Operating System
HW2: CPU Scheduling

Part 1. Trace code.

Explain the purposes and details of the following 6 code paths to understand how nachos manages the lifecycle of a process (or thread) as described in the Diagram of Process State.

1-1. New→Ready

UserProgKernel::ExecAll()
↓
UserProgKernel::Exec(char*, int)
↓
Thread::Fork(VoidFunctionPtr, void*)
↓
Thread::StackAllocate(VoidFunctionPtr, void*)
↓
Scheduler::ReadyToRun(Thread*)

1-2. Running→Ready

Machine::Run()
↓
Interrupt::OneTick()
↓
Thread::Yield()
↓
Scheduler::FindNextToRun()
↓
Scheduler::ReadyToRun(Thread*)
↓
Scheduler::Run(Thread*, bool)

1-3. Running→Waiting (Note: only need to consider console output as an example)

SynchConsoleOutput::PutChar(char)

↓

Semaphore::P()

↓

SynchList<T>::Append(T)

↓

Thread::Sleep(bool)

↓

Scheduler::FindNextToRun()

↓

Scheduler::Run(Thread*, bool)

1-4. Waiting→Ready (Note: only need to consider console output as an example)

Semaphore::V()

↓

Scheduler::ReadyToRun(Thread*)

1-5. Running→Terminated (Note: start from the Exit system call is called)

ExceptionHandler(ExceptionType) case SC_Exit

↓

Thread::Finish()

↓

Thread::Sleep(bool)

↓

Scheduler::FindNextToRun()

↓

Scheduler::Run(Thread*, bool)

1-6. Ready→Running

Scheduler::FindNextToRun()

↓

Scheduler::Run(Thread*, bool)

↓

SWITCH(Thread*, Thread*)

↓

(depends on the previous process state, e.g.,[New, Running, Waiting]→Ready)

↓

for loop in Machine::Run()

Note: **switch.S** contains the instructions to perform context switch. You must understand and describe the purpose of these instructions in your report. (Try to understand the x86 instructions first. Concept in MIPS version is equivalent to x86 in switch.s, and learn more about x86 in following link: https://www.cs.virginia.edu/~evans/cs216/guides/x86.html)

Part 2. Implementation

2.1 Implement a **multilevel feedback queue scheduler** with aging mechanism as described below:

(a) There are 3 levels of queues: L1, L2 and L3. L1 is the highest level queue, and L3 is the lowest level queue.

(b) All processes must have a valid **scheduling priority between 0 to 149**. Higher value means higher priority. So 149 is the highest priority, and 0 is the lowest priority.

(c) A process with priority between **0 - 49** is in **L3** queue, priority between **50 - 99** is in **L2** queue, and priority between **100 - 149** is in **L1** queue.

(d) **L1** queue uses **non-preemptive SJF (shortest job first)** scheduling algorithm. If current thread has the lowest approximate burst time, it should not be preempted by the threads in the ready queue. The burst time (job execution time) is approximated using the equation:

$$t_i = 0.5 * T + 0.5 * t_{i-1} \text{ (type double)}, i > 0 , \ t_0 = 0$$

Update burst time when state becomes waiting state, stop updating when state becomes ready state, and resume accumulating when state moves back to running state.

(e) **L2** queue uses a **preemptive priority** scheduling algorithm. A thread in L2 queue will preempt other threads in L2 queue; however, it will preempt thread in L3 queue.

(f) **L3** queue uses a **round-robin** scheduling algorithm with time quantum 100 ticks (you should select a thread to run once 100 ticks elapsed). If two threads enter the L3 queue with the same priority, either one of them can execute first.

(g) An **aging mechanism** must be implemented, so that the priority of a process is **increased by 10** after waiting for more than **900 ticks** (Note: The operations of preemption and priority updating can be delayed until the **next timer alarm** interval).

2.2 Add a command line argument **-ep** for nachos to initialize priority of process. E.g., the command below will launch 2 processes: hw2_test1 with initial priority 40, and hw2_test2 with initial priority 80.

```
$ userprog/nachos -ep test/hw2_test1 40 -ep test/hw2_test2 80
```

2.3 Add a debugging flag z and use the DEBUG('z', expr) macro (defined in debug.h) to print following messages. Replace {...} to the corresponding value.

