OS course Design Reoprt

1. Library Choice :Javafx

I decide not to use `swing`, because it is too old for GUI application. So `javafx` seems to be the only choice.

How to achieve Concurrency

2.1 Task Class

using `import javafx.concurrent.Task`

Task is similar with `Runnable`, which can be used to init a Thread. It is specially designed for javafx GUI, because In javafx, all updates of UI elements must be applied on main javafx thread, for thread safety's sake. but in some cases, like this course design, We must update UI elements in an other thread. For example, in this course design. each instruction of a process should be observed clearly by the user. To achieve this, we made the process sleep for a while before executes next instruction, If process runs on the main javafx thread, the GUI will also be stopped, until no more sleep on any preocess. So we must made process running on other thread. Task can make it possible with thread safety.

2.2 Platform.RunLater

In some cases, directly change variables on fx thread from another thread is necessary `Platform.RunLater` poses a synchronization between two threads, to ensure thread safety, you can use put your code in `Platform.RunLater`. For more information,

2.3 Property binding

I taskes advantage of java bean's property class and bind method. property is a wrapper for a vairiable , you can bind two properties, if any one of the two's value is changed, the another will change either. Task has some builtin properties to bind, and UI elements also have builtin properties waiting for binding. you can bind them, change a property on the Task's Thread, but made UI element's property , which is safely on javafx thread, changes too.

