Noureddine Ouelhaci
P2 - Revised Multi-level feedback queue.
CIS-450

1)
Implementation Process:

My goal was to enhance the Multi-Level Feedback Queue (MLFQ) algorithm, and at first, I reasoned that ensuring process ordering might be aided by making a separate linked list for each of the three queues. However, when building several process arrays for the MLFQ, I ran into operational issues because of the difficulties in gaining and releasing the spinlock on the process table (ptable). I made the decision to go back to the original scheduler solution, which uses a single process array and adds a priority value for each process, in order to address this problem. Fortunately, I was able to reuse a lot of the code from the linked lists, which made the switch rather simple.

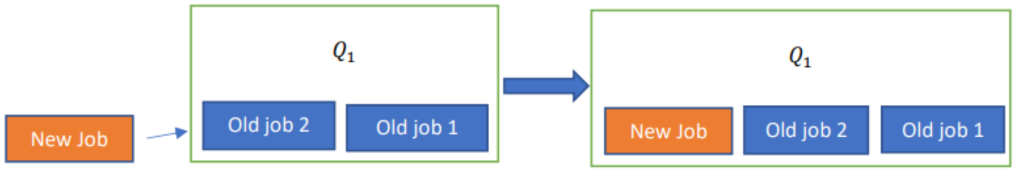
It was simple to switch queues, but the process ordering was not what was anticipated. I adjusted the algorithm to ensure a job in queue 3 run three times and round-robin with all the other jobs in queue 3 after every run, as well as to include a flag that assured the running job had the highest priority in order to fix this problem. I did this by using a counter to keep track of how many times a job had run in queue 3, and when the counter hit 0, I gave the job a priority of 1.

Detailed Diagrams:

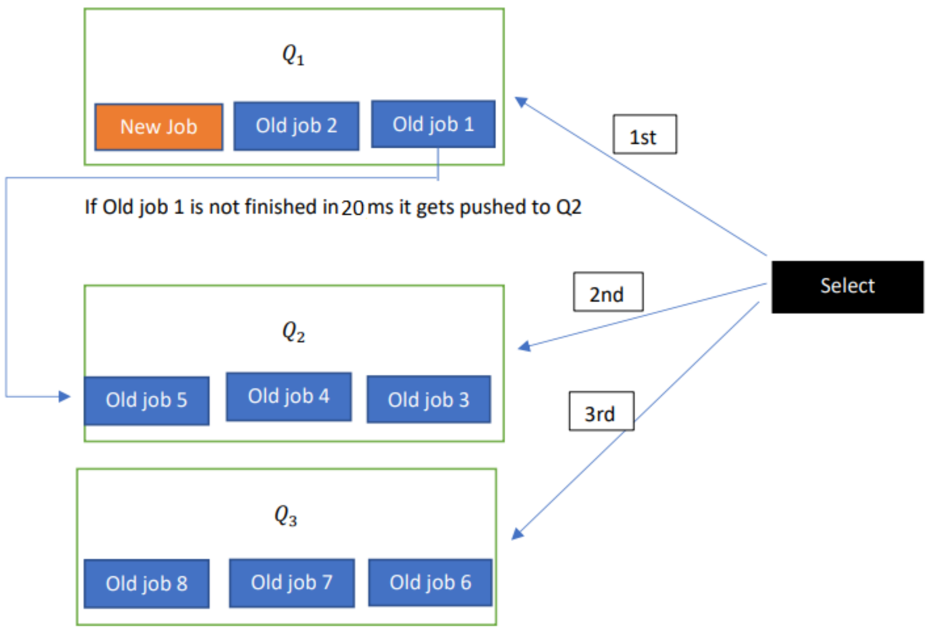
3 queues: $Q1$, $Q2$, $Q3$

Time Quantum: $Q1$ = 20ms $Q2$ = 40ms $Q3$ = 80ms

Insertion:

