ",++shared\_var);  fflush(stdout);  sleep(1);  }    }  } |
| **Conclusion** |
| From this lab session I ‘ve learned the how to create process and thread, and do some simple operation on them in Linux operating system. This process meaningfully enhances 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： |