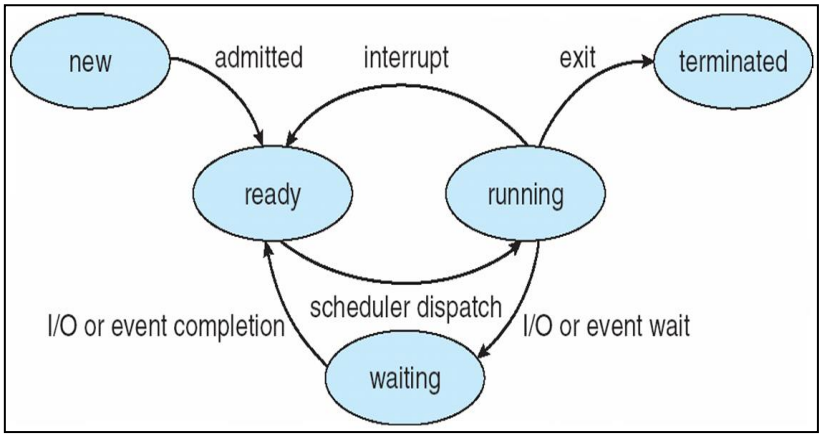
1102 Operating System
Programming Project - Nachos
Topic: Scheduling

Part 1. trace code

For the given diagram of process state, please describe how nachos manages the lifecycle of a thread.

(Note: You need to explain the purposes and details of the 6 code paths.)



I. New→Ready
   1) userprog/userkernel.cc UserProgKernel::InitializeAllThreads()
   2) userprog/userkernel.cc UserProgKernel:: InitializeOneThread(char*~~, int, int~~)
   3) threads/thread.cc Thread::Fork(VoidFunctionPtr, void*)
   4) threads/thread.cc Thread::StackAllocate(VoidFunctionPtr, void*)
   5) threads/scheduler.cc Scheduler::ReadyToRun(Thread*)

II. Ready→Running
   1) threads/scheduler.cc Scheduler::FindNextToRun()
   2) threads/scheduler.cc Scheduler::Run(Thread*, bool)
   3) threads/switch.s SWITCH(Thread*, Thread*)
   4) machine/mipssim.cc Machine::Run()

III. Running→Ready
   1) machine/mipssim.cc Machine::Run()
   2) machine/interrupt.cc Interrupt::OneTick()
   3) threads/thread.cc Thread::Yield()
   4) threads/scheduler.cc Scheduler::FindNextToRun()
   5) threads/scheduler.cc Scheduler::ReadyToRun(Thread*)
   6) threads/scheduler.cc Scheduler::Run(Thread*, bool)

IV. Running→Waiting
    1) userprog/exception.cc ExceptionHandler(ExceptionType) case SC_PrintInt
    2) userprog/synchconsole.cc SynchConsoleOutput::PutInt()
    3) machine/console.cc ConsoleOutput::PutChar(char)
    4) threads/synch.cc Semaphore::P()
    5) threads/synchlist.cc SynchList<T>::Append(T)
    6) threads/thread.cc Thread::Sleep(bool)
    7) threads/scheduler.cc Scheduler::FindNextToRun()
    8) threads/scheduler.cc Scheduler::Run(Thread*, bool)

V. Waiting→Ready
    1) threads/synch.cc Semaphore::V()
    2) threads/scheduler.cc Scheduler::ReadyToRun(Thread*)

※Note: When a thread has a console output(I/O), it needs to yield CPU resource and go to waiting state. After finishing console output(I/O), this thread can return to ready queue.

VI. Running→Terminated (Note: start from the Exit system call is called)
    1) userprog/exception.cc ExceptionHandler(ExceptionType) case SC_Exit
    2) threads/thread.cc Thread::Finish()
    3) threads/thread.cc Thread::Sleep(bool)
    4) threads/scheduler.cc Scheduler::FindNextToRun()
    5) threads/scheduler.cc Scheduler::Run(Thread*, bool)

Part 2. Implementation

I. Implement **Preemptive** Shortest Job First
    1) If current thread has the same burst time, the one with greater thread id should be executed first.
    2) Take the Approximate Burst Time formula shown below as consideration:

$$t_i = 0.5 * T + 0.5 * t_{i-1},\ i > 0,\ t_0 = 0$$

(T: CPU burst time of current thread, t_{i-1}: predicted CPU burst time of last thread)

II. Add a debugging flag **j** in your code and use the DEBUG('j', expression) macro (which is defined in debug.h) to print following messages. Remember to replace {...} to the corresponding value.
    1) Before a thread is inserted into a queue:
    ***Thread [{**thread x ID**}]'s and thread [{**thread y ID**}]'s burst time are

[{**thread x burst time**}] and [{**thread y burst time**}]***

2) When a thread is inserted into a queue:
   <I> Tick [{**current total tick**}]: Thread [{**thread ID**}] is inserted into readyQueue
3) When a thread is removed from a queue:
   <R> Tick [{**current total tick**}]: Thread [{**thread ID**}] is removed from readyQueue
4) When a thread updates its approximate burst time:
   <U> Tick [{**current total tick**}]: Thread [{**thread ID**}] update approximate burst time, from: [{$\mathbf{t_{i-1}}$}] + [{**T**}], to [{$\mathbf{t_i}$}]
5) When a context switch occurs:
   If preemption happens, the flag will be <YS>. Otherwise, it will be <S>.
   <YS/S> Tick [{**current total tick**}]: Thread [{**new thread ID**}] is now selected for execution, thread [{**prev thread ID**}] is replaced, and it has executed [{**accumulated ticks**}] ticks

- Hint: (you MUST follow the following rules in your implementation)

~~1. The operations of preemption can be delayed until the timer alarm is triggered (the next 100 ticks timer interval).~~

1. You should only modify the file which include “TODO” in the folder.
2. Refer to time clock in machine/stats, stats is also a member of kernel.