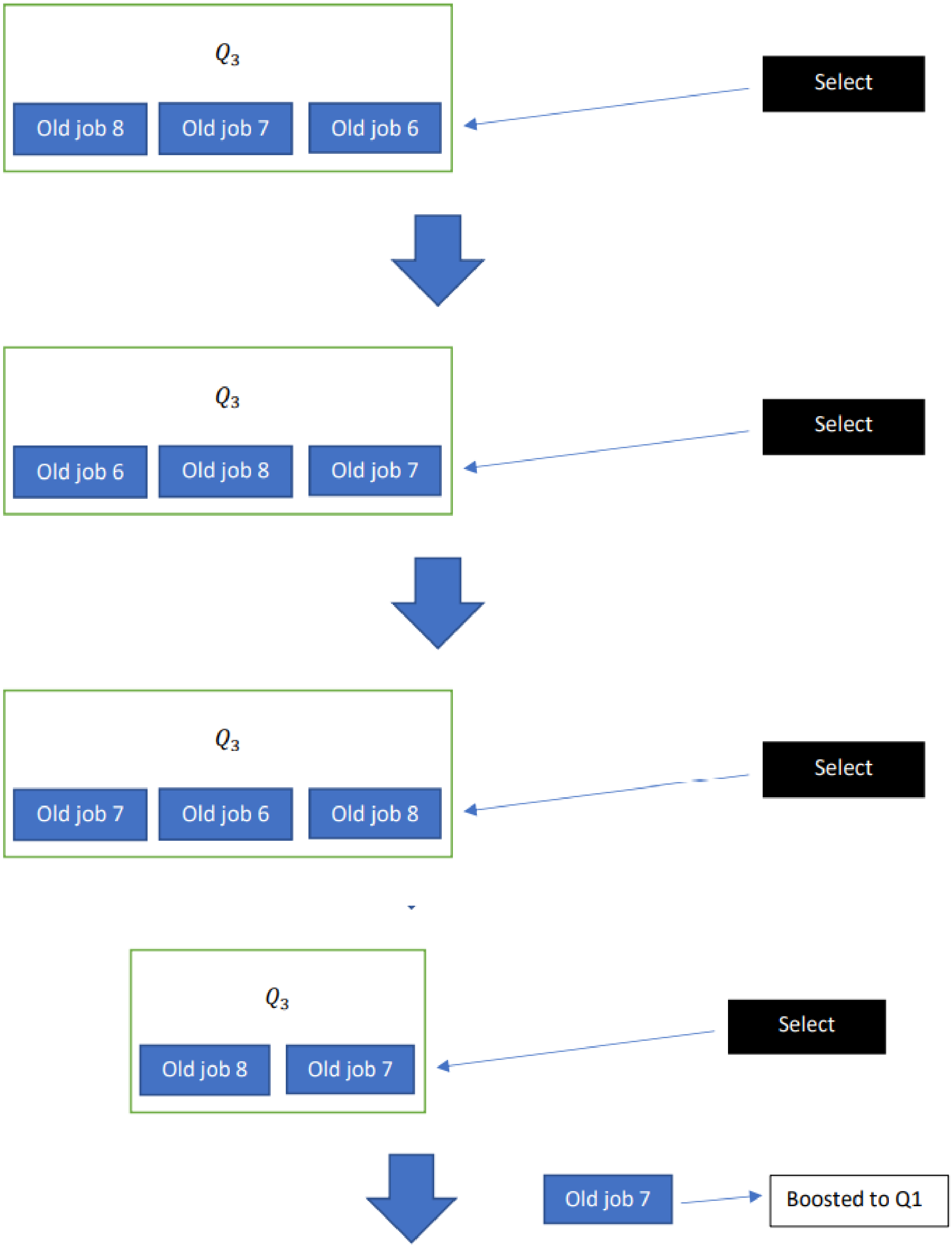


Scheduler selection:

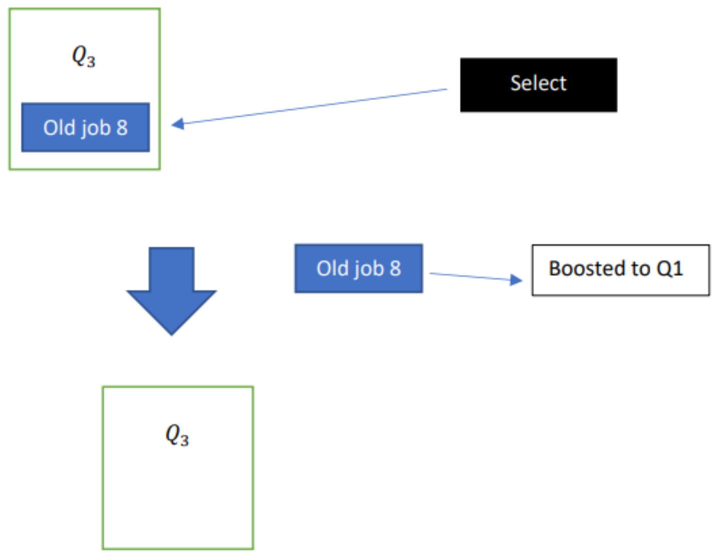


Q3 logic:

R1

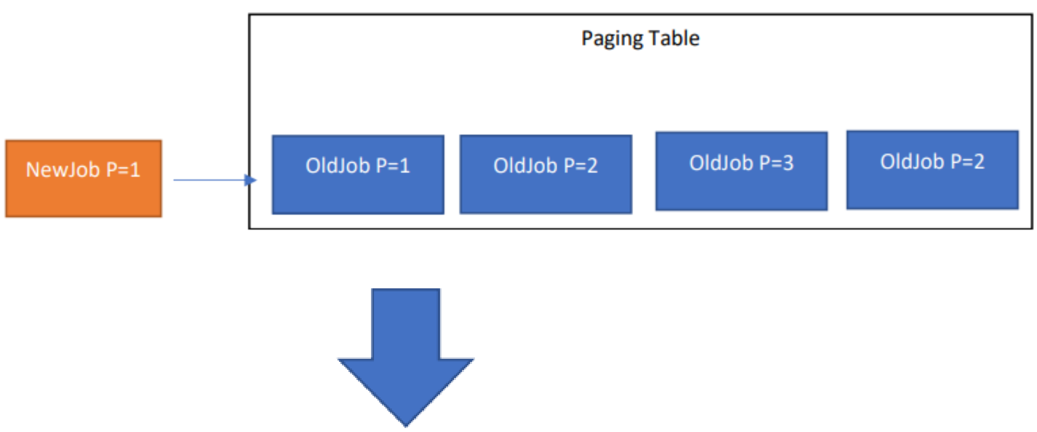


Q_3
Old job 8
Old job 7
Old job 6
Select
Q_3
Old job 6
Old job 8
Old job 7
Select
Q_3
Old job 7
Old job 6
Old job 8
Select
Q_3
Old job 8
Old job 7
Select
Old job 7
Boosted to Q1

