

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Operating System Principle |
| **Semester**: | 2nd semester of the academic year **2019-2020** |
| **Major**: | Software Engineering |
| **Class**: | 2019 |
| **Student Name**: | 吴嘉诚 |
| **Student ID:** | 222019321062111 |
| **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**)**   Ubuntu20.04 | | | | | |
| 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.   **T1:**   1. Codes   #include <stdio.h>  long int test\_n\_set(long int \*lock)  {  long int res;  **asm(**  **"mov rax,0;"**  **"mov rbx,1;"**  **"lock cmpxchg [%1],rbx;"**  **"mov %0,rax;"**  **:**  **:"m"(res),"r"(lock)**  **:"eax","ebx"**  **);**  }  int main(int argc, char const \*argv[]) //测试  {  long int lock0 = 0;  long int lock1 = 1;  printf("origin lock0 is 0, now lock is:%ld, return value is: %ld\n", lock0,test\_n\_set(&lock0));  printf("origin lock1 is 1, now lock is:%ld, return value is: %ld\n", lock1,test\_n\_set(&lock1));  return 0;  }   1. 实验测试      1. 代码分析  * 对于内联汇编代码，我首先将rax置为0（用于比较），rbx置为1（用于lock为0时，将lock置为1） * 执行cmpxchg后，若lock为0，因为与rax相同，则lock被置为rbx的值1，且由于rax未被改写，则0被写入到待返回的值res处；若lock为1，因为与rax不同，则rax被置为lock的值1，并写入到待返回的值res处。   **T2:**   1. Codes   #include <stdio.h>  #ifndef \_\_USE\_GNU  #define \_\_USE\_GNU  #endif  #include <unistd.h>  #include <sched.h>  #include <pthread.h>  **long int test\_n\_set(long int \*lock)**  **{**  **long int res;**  **asm(**  **"mov rax,0;"**  **"mov rbx,1;"**  **"lock cmpxchg [%1],rbx;"**  **"mov %0,rax;"**  **:**  **:"m"(res),"r"(lock)**  **:"eax","ebx"**  **);**  **}**  **void lock(long int \*l)**  **{**  **while(test\_n\_set(l));**  **}**  **void unlock(long int \*l)**  **{**  **\*l= 0;**  **}**  **long int mutex = 0; //delcare mutex**  int count;  void \*producer(void \*param); /\* threads call this function \*/  void \*consumer(void \*param); /\* threads call this function \*/  int main(int argc, char \*argv[])  {  pthread\_t tid1, tid2; /\* the thread identifier \*/  pthread\_attr\_t attr; /\* set of thread attributes \*/  pthread\_attr\_init(&attr); /\* get the default attributes \*/  while(1)  {  count = 5; //init  pthread\_create(&tid1, &attr, producer, NULL); /\* create the thread \*/  pthread\_create(&tid2, &attr, consumer, NULL); /\* create the thread \*/  pthread\_join(tid1, NULL);  pthread\_join(tid2, NULL);  printf("count is: %d\n", count);  if(count == 4 || count == 6)  {  printf("press enter to continue\n");  getchar();  }  }//end-of-while  }  void \*producer(void \*param)  {  cpu\_set\_t cpuSet;  CPU\_ZERO(&cpuSet); //clear  CPU\_SET(0, &cpuSet); //set core 0  sched\_setaffinity(0, sizeof(cpuSet), &cpuSet);  lock(&mutex); //lock  count++; //produce one element  unlock(&mutex); //unlock  pthread\_exit(0);  }  void \*consumer(void \*param)  {  cpu\_set\_t cpuSet;  CPU\_ZERO(&cpuSet); //clear  CPU\_SET(1, &cpuSet); //set core 1  sched\_setaffinity(0, sizeof(cpuSet), &cpuSet);  while (count == 0);  lock(&mutex); //lock  count--; //consume one element  unlock(&mutex); //unlock  pthread\_exit(0);  }   1. 代码分析  * 对于lock()，我循环调用test\_n\_set()来判断当前mutex锁的值，若为0，则意味着未加锁，则将mutex锁的值置为1，且跳出循环；若为1，则不改变mutex的值，且一直循环调用test\_n\_set() * 对于unlock()，只需将锁的值置为0即可表示释放锁。 * 我使用了老师上课时的mutex实例代码，来检验我的lock(),unlock()是否能在原子级别上检验并改写mutex的值。若是，则代码运行效果将一直打印5，不会打印出4或6并暂停。  1. 实验测试   执行11m后，仍未出现4或6：  可认为结果符合预期  2021-05-22 14-19-43 的屏幕截图 | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   实验总结   * 在有汇编学习的基础上，成功写出了test\_n\_set()函数的汇编实现部分，在构思的过程中，在cmpxchg指令的逻辑上纠结了很久，在列出所有情况后，验证了自己写的函数的正确性。   实验收获   * 学习到了内联汇编在c语言中的用法，尤其学习到了内联汇编中冒号后optional的部分，如”m”(a)以及”r”(b)的用法，以及clobbered register部分”eax”等。 * 学习到了原子指令lock cmpxchg的用法。Lock用于给cpu指定这个原子指令，cmpxchg用于原子级地比较并交换值。 * 加深了对原子性和锁的理解。Count++在汇编中非原子，导致竞争。 | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |