Operating System
Final Project: 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

- userprog/userkernel.cc UserProgKernel::InitializeAllThreads()
- userprog/userkernel.cc UserProgKernel:: InitializeOneThread(char*, int, int)
- threads/thread.cc Thread::Fork(VoidFunctionPtr, void*)
- threads/thread.cc Thread::StackAllocate(VoidFunctionPtr, void*)
- threads/scheduler.cc Scheduler::ReadyToRun(Thread*)

1-2. Running→Ready

- machine/mipssim.cc Machine::Run()
- machine/interrupt.cc Interrupt::OneTick()
- threads/thread.cc Thread::Yield()
- threads/scheduler.cc Scheduler::FindNextToRun()
- threads/scheduler.cc Scheduler::ReadyToRun(Thread*)
- threads/scheduler.cc Scheduler::Run(Thread*, bool)

1-3. Running→Waiting (Hint: When a thread has a console output(I/O), it needs to yield CPU resource and go to waiting state.)

- userprog/exception.cc ExceptionHandler(ExceptionType) case SC_PrintInt
- userprog/synchconsole.cc SynchConsoleOutput::PutInt()
- machine/console.cc ConsoleOutput::PutChar(char)
- threads/synch.cc Semaphore::P()
- threads/synchlist.cc SynchList<T>::Append(T)
- threads/thread.cc Thread::Sleep(bool)
- threads/scheduler.cc Scheduler::FindNextToRun()
- threads/scheduler.cc Scheduler::Run(Thread*, bool)

1-4. Waiting→Ready (Hint: After finishing console output(I/O), this thread can return to ready queue.)

- threads/synch.cc Semaphore::V()
- threads/scheduler.cc Scheduler::ReadyToRun(Thread*)

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

- userprog/exception.cc ExceptionHandler(ExceptionType) case SC_Exit
- threads/thread.cc Thread::Finish()
- threads/thread.cc Thread::Sleep(bool)
- threads/scheduler.cc Scheduler::FindNextToRun()
- threads/scheduler.cc Scheduler::Run(Thread*, bool)

1-6. Ready→Running

- threads/scheduler.cc Scheduler::FindNextToRun()
- threads/scheduler.cc Scheduler::Run(Thread*, bool)
- threads/switch.s SWITCH(Thread*, Thread*)

- Machine/mipssim.cc 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 **preemptive SRTN (shortest remaining time first)** scheduling algorithm. If current thread has the lowest remaining burst time, it should not be preempted by the threads in the ready queue. The burst time (job execution time) is provided by user when execute the test case.

(e) **L2** queue uses a **FCFS (First-Come First-Served)** scheduling algorithm which means lower thread ID has higher priority.

(f) **L3** queue uses a **round-robin** scheduling algorithm with time quantum 200 ticks (you should select a thread to run once 200 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 **400 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 **-epb** for nachos to initialize priority of process. E.g., the command below will launch 2 processes: hw2_test1 with initial priority 40 and burst time 5000, and hw2_test2 with initial priority 80 and burst time 4000.

```
$ userprog/nachos -epb test/hw2_test1 40 5000 -epb test/hw2_test2 80 4000
```

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:
[InsertToQueue] Tick [{**current total tick**}]: Thread [{**thread ID**}] is inserted into queue L[{**queue level**}]

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

(c) Whenever a process changes its scheduling priority:
[UpdatePriority] 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:
[UpdateRemainingBurstTime] Tick [{**current total tick**}]: Thread [{**thread ID**}] update remaining burst time, from: [{**ti-1**}] - [{**T**}], to [{**ti**}]

(e) Whenever a context switch occurs:
[ContextSwitch] 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)
  (a) The operations of preemption and rescheduling events of aging can be delayed until the timer alarm is triggered (the next 100 ticks timer interval).
  (b) You should only modify the file which include “TODO” in the folder
  (c) Refer to the annotation which include "REPORT" in the folder to understand how final project runs and explain it in the report.
  (d) Refer to time clock in machine/stats, stats is also a member of kernel.