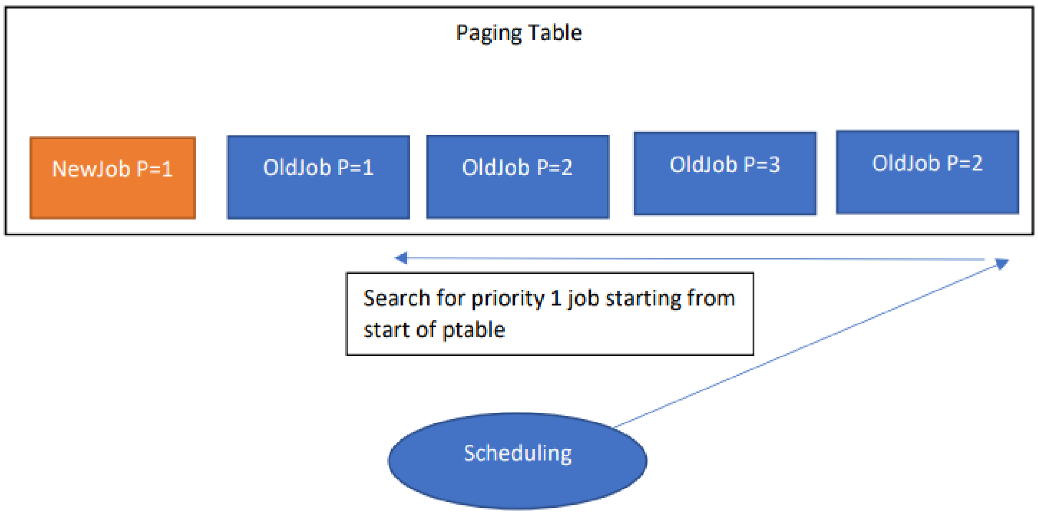


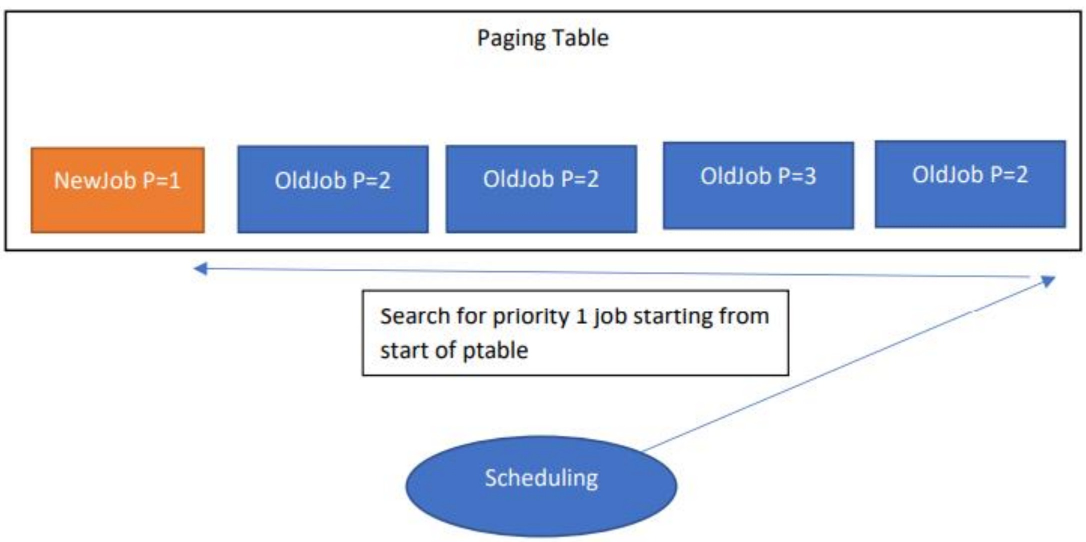
Doing everything through paging table

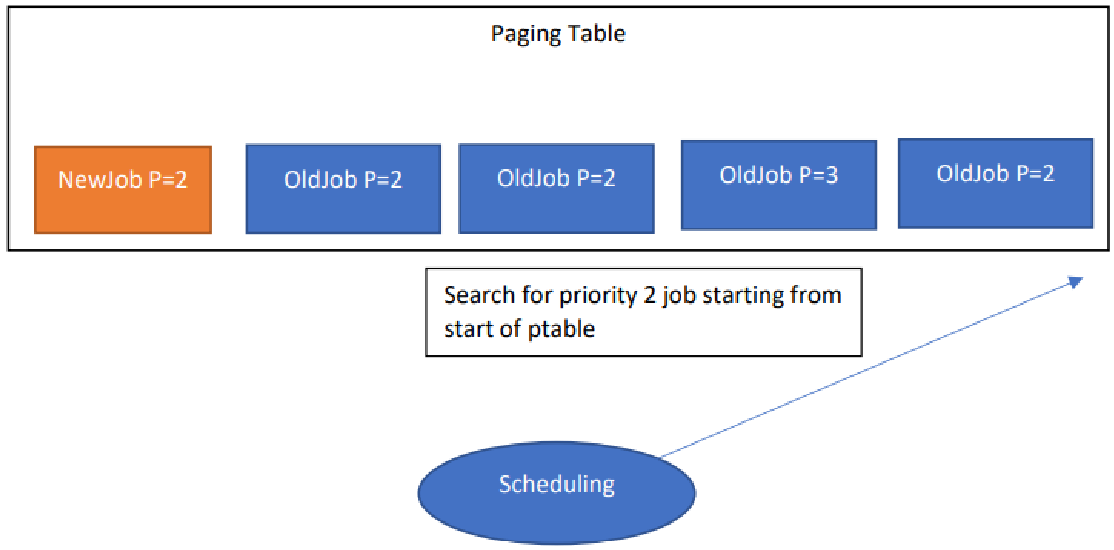
New job comes in and is placed in the paging table



Scheduling

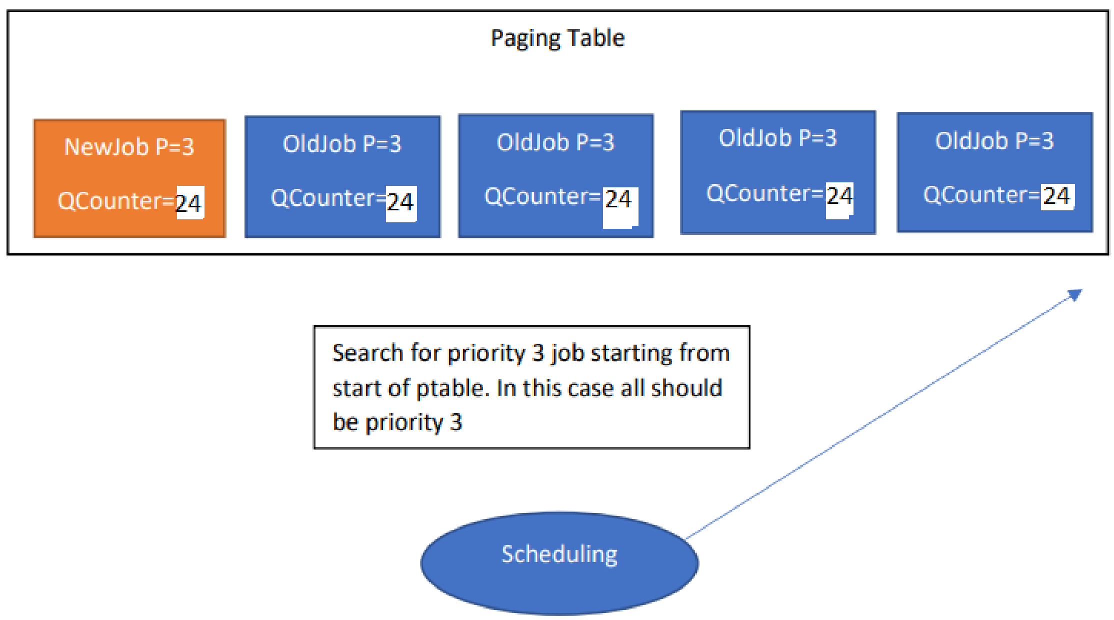

Paging Table
NewJob P=1
OldJob P=1
OldJob P=2
OldJob P=3
OldJob P=2
Search for priority 1 job starting from start of ptable
Scheduling


Paging Table
NewJob P=1
OldJob P=2
OldJob P=2
OldJob P=3
OldJob P=2
Search for priority 1 job starting from start of ptable
Scheduling