(a) Whenever a process is inserted into a queue:
[A] Tick [{**current total tick**}]: Thread [{**thread ID**}] is inserted into queue L[{**queue level**}]

(b) Whenever a process is removed from a queue:
[B] Tick [{**current total tick**}]: Thread [{**thread ID**}] is removed from queue L[{**queue level**}]

(c) Whenever a process changes its scheduling priority:
[C] Tick [{**current total tick**}]: Thread [{**thread ID**}] changes its priority from [{**old value**}] to [{**new value**}]

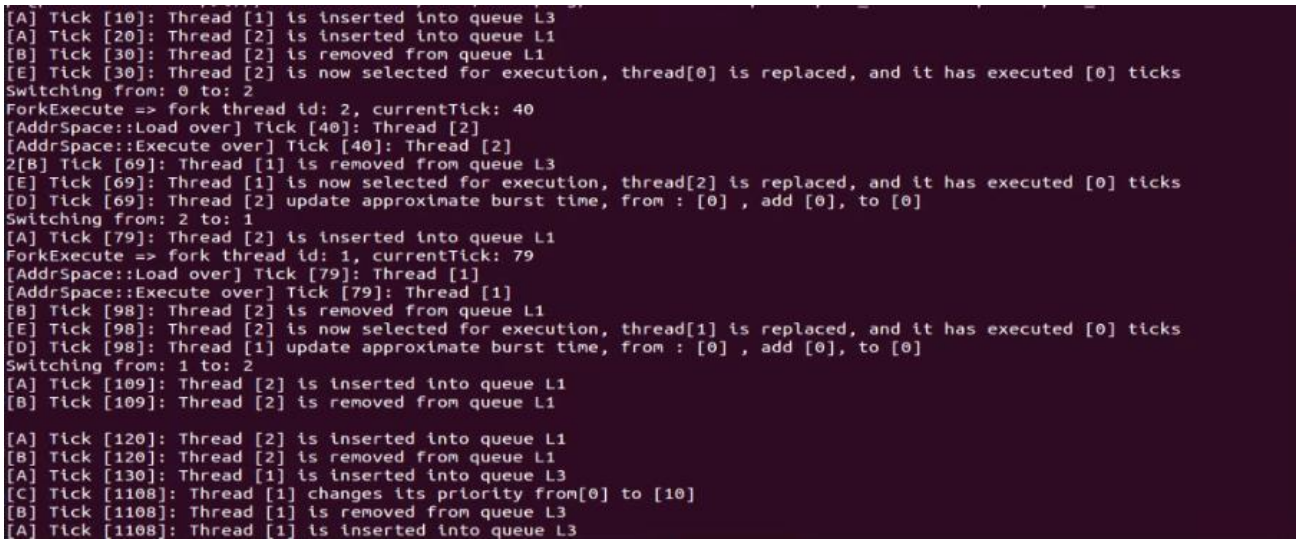
(d) Whenever a process updates its approximate burst time:
[D] Tick [{**current total tick**}]: Thread [{**thread ID**}] update approximate burst time, from: [{**ti-1**}], add [{**T**}], to [{**ti**}]

(e) Whenever a context switch occurs:
[E] 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

- Rules: (you MUST follow the following rules in your implementation)
  (a) Do not modify any code under the machine folder.
  (b) Do NOT call the Interrupt::Schedule() function from your implemented code. (It simulates the hardware interrupt produced by hardware only.)
  (c) Only update approximate burst time ti(include both user and kernel mode) when the process changes its state from running state to waiting state. In case of running to ready (interrupted), its CPU burst time T must keep accumulating after it resumes running.
  (d) The operations of preemption and rescheduling events of aging can be delayed until the timer alarm is triggered (the next 100 ticks timer interval).

Hint1: You should only modify the file which include "TODO" in the folder.
Hint2: Refer to the annotation which include "TRACE" in the folder to understand how hw2 run.

- Instruction
  1. Switch to the code folder
     cd nachos-4.0-hw2
  2. Compile
     make clean
     make
  3. Test your implementation with different p1 and p2
     userprog/nachos -ep <execute file> <p1> -ep <execute file> <p2> -d z
     ex:
     userprog/nachos -ep test/hw2_test1 0 -ep test/hw2_test2 80 -d z
     You should see the results as below:



other test case:

userprog/nachos -ep test/hw2_test1 0 -ep test/hw2_test2 0 -d z

userprog/nachos -ep test/hw2_test1 50 -ep test/hw2_test2 80 -d z

userprog/nachos -ep test/hw2_test1 50 -ep test/hw2_test2 100 -d z

userprog/nachos -ep test/hw2_test1 100 -ep test/hw2_test2 100 -d z

- Report

  Including your results of code tracing, how you implement your system calls, and your team member contribution.

  File name: HW2_Report_<group number>.pdf

- Demo

  We will check your code and the results of your implementation on the spot.

  Thus, prepare your own devices and make sure it works in advance.

  Some questions about this homework will be asked, too.

  Demo time will be announced later.

- **Grading**

  Report ---30%

  Demo ---70%

  Implementation correctness --- 50%

  Question --- 20%

- Please upload your report to iLMS before 2021/05/17 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 iLMS forum