# Report for lab2

## The thread control methods provided by Linux

Linux provided 7 thread control method.They are: 1.pthread_self() to get current thread's pid. 2.pthread_create() to create a new thread executing a function. 3.pthread_exit() to exit the current thread, sometime a thread has done its job, but the process who holds it care nothing about it, it calls this function to exit. 4.pthread_join() to make the current thread blocked until that thread finished. 5.pthread_detach() to make that thread free, running by itself. 6.pthread_cancel() to kill that thread, usually used when a main thread managing other threads. 7.pthread_equal() to check if two pid is the same, usually used like pthread_equal(pid,pthread_self());

## About My program

The lab2 has two part, both need Multithreaded programming.My design is as following:

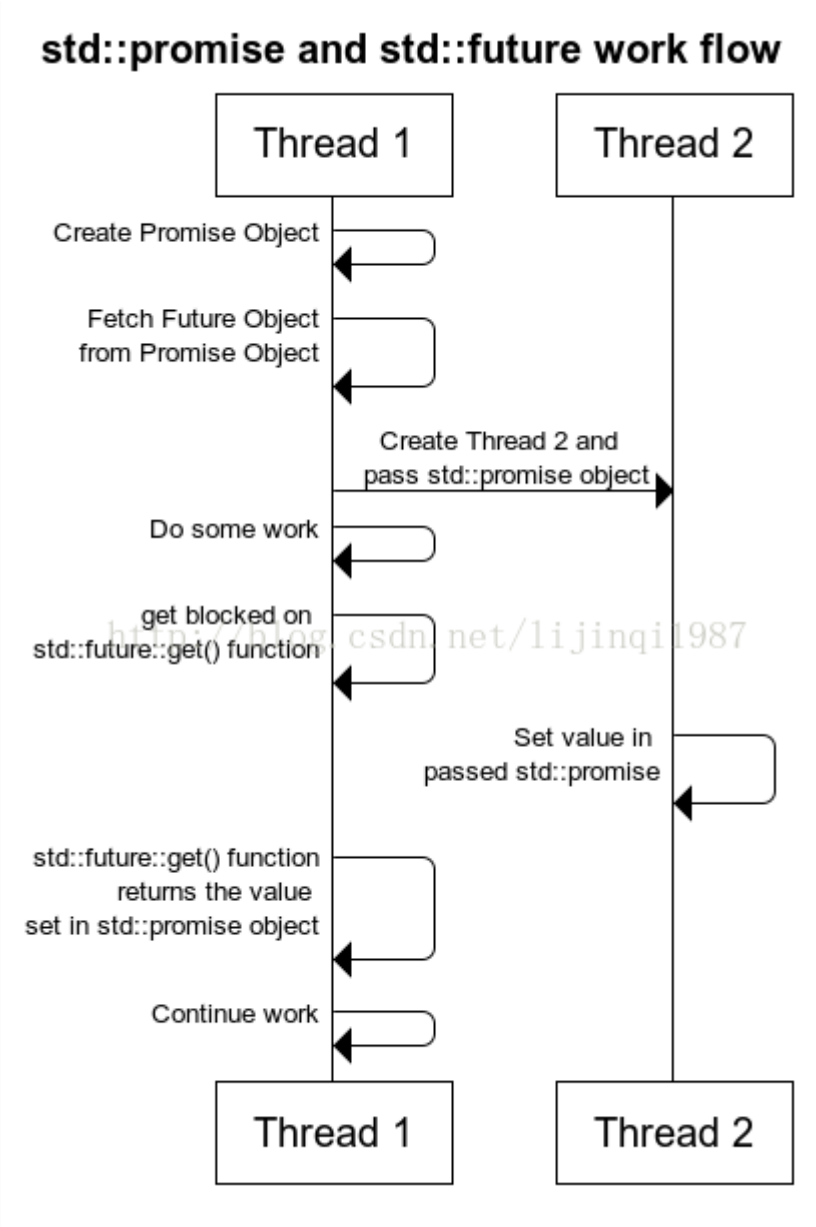
## Part 1: sudoko solution

The solution is very simple , thus there is no need to share date between threads.The data transmit is one-way,when creating sub thread, the data flow main thread->sub thread, when a sub thread having finished , the data flow sub thread->main thread.I'd use some features in C++11 instead of pthread, bc I'm more familiar with C++.