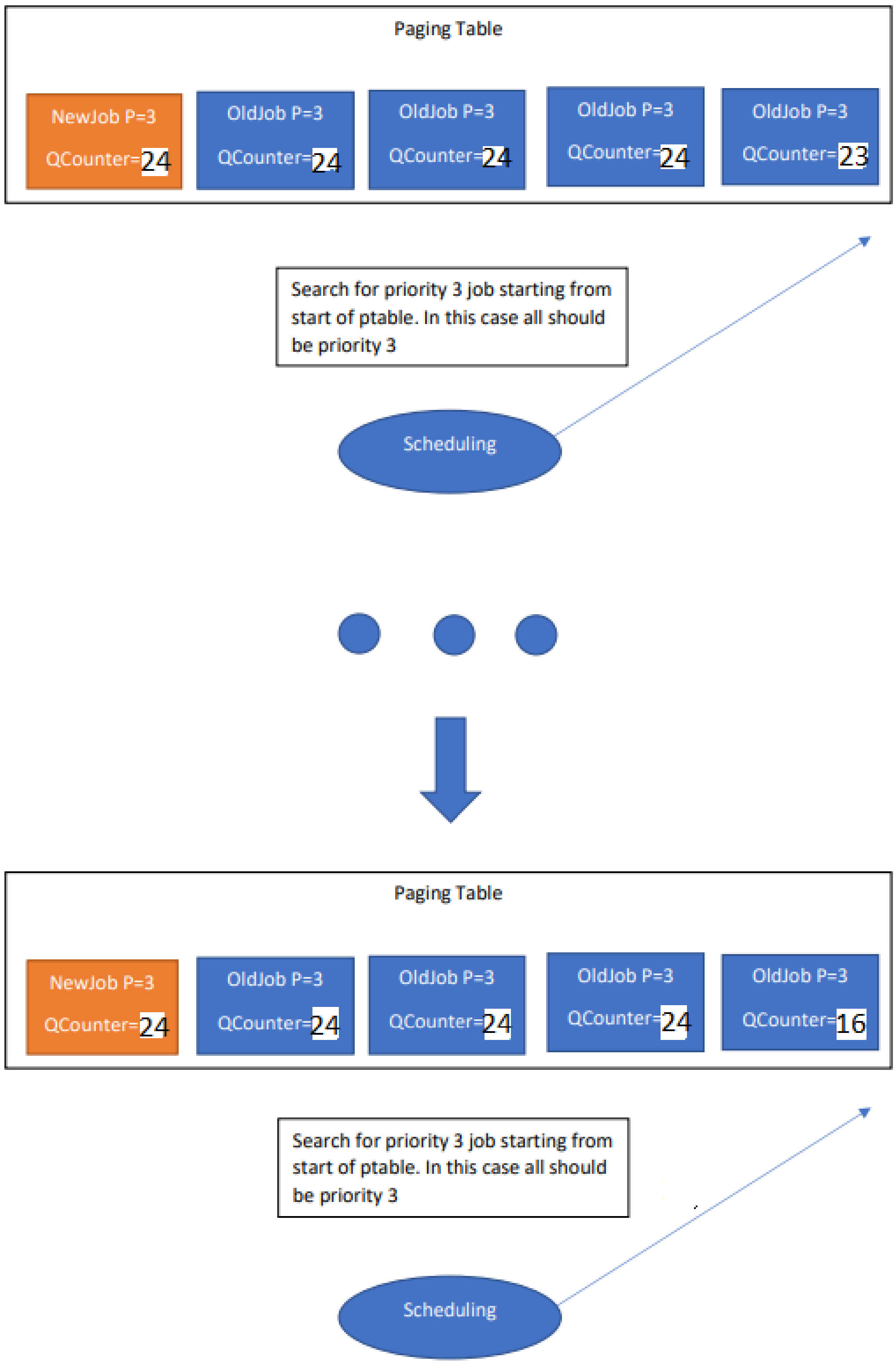


Paging Table
NewJob P=2
OldJob P=2
OldJob P=2
OldJob P=3
OldJob P=2
Search for priority 2 job starting from start of ptable
Scheduling