3. Data Structure And Class Diagram

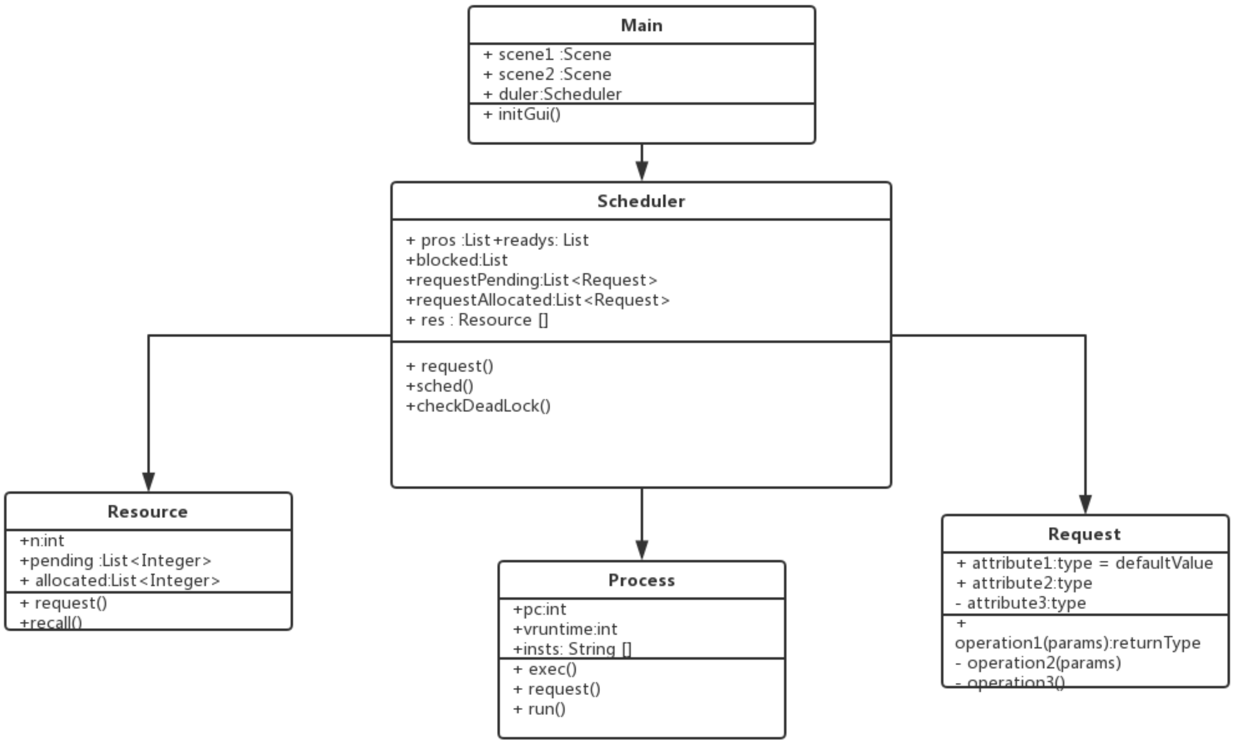
3.1 Frontend

screenshot absent[todo!] 还没做好
We have two scenes. **scene1** and **scene2**.
scene1 is for user to observing Process scheduling. User can easily observing process's state switching. and current process's inst list.
scene2 is for observing resource management. You can watch current resource's number, see each reuquests and their dealing state.

3.2 Backend

TO BE DONE All schelduling and requesting work are done background by scheduler. It loads process insts ....

3.3 Total Class Diagram



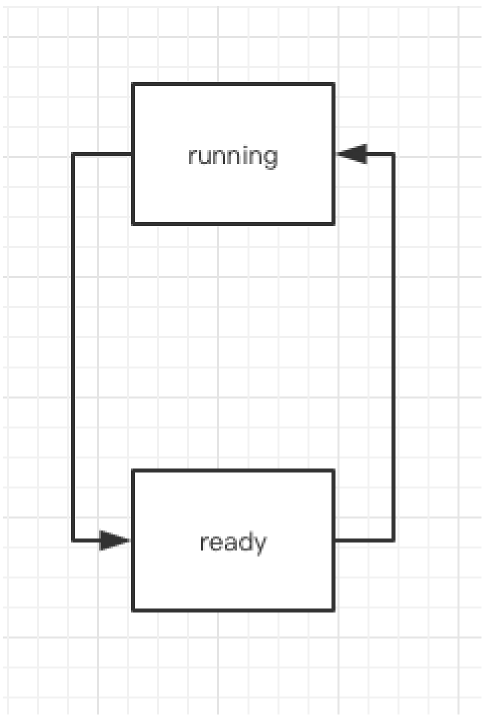
4. Module Functions

4.1 Process Instruction Execution

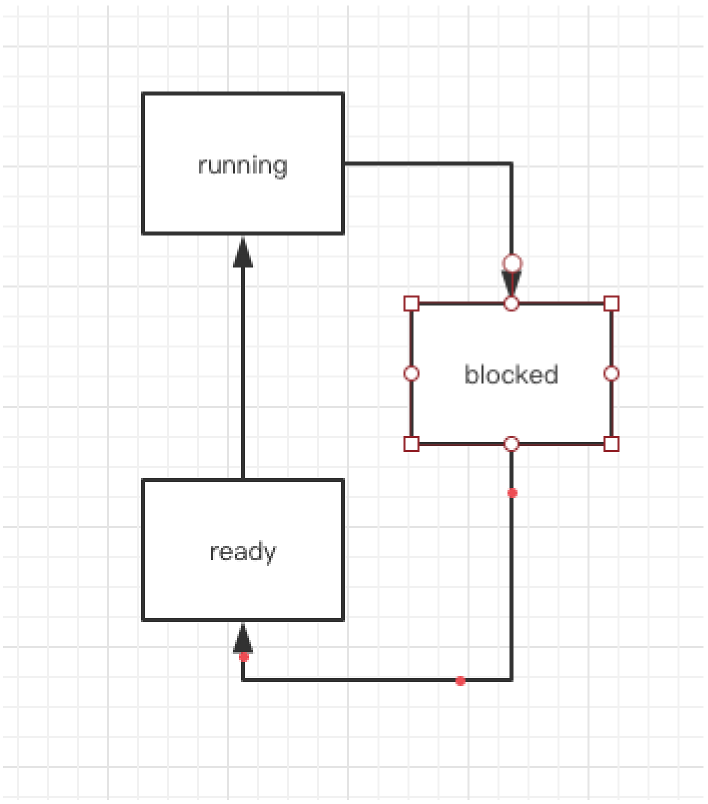
Each Process has a list of instructions, executed by exec(), one by one, all resource request commands are issued from there. Here we can see how Instruction been executed in Process.