The code to validate the suduko is really simple, so I'd like to use some time to introduce the most important feature used.

### std::promise & std::future

In the past decades, it's impossible to get a return value from a thread(POSIX thread and Linux's thread). But C++11 had introduced a feature to change the situation.U could define a promise variable, get its future Object, set the future's value in a thread, and get the value in the main thread later(get() will be blocked if the future's value is not set already).




This feature is very useful when we want to know result from a validate function, we could use a single get(), without explicit calling wait(), or using any lock, or use a looping to check the whether variable has been set.

```
//std::promised & std::future
void validateALine(Line data, std::promise<int>* promObj){
    //....some code
    promObj->set_value(ture);
}

int main(){
    //....some code
    std::promise<bool> promiseObj;
    std::future<bool> futureObj = promiseObj.get_future();
    std::thread th(validateALine, &data, &promiseObj);
    std::cout<<futureObj.get()<<std::endl;
    th.join();
```

```
return 0;

}
```

## Something intersting or not very easy

I've met something intersting while debuging my code, I'd like to share them in case some other may met the same problem.

The first problem is, a Sudoko is sizeof 9*9, it's more reasonable to read a sudoko from a file instead of from stdin.But how could we convert the char stream to a list/array of int?There is atoi() to convert a C-style string to a int, but there is never a function to convert a char stream to a list/array of int.So I write codes as following, use a regex to search all subString be made up of digit only, then convert them to int.

```
    regex expression("(\\d)+");
    regex_iterator<string::iterator> it(contents.begin(), contents.end(), expression);
    regex_iterator<string::iterator> end;


    vector<int> temp;
    for (; it != end; ++it)
    {
        temp.push_back(atoi(it->str().c_str()));
    }
```

The Second problem is, I tried to create some thread by method likes std::thread(std::function, args) in a class, but if the function is a member function, you have to use std::bind , writing code likes this to create a std::function.

```
thread t1(bind(&Class::yourFunction, Class*, args)
```

But, if one of your args is a instance of template promise<*R>, bind() will failed bc there is no copy constructor of template promise*<R>.That means you have to put the arg promise<_R> in ypur class as a member.

The Third problem I met is, since I was using using std::thread, I believe the IDE will automatically bind a thread library, but the compiling failed and throw an error say "undifined calling pthread_create".I think this maybe owing to g++ implement std::thread by calling pthread instead of implement a independent library.So I add "-pthread" in the link option, that works.

In addition, the pdf require us to pass argument to thread, but in my design, the argument is stored in a Class, there's no need to pass it by pthread_create().But In the part two, I use some method to prove I do know how to pass argument to thread.

# Part 2: Multi-thread sort

The second part didnt accquire the which kind of sorting function to use, but we need to pass argument about index to the sorting thread, the std::sort meets our demand.I established thread and let them run as following.

```
vector<int> unsorted;
vector<int> sorted;

void part_sort_function(<vector<int>>::iterator Begin, iterator<vector<int>>::iterator
End)
{
    //sort the element between Begin and End
    //write result into vector unsorted
}
void merge_sort_function()
{
    sorted = vector<int>(unsorted.size());
    thread sort_1(sort_function, unsorted.begin(), unsorted.begin()+unsorted.size()/2);
    thread sort_2(sort_function, unsorted.begin()+unsorted.size()/2+1, unsorted.end());
    sort_1.join();
    sort_2.join();
    //the merging thread will be blocked untill sort_1 & sort_2 exited
    //merge vecotr_1 and vector_2
    //write result into vector sorted
}


int main()
{
unsorted = vector<int>({...});
thread merge(merge_sort_function);
merge.join();
}
```

A snapshot of part 2:


# Conclution

In addition, the whole project is released on github, you could get it from https://github.com/devoteasecond/OS_course_lab.git