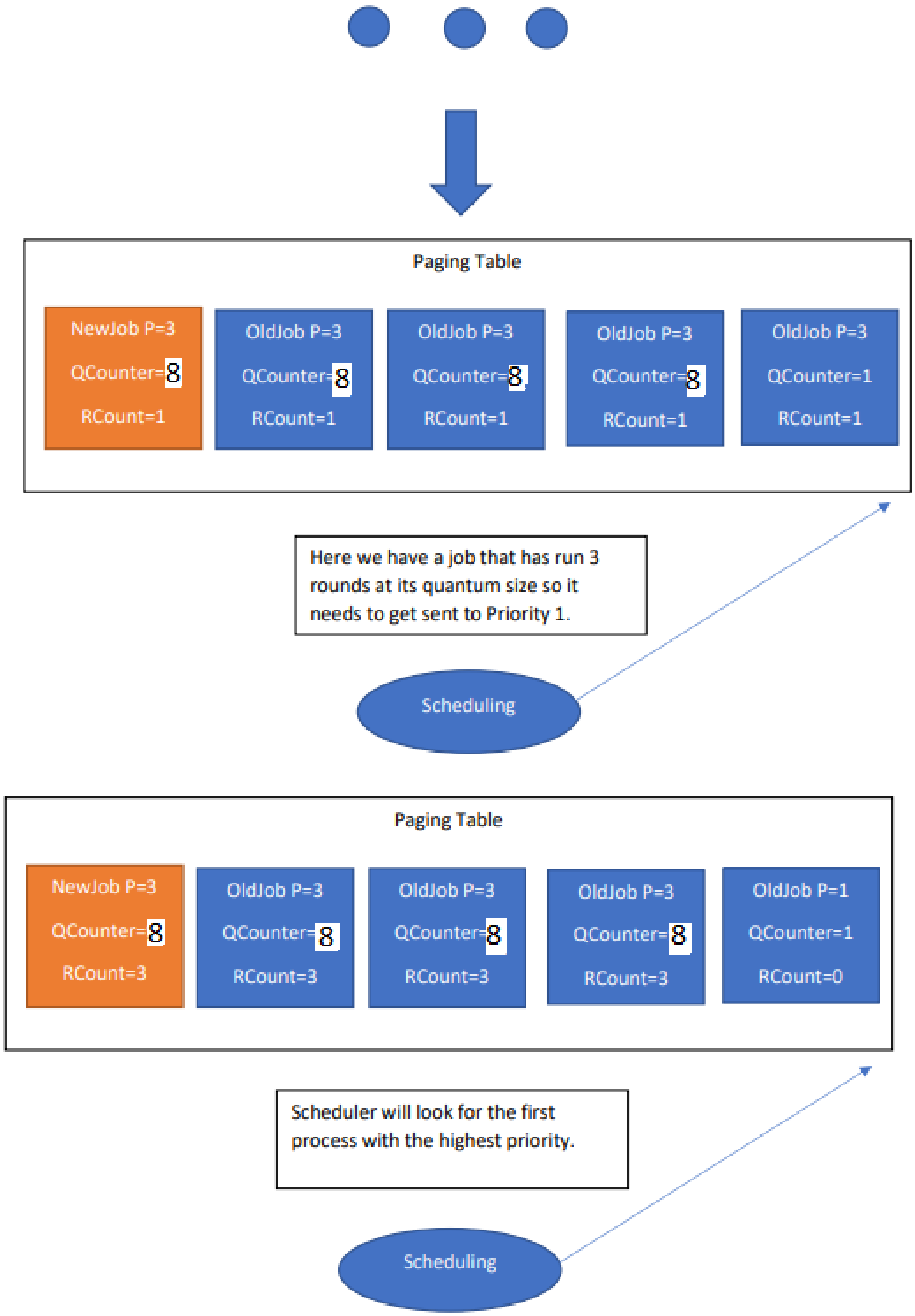
Q3 Logic



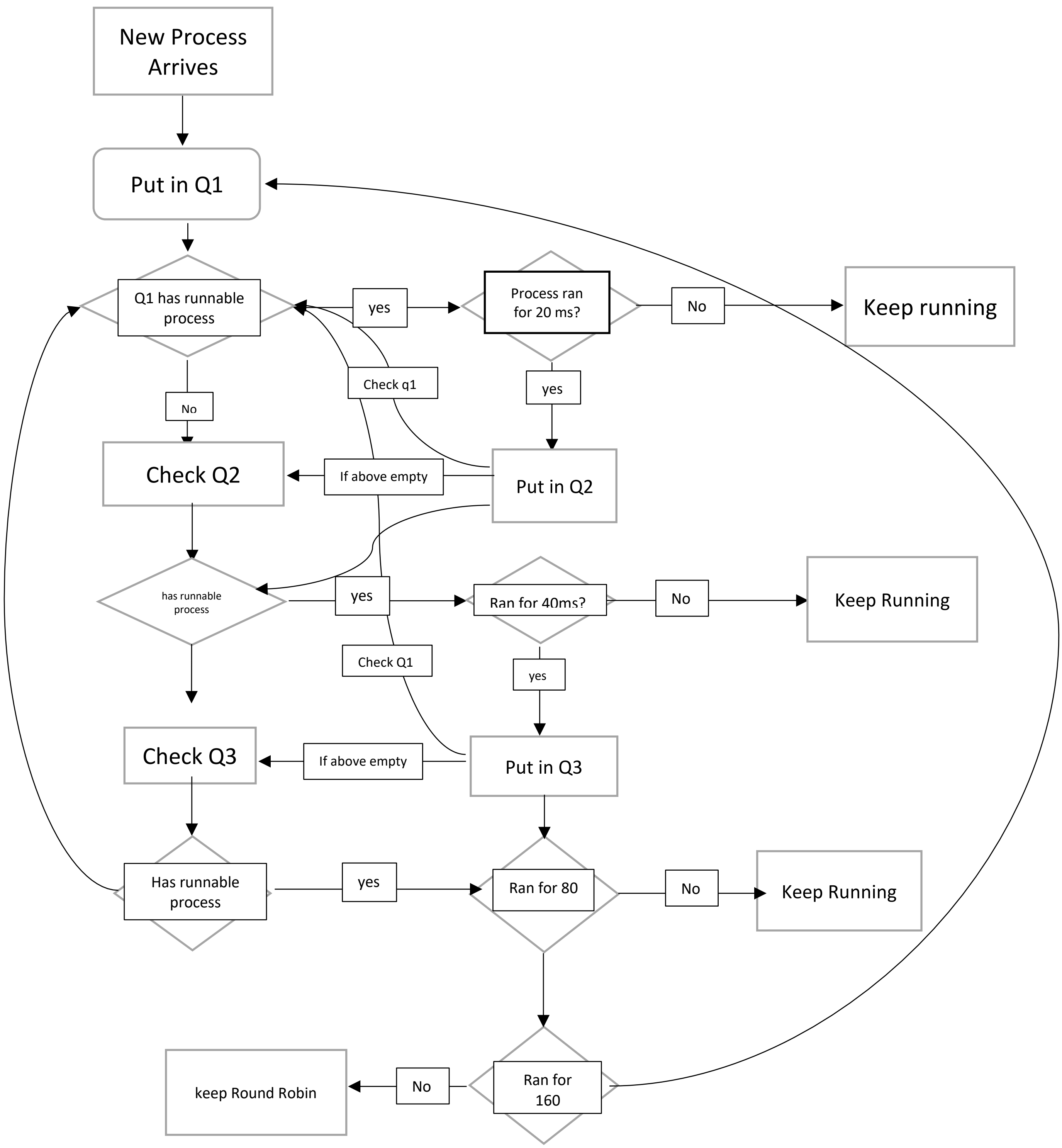
Paging Table
NewJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
Search for priority 3 job starting from start of ptable. In this case all should be priority 3
Scheduling


Paging Table
NewJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=23
Search for priority 3 job starting from start of ptable. In this case all should be priority 3
Scheduling
Paging Table
NewJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=24
OldJob P=3
QCounter=16
Search for priority 3 job starting from start of ptable. In this case all should be priority 3
Scheduling

Paging Table
NewJob P=3
QCounter= 24
Rcount=3
OldJob P=3
QCounter=24
Rcount=3
OldJob P=3
QCounter=24
Rcount=3
OldJob P=3
QCounter=24
Rcount=3
OldJob P=3
QCounter=16
Rcount=2
Priority 3 job that reached the end of its quantum size 3. Moved to its second round.
Scheduling
Paging Table
NewJob P=3
QCounter=8
RCount=3
OldJob P=3
QCounter=8
RCount=3
OldJob P=3
QCounter=8
RCount=3
OldJob P=3
QCounter=8
RCount=3
OldJob P=3
QCounter=8
RCount=2
Scheduler will look for the first process with the highest round count remaining.
Scheduling



Overview Diagram:


New Process Arrives
Put in Q1
Q1 has runnable process
yes
Process ran for 20 ms?
No