- Please comment out this line in threads/alarm.cc file.

```
//<TODO>
// In each 100 ticks, Update RunTime
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();

    kernel->currentThread->setRunTime(kernel->currentThread->getRunTime() + TimerTicks);

    if (status == IdleMode) {    // is it time to quit?
        if (!interrupt->AnyFutureInterrupts()) {
            timer->Disable();   // turn off the timer
        }
    }
    else {          // there's someone to preempt
        //interrupt->YieldOnReturn();
    }
}
//<TODO>
```

- Instruction
  1. Switch to the code folder
     cd nachos-4.0-final/code
  2. Compile
     make clean
     make

3. Test your implementation with test file

   userprog/nachos -e <execute file> -e <execute file> -d j

   ex:

   userprog/nachos -e test/sjf_test1 -e test/sjf_test2 -d j

   You should see the results as below:

```
shiuan@shiuan-VirtualBox:~/1102_nachos/nachos-4.0-final-ans/code$ userprog/nachos -e test/sjf_test1 -e test/sjf_test2 -d j
[I] Tick [10]: Thread [1] is inserted into readyQueue
***Thread[1]'s and thread[2]'s burst time are [0] and [0]***
[I] Tick [20]: Thread [2] is inserted into readyQueue
[R] Tick [30]: Thread [2] is removed from readyQueue
[S] Tick [30]: Thread [2] is now selected for execution, thread[0] is replaced, and it has executed [0] ticks
Switching from: 0 to: 2
ForkExecute => fork thread id: 2, currentTick: 40
[AddrSpace::Load over] Tick [40]: Thread [2]
[AddrSpace::Execute over] Tick [40]: Thread [2]
2[R] Tick [69]: Thread [1] is removed from readyQueue
[S] Tick [69]: Thread [1] is now selected for execution, thread[2] is replaced, and it has executed [0] ticks
[U] Tick [69]: Thread [2] update approximate burst time, from : [0], add [0], to [0]
Switching from: 2 to: 1
[I] Tick [79]: Thread [2] is inserted into readyQueue
ForkExecute => fork thread id: 1, currentTick: 79
[AddrSpace::Load over] Tick [79]: Thread [1]
[AddrSpace::Execute over] Tick [79]: Thread [1]
[R] Tick [98]: Thread [2] is removed from readyQueue
[S] Tick [98]: Thread [2] is now selected for execution, thread[1] is replaced, and it has executed [0] ticks
[U] Tick [98]: Thread [1] update approximate burst time, from : [0], add [0], to [0]
Switching from: 1 to: 2
[I] Tick [108]: Thread [1] is inserted into readyQueue


2[R] Tick [643]: Thread [1] is removed from readyQueue
[S] Tick [643]: Thread [1] is now selected for execution, thread[2] is replaced, and it has executed [600] ticks
[U] Tick [643]: Thread [2] update approximate burst time, from : [0], add [300], to [300]
Switching from: 2 to: 1
[I] Tick [644]: Thread [2] is inserted into readyQueue
[R] Tick [644]: Thread [2] is removed from readyQueue
[S] Tick [644]: Thread [2] is now selected for execution, thread[1] is replaced, and it has executed [0] ticks
[U] Tick [644]: Thread [1] update approximate burst time, from : [0], add [0], to [0]
Switching from: 1 to: 2
[I] Tick [654]: Thread [1] is inserted into readyQueue
[R] Tick [664]: Thread [1] is removed from readyQueue
[YS] Tick [664]: Thread [1] is now selected for execution, thread[2] is replaced, and it has executed [0] ticks
Switching from: 2 to: 1
1[I] Tick [675]: Thread [1] is inserted into readyQueue
[R] Tick [675]: Thread [1] is removed from readyQueue
[S] Tick [675]: Thread [1] is now selected for execution, thread[1] is replaced, and it has executed [0] ticks
[U] Tick [675]: Thread [1] update approximate burst time, from : [0], add [0], to [0]
Switching from: 1 to: 1
```

   This is just part of the output.

- Report
  1. Including your results of code tracing, how you implement your scheduling, and your team member contribution.
  2. File name: Final_Report_<group number>.pdf

- Demo
  1. We will check your code and the results of your implementation on the spot.
  2. Thus, prepare your own devices and make sure it works in advance.
  3. Some questions about this homework will be asked, too.
  4. Demo time will be announced later.

- Grading
  1. Report ---30%
  2. Demo ---70%
     - ✧ Implementation --- 40%
     - ✧ Question --- 30%

- Please upload your report to **eeclass** before **2022/05/31(Tue.) 23:59**.
- Late submission is not allowed.
- Discussion is encouraged, but plagiarism will be punished strictly.
- Feel free to discuss with TAs, and it's encouraged to discuss on **eeclass** forum. (Please mail TA to make an reservation on TA time).