ASSIGNMENT 5 XV6

How to run

1. Run on this terminal as make clean ; make qemu SCHEDULER= PLOT
2. Here the flag1 is RR, FCFS,PBS,MLFQ. If SCHEDULER is not mentioned defualt we are taking RR. Flag2 is YESPLOT , NOPLOT, If flag2 is YESPLOT then the required things for a graph will be printed to times.txt file and later used for plotting graph with python.

I have written the performance evalutions and graph plots in report.pdf , Here I have written implementations of things asked in assignment. I made a change in make file for flags;

```
ifndef SCHEDULER
SCHEDULER := RR
endif
ifndef PLOT
PLOT := NOPLOT
endif

CFLAGS += "-D$(SCHEDULER)"
CFLAGS += "-D$(PLOT)"
```

IMPLEMENTATION

TASK1:

waitx syscall

int waitx(int *wtime,int *rtime) ; for implementing this I have added

1. ctime -> creation time of a process innitialised to ticks in allocproc() fn
2. etime -> endtime of a process intialised to 0 and equating to ticks in exit() fn
3. rtime -> runtime of a process initialised to 0 and doing ++ in updatetimes() when the process is in running state.
4. iotime -> iotime of a process initialised to 0 and doing ++ in updatetimes() when the process is in sleeping state. And waitx is the modification of wait syscall with just changing to things

```
*wtime=p->etime - p->ctime - p->iotime - p->rtime;
*rtime=p->rtime;
```

As for adding the syscall we have to add the some statements in these code sysproc.c , user.h , usys.S , syscall.h, syscall.c , sysproc.c , defs.h , proc.c , proc.h.

and a userprogram is time.c in that i am calling waitx to know that waiting and runtime of a command, the command in terminal is time command is like ls , echo etc;

```
$ time ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 16356
echo           2 4 15208
forktest       2 5 9520
grep           2 6 18576
init           2 7 15796
kill           2 8 15236
ln             2 9 15092
ls             2 10 17724
mkdir          2 11 15336
rm             2 12 15316
sh             2 13 27952
stressfs       2 14 16224
usertests      2 15 67336
wc             2 16 17088
time           2 17 15832
benchmark      2 18 16144
setPriority    2 19 15264
ps             2 20 16056
zombie         2 21 14908
console        3 22 0
wait time 1 , run time 24
```

ps user program

for this I made a ps syscall in that i am passing a array of data type struct procstatus. the struct is as follows.

```
struct procstatus {

    int pid;                    // Process ID

    char state[15];             //state

    int rtime;                  //run time

    int priority;               //default priority of the process is 60
    int w_timeforrunning;       //Time for which the process has been
waiting (reseting this to 0 whenever the process gets to run on CPU or if a
change in the queue takes place (in the case of MLFQ scheduler))
    int n_run;                  //Number of times the process was picked by
the scheduler
    int cur_q;                  //Current queue
    int q[5];                   // Number of ticks the process has received
at each of the 5 queues
};
```

I am taking these quantities form ps fn form ptable and printing these in ps.c userprogram.

updatetimes()

in this fn i am doing the updation of runtimes,iotimes,lastexecuted , currentslice and qticks, as this fn is called from trap.c when there is timer interrupt to update ticks so all the mentioned above things are related to ticks so the updation of them also should be with ticks.

```
void updatetimes(void)
{
  struct proc *p;
  acquire(&ptable.lock);
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
  {
    if(p->state==RUNNING)
    {
      p->rtime++;
      p->lastexecuted=ticks;
      p->currentslice++;
      if(p->cur_q!=-1)
      {
        p->q[p->cur_q]++;
      }
    }
    else if(p->state==SLEEPING)
    {
      p->iotime++;
      p->lastexecuted=ticks;
    }
  }
  release(&ptable.lock);
}
```

TASK2

All the scheduling algorithms are written in scheduler fn. each algorithm will be taken according to the flag mentioned that was done in makefile.

