Assignment 2 Design Document

Alston Huang:1471706,

Kevin Lee:1480757

CMPS 111, Winter 2019

1.Goal

The goal of this program is take user time-sharing processes that are being added to the run queue and add it to our own lottery queue. Then when the kernel wants to run a user time-sharing processes go to the lottery queue and use the rule ("The probability of picking process i = (the number of tickets of process i)/(the total number of tickets for all processes)") as the probability to pick a process. Then remove the process from the queue. For every add and remove we print (1) the event type, add or remove, (2) the size of the queue, (3) the smallest number of tickets of a process in the lottery queue, (4) the largest number of tickets of a process in the lottery queue, (5) the total number of tickets of all processes, (6) the number of tickets of the process added to or removed from the lottery queue.

2.Assumptions

3.Design

The general approach is pretty simple. We first initialize a queue called lottery queue. Then in tdq\_runq\_add we filtered out non-rooted user time-sharing processes by using its priority. If the priority is between 120 and 223 then it will be a user time-sharing process and we add it into our priority queue. Next in tdq\_choose when it chooses a user time-sharing process it will go into our lottery choose function. This function will generate a random ticket based on a random number divided by the total niceness + 1. The +1 is there because you can’t divide by zero. Then we set the head of the queue and grab the amount of tickets the first process so we can reference its value when we compare it to the first value

(other-wise we would be comparing the first value to 0). Then we iterate through our lottery queue. While iterating we check if the random ticket generated is less than all the ticket values previously added up. If so we know we have found the winner. Otherwise we add the current amount of tickets the current process has and iterate to the next process. If there is nothing in the lottery queue return null. Lastly when our lottery queue is called to be removed in tdq\_runq\_rem we run our own lottery\_remove. In this function we initialize the head of the lottery\_queue. Then we call TAILQ\_REMOVE with the head, thread and the lottery\_queue and remove the thread from lottery\_queue.

4. pseudocode

procedure tdq\_runq\_add

if non-root process

if priority >= 120 && priority <= 223 then

runq\_add\_lottery(lottery\_queue, thread)

end if

end procedure

procedure runq\_add\_lottery

rhq = lotter\_queue head

TAILQ\_INSERT\_HEAD(rhq, thread, lottery\_queue)

Procedure tdq\_choose

Thread = runq\_lottery\_choose

If thread != null then

Return thread

end if

end procedure

procedure runq\_lottery\_choose

random ticket = random number % (total niceness + 1)

rqh = header of lottery\_queue

current\_ticket\_value = total tickets of first process

TAILQ\_FOREACH(thread, head, lottery\_queue) do

If random ticket >= current\_ticket\_value then

Return thread

End if

Current\_ticket\_value = Current\_ticket\_value + thread->pnice + 21

End foreach

Return null

end procedure

procedure runq\_lottery\_remove

TAILQ\_REMOVE(head, thread, lottery\_queue)

end procedure

procedure tdq\_runq\_rem

if(ts->ts\_runq == &tdq->lottery\_queue)

runq\_lottery\_remove(ts->ts\_runq, td);

end if

end procedure