4.2 Process Schedule And Three States Switch

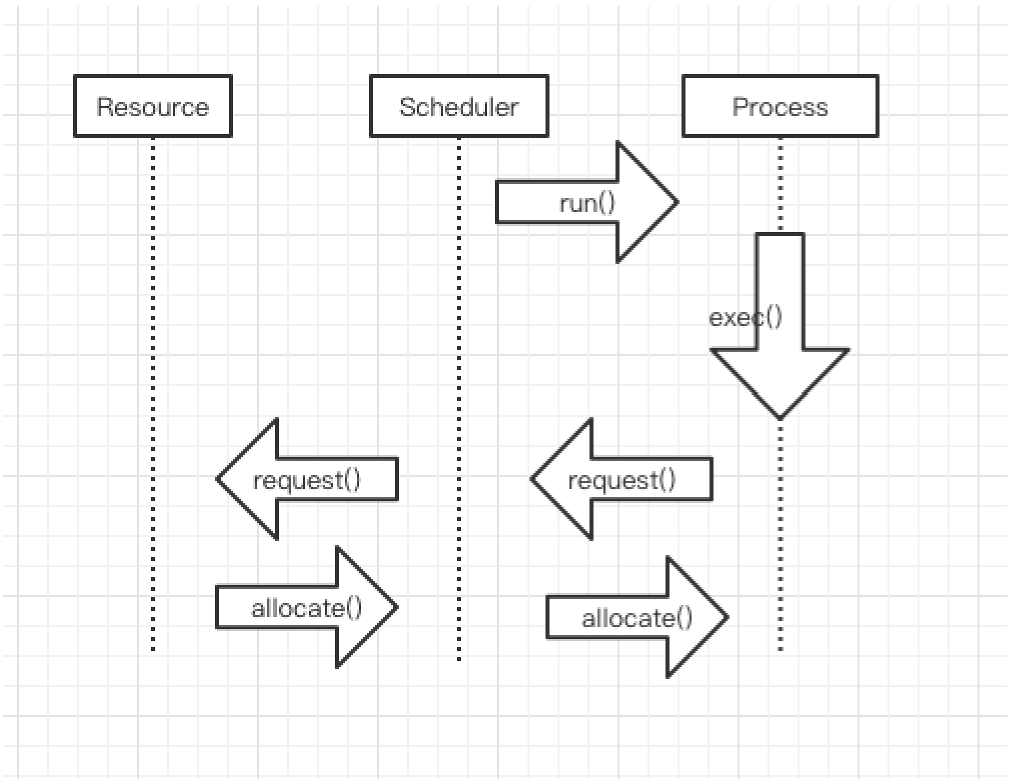
We have to deal two situations. First, running and ready switch. We use round time robin algorithm, without consideration of resource request. We sort **readys** by **Process.vruntime** to decide which one is a running candidate.



Second one has a connection with Resource Request. When a process invoke request(), it must be blocked, until resource is allocated properly. Then it can be put in **readys** , waiting to be choosed as running.



4.3 Process Resource Request



4.4 DeadLock Check And Relaese

We use following condition to check whether there is a deadlock. If there are two resource, **res1** and **res2**. And two process pids **pida** and **pidb** meet following conditions:

1. pida in res1.pending
2. pidb in res1.allocated
3. pida in res2.allocated
4. pidb in res2.pending
5. pida and pidb can not be satisfied either.

Then we find a deadlock.

Here is part of source code implement.

```
ArrayList<Integer> pida = (ArrayList<Integer>)res1.pending.clone();

ArrayList<Integer> pidb = (ArrayList<Integer>)res2.pending.clone();

pida.retainAll(res2.allocated);

pidb.retainAll(res1.allocated);
```

Reference

1. JavaFX并发

2. Updating the UI using Platform.runLater