FCFS:

in this at first i am finding the least the process which has least creation time. and cpu takes that.

```
for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
{
  if (p->state != RUNNABLE)
    continue;
  if(nxt==0)
  {
    nxt=p;
  }
  else
  {
    if (p->ctime < nxt->ctime)
    {
      nxt=p;
    }
  }
}
```

as here nxt is the process whcih has least ctime in whole proc table. so this nxt will take the cpu

```
  if(nxt==0)
      {
        release(&ptable.lock);
        continue;
      }
      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      c->proc = nxt;
      switchuvm(nxt);
      nxt->state = RUNNING;
      nxt->n_run++;
      swtch(&(c->scheduler), nxt->context);

    //  nxt->w_timeforrunning = 0;
      switchkvm();
      nxt->lastexecuted = ticks;

      // Process is done running for now.
      // It should have changed its p->state before coming back.
      c->proc = 0;
```

and In assingment they said to do nonpremptive fcfs so i am not doing yield() when the scheduler is fcfs

```
  if(myproc() && myproc()->state == RUNNING && tf->trapno ==
  T_IRQ0+IRQ_TIMER)
    {
      #ifndef FCFS
      yield();
      #endif
    }
```

PBS:

1. For all the processes I am keeping 60 as the default priority for a process and for exec processes I am doing the priority as 3 and this 60 is assigning in allocproc().

2. here also finding min priority from the ptable.

```
struct proc *nxt = 0;
struct proc *next=0;
for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
{
  if (p->state != RUNNABLE)
    continue;
  if (nxt == 0)
  {
    nxt = p;
  }
  else
  {
    if (p->priority < nxt->priority)
    {
      nxt = p;
    }
  }
}
```

- In this assignment we have to do that the same priority process should run according to Round robin. So at first we got the process with minimum priority. as there can be multiple proccesses which have same priority so thate means this minimum priority also have multiple processes, so for that I am looping through the ptable and finding the first procses in the same priority.
- As we given to the cpu after that we will again loop through ptable and checking that the priority of any process is increased(i.e the value is decreased) if one of the process is increased then we break the loops and schedule again. if no ones priority is decreased then no need to break and continue in that loop this gives a RR fashion for same priority.

```
 for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    {
      if (p->state != RUNNABLE)
        continue;
      if (p->state==RUNNABLE && nxt->priority == p->priority)
      {
        // Switch to chosen process.  It is the process's job
        // to release ptable.lock and then reacquire it
        // before jumping back to us.
        c->proc = p;
        switchuvm(p);
        p->state = RUNNING;
        p->n_run++;
        swtch(&(c->scheduler), p->context);

   //     p->w__timeforrunning = 0;
        switchkvm();
        p->lastexecuted = ticks;
        // Process is done running for now.
        // It should have changed its p->state before coming back.
        c->proc = 0;
        int flag=0;
        for(next = ptable.proc; next < &ptable.proc[NPROC]; next++)
        {
          if(next->state!=RUNNABLE)
            continue;
          if(next->state==RUNNABLE && next->priority < nxt->priority)
          {
            flag=1;
            break;
          }
        }
        if(flag==1)
        {
          break;
        }
      }
    }
```

- The processes can have priorities from range [0,100] , and we can set the priority of the process by calling a system call int set_priority(int new_priority,int pid) This system call is not given we have to do it. for this i am writiing a syscall set_priority() and adding a system call has been done in waitx thing here is also same way , In this fn if the new_prioriy is not in range of [0,100] or pid<0 then it is a invalid thing and i am checking that whether the pid is there or not if it is there then we will change the priority of the pid and we will compare old priority of the procces with new_priority if new_priority < oldpriority then that means the processes priority increased (lower the value higher the priority).
- As the priority is increased we have to schedule again because this proccess priority may be greater than the running proccesses priority then we have to prempt the runninh procces. so in this case we have to reschedule again so thats why we are calling yeild() fn when new < old other wise noneed , And if the pid is not there then also it prints error.

```
int
set_priority(int new_priority,int pid)
{
  struct proc *p;
  int flag=0;
  if(new_priority<0 || new_priority>100 || pid<0)
  {
    return -1;
  }
  acquire(&ptable.lock);
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
  {
    if(p->pid==pid)
    {
      flag=1;
      break;
    }
  }

  if(flag==0)
  {
    release(&ptable.lock);
    return -1;
  }
  int old_priority=p->priority;
  p->priority = new_priority;
  release(&ptable.lock);
  if(new_priority <  old_priority)
  {
    yield();
  }
  return old_priority;

}
```

For this system to call we have a setPriority.c file where it calls this to set priority. and the syntax of the command to run on terminal should be setPriority <new_priority> .