- Instruction
  1. Clone the NachOS source code and cross compiler on github (Refer to Environment & NachOS Installation file)
     git clone https://github.com/joshhsieh1999/2024_OS_Final.git
  2. Decompress file
     tar -zxf <Compressed-File>
  3. Switch to the code folder
     cd nachos-4.0-final
  4. Compile
     make clean
     make
  5. Test your implementation with different p1, bt1, p2, bt2
     userprog/nachos -epb <execute file> <p1> <bt1> -epb <execute file> <p2> <bt2> -d z
     ex:
     userprog/nachos -epb test/hw2_test1 40 5000 -epb test/hw2_test2 80 4000 -d z
     You should see the results as below:

```
[InsertToQueue] Tick [10]: Thread [1] is inserted into queue L3
[InsertToQueue] Tick [20]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [30]: Thread [2] is removed from queue L2
[ContextSwitch] 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[RemoveFromQueue] Tick [69]: Thread [1] is removed from queue L3
[ContextSwitch] Tick [69]: Thread [1] is now selected for execution, thread[2] is replaced, and it has executed [0] ticks
Switching from: 2 to: 1
[InsertToQueue] Tick [79]: Thread [2] is inserted into queue L2
ForkExecute => fork thread id: 1, currentTick: 79
[AddrSpace::Load over] Tick [79]: Thread [1]
[AddrSpace::Execute over] Tick [79]: Thread [1]
[RemoveFromQueue] Tick [98]: Thread [2] is removed from queue L2
[ContextSwitch] Tick [98]: Thread [2] is now selected for execution, thread[1] is replaced, and it has executed [0] ticks
Switching from: 1 to: 2
[InsertToQueue] Tick [109]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [109]: Thread [2] is removed from queue L2

[InsertToQueue] Tick [120]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [120]: Thread [2] is removed from queue L2
[InsertToQueue] Tick [130]: Thread [1] is inserted into queue L3


[UpdatePriority] Tick [608]: Thread [1] changes its priority from[40] to [50]
[RemoveFromQueue] Tick [608]: Thread [1] is removed from queue L3
[InsertToQueue] Tick [608]: Thread [1] is inserted into queue L2
2[RemoveFromQueue] Tick [665]: Thread [1] is removed from queue L2
[ContextSwitch] Tick [665]: Thread [1] is now selected for execution, thread[2] is replaced, and it has executed [600] ticks
[UpdateRemainingBurstTime] Tick [665]: Thread [2] update remaining burst time, from : [4000] - [600], to [3400]
Switching from: 2 to: 1
[InsertToQueue] Tick [666]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [666]: Thread [2] is removed from queue L2
[ContextSwitch] Tick [666]: Thread [2] is now selected for execution, thread[1] is replaced, and it has executed [0] ticks
Switching from: 1 to: 2
[InsertToQueue] Tick [677]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [677]: Thread [2] is removed from queue L2

[InsertToQueue] Tick [688]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [688]: Thread [2] is removed from queue L2
[InsertToQueue] Tick [698]: Thread [1] is inserted into queue L2


[UpdatePriority] Tick [1108]: Thread [1] changes its priority from[50] to [60]
[RemoveFromQueue] Tick [1108]: Thread [1] is removed from queue L2
[InsertToQueue] Tick [1108]: Thread [1] is inserted into queue L2
2[RemoveFromQueue] Tick [1233]: Thread [1] is removed from queue L2
[ContextSwitch] Tick [1233]: Thread [1] is now selected for execution, thread[2] is replaced, and it has executed [600] ticks
[UpdateRemainingBurstTime] Tick [1233]: Thread [2] update remaining burst time, from : [3400] - [600], to [2800]
Switching from: 2 to: 1
[InsertToQueue] Tick [1234]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [1234]: Thread [2] is removed from queue L2
[ContextSwitch] Tick [1234]: Thread [2] is now selected for execution, thread[1] is replaced, and it has executed [0] ticks
Switching from: 1 to: 2
[InsertToQueue] Tick [1245]: Thread [2] is inserted into queue L2
[RemoveFromQueue] Tick [1245]: Thread [2] is removed from queue L2
```

This is just part of the

output.
other test case:
userprog/nachos -epb test/hw2_test1 40 5000 -epb test/hw2_test2 40 4000 -d z
userprog/nachos -epb test/hw2_test1 90 5000 -epb test/hw2_test2 100 4000 -d z

- Report
  Including your results of code tracing, how you implement your system calls, and your team member contribution.
  File name: Final_Project_<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**
  1. Report — 30%
  2. Demo — 70%
     (a) Implementation correctness — 50%
     (b) Question — 20%(Individual score)

- Please upload your report to eeclass before 2024/06/20 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.