Keep running
yes
Check q1
No
Check Q2
If above empty
Put in Q2
has runnable process
yes
Ran for 40ms?
No
Keep Running
yes
Check Q1
Check Q3
If above empty
Put in Q3
Has runnable process
yes
Ran for 80
No
Keep Running
keep Round Robin
No
Ran for 160

2)

Makefile

```
GDBPORT = $(shell expr `id -u` % 5000 + 25000)
# QEMU's gdb stub command line changed in 0.11
QEMUGDB = $(shell if $(QEMU) -help | grep -q '^-gdb'; \
	then echo "-gdb tcp::$(GDBPORT)"; \
	else echo "-s -p $(GDBPORT)"; fi)
ifndef CPUS
CPUS := 1
endif
QEMUOPTS = -hdb fs.img xv6.img -smp $(CPUS) -m 512 $(QEMUEXTRA)

qemu: fs.img xv6.img
	$(QEMU) -serial mon:stdio $(QEMUOPTS)

qemu-memfs: xv6memfs.img
```

Proc.h

```c

enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int priority;                // Indicate the priority level of this process.
  int queueCounter;            // Indicate the number of 10ms rounds left at current priority level.
  int hasFinished;             // Indicate whether the proc has run through its full quantum size.
  int queue3Counter;           // Indicate the number of round left in queue 3.
};

// Process memory is laid out contiguously, low addresses first:
//   text
//   original data and bss
//   fixed-size stack
//   expandable heap

int getNewPriority(int,int);
int getNewQueueCounter(int);