MLFQ:

1. In mlfq we have 5 queues each has different priorities i.e 0 has high priority and 4 has low priority , and each as time slices as 1,2,4,8,16 i.e if the process runs greater that the ticks in that queue the demote the process to lower priiority queue.
2. Here I am doing aging for the process when their waiting time is greater than some time. this the thing stores the max ticks for a queue.

```
  int maxarr[]={1,2,4,8,16}; //max ticks array in queue
```

The code below is the code for aging. here i am checking that ticks - p->lastexecuted > 50 the do aging i.e changing the current queue of a procces to higher priority queue. Here the lastexecuted is the tick when the process last ran.

```
   for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
      {
        if(p->state!=RUNNABLE)
          continue;
        if(ticks - p->lastexecuted > 50)
        {
          if(p->cur_q!=0)
          {
            p->cur_q--;
            #ifdef YESPLOT
            cprintf("%d %d %d\n",ticks,p->pid,p->cur_q);
            #endif
            p->currentslice=0;
            p->lastexecuted=ticks;
          }
        }
      }
```

the cur_q variable is initialised to -1 for other scheduling algos and for mlfq it is 0;

- In my implementation i am calculating the top of a each queue, As i am iterating through queues as in order like 0 to 4 in each iteration i am checking whole ptable and which has less value of lastexecuted then making the procces as top of that queue and after that ptable for loop we will get the top of the currernt queue i.e i.
- so we have a while loop running until the top is in runnable state in that we will assign the cpu to the process every time in this loop and we are checking that how much is the process taking the currentslice like how time it is running on cpu. we have max ticks for each queue . we have to compare the currentslice and maxticks in corresponding queue if currentslice is greater then break form tha loop and demoting the procces (i.e moving the lower process to lower priority) and incresing the noof runs and updating the lastexecuted to ticks and break form the iterating queue loop and again we will do the same.

```
struct proc *top=0;
    for(int i=0;i<5;i++)
    {
      for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
      {
        if (p->state != RUNNABLE)
          continue;
        if(p->cur_q==i)
        {
          if(top==0)
          {
            top=p;
          }
          else if(p->lastexecuted < top->lastexecuted )
          {
            top=p;
          }
        }
      }
      if(top==0)
        continue;

      top->currentslice=0;
      while(top->state==RUNNABLE)
      {
        c->proc = top;
        switchuvm(top);
        top->state = RUNNING;
```

```
          swtch(&(c->scheduler), top->context);

          //p->w_timeforrunning=0;
          switchkvm();


          // Process is done running for now.
          // It should have changed its p->state before coming back.
          c->proc = 0;
          if(top->currentslice>=maxarr[top->cur_q])
          {
            break;
          }
        }
        if (top->currentslice >= maxarr[top->cur_q])
        {
          if (top->cur_q != 4)
          {
            //demote priority
            // if(top->cur_q==-1)
            // {
            //   if(top->state==ZOMBIE)
            //   {
            //     cprintf("ZOMBIE\n");
            //   }
            // }
            if(top->state !=ZOMBIE)
            {
            top->cur_q++;
           #ifdef YESPLOT
            cprintf("%d %d %d\n", ticks, top->pid, top->cur_q);
            #endif
            }
          }
          top->currentslice = 0;
        }
        top->n_run++;
        top->lastexecuted=ticks;
        break;
      }
```

Testing

for testing the code i have created my test.c for different cases I have 6 cases

1. normal benchmark
2. only cpu bound processes
3. only io bound processes
4. mixed of cpu and io processes
5. it is random cpu and io processes
6. late fcfs this testcase is for testing explixitly it has first half cpu bounds and second half have iobounds.

how to use is like test

BONUS

- plotting graphs
- I will print ticks, pid, currentqueue when there is a aging or demotion in mlfq.
- And I am printing these only when the flag is like PLOT=YESPLOT so I made a flag like scheduler to print or not print these
- for ploting graph u need to do like this

```
make clean ; make qemu SCHEDULER=MLFQ PLOT=YESPLOT > times.txt
```

if the PLOT flag is not given then default is NOPLOT and the print statements in test.c will print after running this make command run test so the prints will be printed to times.txt file For plotting graphs u have to run "python3 plotgraph.py" in that i am opening times.txt and reading and ploting the graphs according to prints. As we know that in times.txt there is some reduntant lines so i am taking only required ones and copying into timescopy.txt and taking the lines and plotting the graphs.

The Plotted graphs are included in the graphs folder. Also the graphs are included in Report.pdf

Performance tests

- For performance tests we can use benchmark processes end and creation time, this is the amount of ticks taken to run this. the difference between these ticks is explained in report.pdf