Design Doc : proj3

Name: Kartik Mathur, Tanay Shah

USC IDs: 7813368313, 9369146501

Following are the xv6 files that were modified in the process of completing the Project 2:

Task 1: (scheduler)

In the scheduler function, we fetch the best process according to the ranking on the priority queue. If q0 has an element then we choose the first one and if it's empty, then we move to q1 and repeat. The scheduler runs forever and picks an available process from the ptable. If that process is not the "best" process, it continues until it finds it.

If no ticks have elapsed after the process has been scheduled, then it has voluntarily yielded the resource. Thus if endTicks - beginTicks is 0, it is a voluntary exit.

If the time slice is used up, we downgrade the process to the next queue.

The Aging mechanism checks through the all runnable processes on q2. If the process has waited for 50 or more ticks, then we upgrade the process to q0 where it will likely run much faster. So we dequeue from q2 and add to q0.

Some of the important helper functions we implemented are:

Dequeue: Remove the specified process from the queue by emptying the index. Then shift all the right sided elements to the left. This is used for all the queues each time a process is run or moved.

Addtoqueue: Add the process to the specified queue by finding an empty index. We call this once in allocproc to ensure that new processes are added to q0.

Task 2: (getpinfo())

In this, we edited *proc.c* file where we added *getpinfo()* and all the other file such as *user.h, sysproc.c, sysproc.h* as we did in hw1 to add a new system call. The implementation of it has been added in *proc.c* file. The function takes the input parameter as the process id whose process info needs to be printed. The code loops through ptable to find the right process and then prints out the process name and id and other info like ticks and times and then loops through the pstat array and print all the stats about scheduling for that process.

Task 3: (Test cases)

In this we added the three test case files for testing the scheduler.

Test1.c:

For test 1, we are testing two workloads on the scheduler, and IO intensive and other one is CPU intensive. This files create a child process and runs an IO intensive test on it while it runs a CPU intensive on the main process.

- IOtest.c:
    - This file contains the IO intensive test, where it creates a big file, writes stuff to it, closes it and then opens it reads from it and then deletes it. We developed this test based on the usertests.c's bigfile test.
- CPUtest.c:
    - This file contains a CPU intensive workload where it calls a function called factorial, which has multiple recursive call, which makes the process slow and time consuming.

Test2.c

The purpose of this file is to test the priority boost system in the scheduler, and how it prevents the process from starvation. This code creates 10 different processes and runs a CPU intensive process from the test1 for each child process. This workload demonstrates the the process that stays more than 50 ticks in lowest priority get moved to the highest so that it does not starve.

Test3.c

The purpose of this test it shows how the MLFQ scheduler can be gamed, and process always stays in the highest priority. We achieved this by creating a child process that sleeps on creation and never yields and so no other process can switch back into the scheduler, and this process stays in the highest priority and no cpu ticks happen at that point. We achieve by creating loops that makes the process sleeps for 50 seconds for 5 times.

Task 4: (Graphs)

We added the graphs that demonstrates visually how these all the test cases run. For more information on what they show, refer wrokload.pdf.