

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2019-2020** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 冯春霖 |
| **Student ID:** | 222019321062074 |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Process synchronization and deadlock | | | |
| Date | | May, 2021 | Type | | √ Confirmatory  √ Design  √ Comprehensive |
| 1. **Objective & Requirements**    1. Understand the concept of process synchronization; understand the concept of deadlock    2. Learn how to use the synchronization mechanisms provided in Linux and pthread library, i.e. the mutex lock    3. Can use mutex lock to solve real synchronization problems    4. Understand the implementation of mutex lock using inline assembly of C programming language | | | | | |
| 1. **Experimental environment (**platform and software**)**   VirtualBox+Ubuntu | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results) 2. Task 1   Implement a function called test\_and\_set() which can test and set the pass value to it **atomically**. Hint: using the atomic instruction on intel x86-64 architecture   * **lock cmpxchg *m64, r64***   The template and requirement of test\_and\_set() is as follows:    Please complete the missing inline assembly code inside asm();   1. Task 2   Based on your implemented test\_and\_set() function in Task 1, implement your own mutex lock mechanism   * Your lock variable is defined as a long integer with initial value 0   **long int lock = 0;**   * Your unlock() and lock() function takes the pointer to lock as the argument   **void my\_lock(long int \*lock);**  **void my\_unlock(long int \*lock);**   * Please use your implemented lock() and unlock() functions to solve the producer-consumer critical section problem.  1. Please provide your procedure to perform the tasks and source codes.   Task1:  The inline assembly code as follow:      asm(          "mov rax, 0;"          "mov rbx, 1;"          "lock cmpchg %1, rbx;"          "mov %0, rax;"          :          :"m"(res), "m"(\*lock)      );  Use cmpxchg to compare lock and rax, if lock==rax==0, lock = rbx, res stores the value of rax, return 0. If lock==1, rax = lock, res==rax==1. return 1. The function has been tested to return the correct value in all cases.  Task2:  Combined with the test\_n\_set code written in Task1, the code to get and unlock locks is shown below  void my\_Lock(long \*lock)  {      while (test\_n\_set(lock));  }  void my\_Unlock(long \*lock)  {      \*lock = 0;  }  Compile and run programs with inline assembly language  After about five minutes of running, we can see that the value of count is still 5, which proves the viability of the lock | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   Through this experiment, I got a better understanding of process synchronization and learned to use some mechanisms such as mutex lock to implement process synchronization and solve practical problems. At the same time, I also tried to use inline assembly language to write C programs. I had not studied assembly language before and it was difficult for me to understand it at first. I got a good grasp of assembly language through self-study and successfully completed this experiment. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |