SUSTech CS302 OS Lab7 Report

Student Name: Shijie Chen ID: 11612028

Time: 2019.04.14

Experiment Environment: Linux C++

Fundamentals

- What is deadlock?

  Deadlock is a circular waiting for resources.

- What are the requirements of deadlock?
  - Mutual exclusion
    - only one thread at a time can use a resource
  - Hold and wait
    - Thread holding at least one resource is waiting to acquire additional resources held by other threads
  - No preemption
    - Resources are released only voluntarily by the thread holding the resources.
  - Circular wait
    - There exists circular waiting for resources.
- What's the difference between deadlock prevention and deadlock avoidance?
  - Deadlock prevention is preventing the above four conditions from happening at the same time. Deadlock is impossible if prevention is done.
  - Deadlock avoidance is a resource allocation strategy (such as banker's algorithm) that OS can adopt to prevent circular waiting for resources.
- How to prevent deadlock? Give at lease two examples.

  We can prevent deadlocks by breaking at least one of the above four conditions.

  Possible actions include:

  - Allocate all required resources to the threads at the beginning.
  - Force the threads to access resources in a particular order so that circular waiting will not happen.
- Which way is chosen by recent UNIX OS to deal with deadlock problem? And why?
  - Ignore the problem and pretend there's no deadlock.
  - Reason:
    - Deadlock situations rarely happen.
    - Detection and resolving of deadlocks is very time and resource consuming.

Banker's Algorithm

- What data structures do you use in your implementation? Where and why do you use them? Are they optimal for your purpose?
    - vector<int*>

      Several vectors containing arrays of integers are used in my implementation to store information about resource allocation and requirement. Each thread will have a corresponding array in each such data structure.
    - map<int, int>

      A map is used to map `pid` of a thread to the corresponding index of their array in each vector. Also, this makes it possible to reuse `pid`s. Suppose a thread is created and then terminated, its `pid` can be reused later without changing the structures (insertion and deletion of arrays) of the vectors to improve efficiency.

Conclusion

In this lab, I learned the condition and avoidance of deadlocks. Also I implemented the banker's algorithm in C++