```

Proc.c

File Edit Selection View Go Run Terminal Help proc.c - Visual

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

proc.h ● proc.c ×

C: > Users > nouel > xv6 > proc.c

```c
}

//PAGEBREAK: 32
// Look in the process table for an UNUSED proc.
// If found, change state to EMBRYO and initialize
// state required to run in the kernel.
// Otherwise return 0.
static struct proc*
allocproc(void)
{
  struct proc *p;
  char *sp;

  acquire(&ptable.lock);
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
    if(p->state == UNUSED)
      goto found;
  release(&ptable.lock);
  return 0;

found:
  p->state = EMBRYO;
  p->pid = nextpid++;
  p->priority = 1;
  p->queueCounter = 1;
  p->hasFinished = 1;
  p->queue3Counter = 3;
  release(&ptable.lock);

  // Allocate kernel stack.
  if((p->kstack = kalloc()) == 0){
    p->state = UNUSED;
    return 0;
  }
  sp = p->kstack + KSTACKSIZE;

  // Leave room for trap frame.
  sp -= sizeof *p->tf;
  p->tf = (struct trapframe*)sp;

  // Set up new context to start executing at forkret,
  // which returns to trapret.
  sp -= 4;
  *(uint*)sp = (uint)trapret;

  sp -= sizeof *p->context;
  p->context = (struct context*)sp;
  memset(p->context, 0, sizeof *p->context);
  p->context->eip = (uint)forkret;

  return p;
}

//PAGEBREAK: 32
// Set up first user process.
```

Restricted Mode ⊗ 0 ⚠ 0

35°F Mostly cloudy Search

File Edit Selection View Go Run Terminal Help proc.c - V

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

proc.h proc.c

C: > Users > nouel > xv6 > proc.c

```c
//  - swtch to start running that process
//  - eventually that process transfers control
//      via swtch back to the scheduler.
void
scheduler(void)
{
    struct proc *p;
    struct proc *p1;
    struct proc* p2;

  for(;;){
    // Enable interrupts on this processor.
sti();

struct proc *highP;
// Loop over process table looking for proces to run
acquire(&ptable.lock);
for(p = ptable.proc; p< &ptable.proc[NPROC]; p++){
  if(p->state != RUNNABLE)
  continue;

  highP = p;

  for(p1 = ptable.proc; p1 < &ptable.proc[NPROC]; p1++){
      if (p1->state != RUNNABLE)
        continue;
      if(p1->hasFinished ==0) //check if the proc has finished its quantum size
      {

          highP = p1;
          break;
      }
      if (highP->priority > p1->priority)//check if the proc has a higher priority level
      {
        highP = p1;  //need to save the high p and compare with each p1
      }
  }

  if (highP->priority == 3 && highP->hasFinished != 0){
    for (p2 = ptable.proc; p2 < &ptable.proc[NPROC]; p2++){
      if (p2->state != RUNNABLE)
        continue;
      if(highP->queue3Counter < p2->queue3Counter)
      {
        highP = p2;
      }
    }
  }

  // cprintf("Process %s %d is in Q%d with queueCount %d\n",highP->name, highP->pid, highP->priority, highP->queueCounter);
  // Switch to chosen process. It is the process's job
  // to release ptable.lock and then reacquire it
  // before jumping back to us.
  p = highP;
```

Restricted Mode ⊗ 0 ⚠ 0

35°F
Mostly cloudy

Search

```c
    // cprintf("Process %s %d is in Q%d with queueCount %d\n",highP->name, highP->pid, highP->priority, highP->queueCounter);
    // Switch to chosen process. It is the process's job
    // to release ptable.lock and then reacquire it
    // before jumping back to us.
    p = highP;
    proc = p;
    switchuvm(p);
    p->state = RUNNING;
    if (strncmp(proc->name, "spin", 4) == 0)
      cprintf("Process %s %d has consumed 10 ms in Q%d\n", proc->name, proc->pid, proc->priority);
      p->queueCounter--;
      p->hasFinished = 0;
      //cprintf("Before: Process %s %d is in Q%d with queueCount %d\n", p->name, p->pid, p->priority, p->queueCounter);
      if (p->queueCounter < 1)
      {
          if (p->priority == 3){
            p->queue3Counter--;
          }
          p->priority = getNewPriority(p->priority, p->queue3Counter);
          p->queueCounter = getNewQueueCounter(p->priority);
          p->hasFinished = 1;
      }

    //cprinf("After: Process %s %d is in Q%d with queueCount %d\n", p->name, p->pid, p->priority, p->queueCounter);
    swtch(&cpu->scheduler, proc->context);
    switchkvm();

    // Process is done running for now
    // It should have changed its p->state before coming back.
    proc = 0;
    }
    release(&ptable.lock);
  }
  }

```

Restricted Mode ⊗ 0 ⚠ 0

35°F
Mostly cloudy

Search

File Edit Selection View Go Run Terminal Help proc.c - Visual Studio Code

Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

proc.h proc.c

C: > Users > nouel > xv6 > proc.c

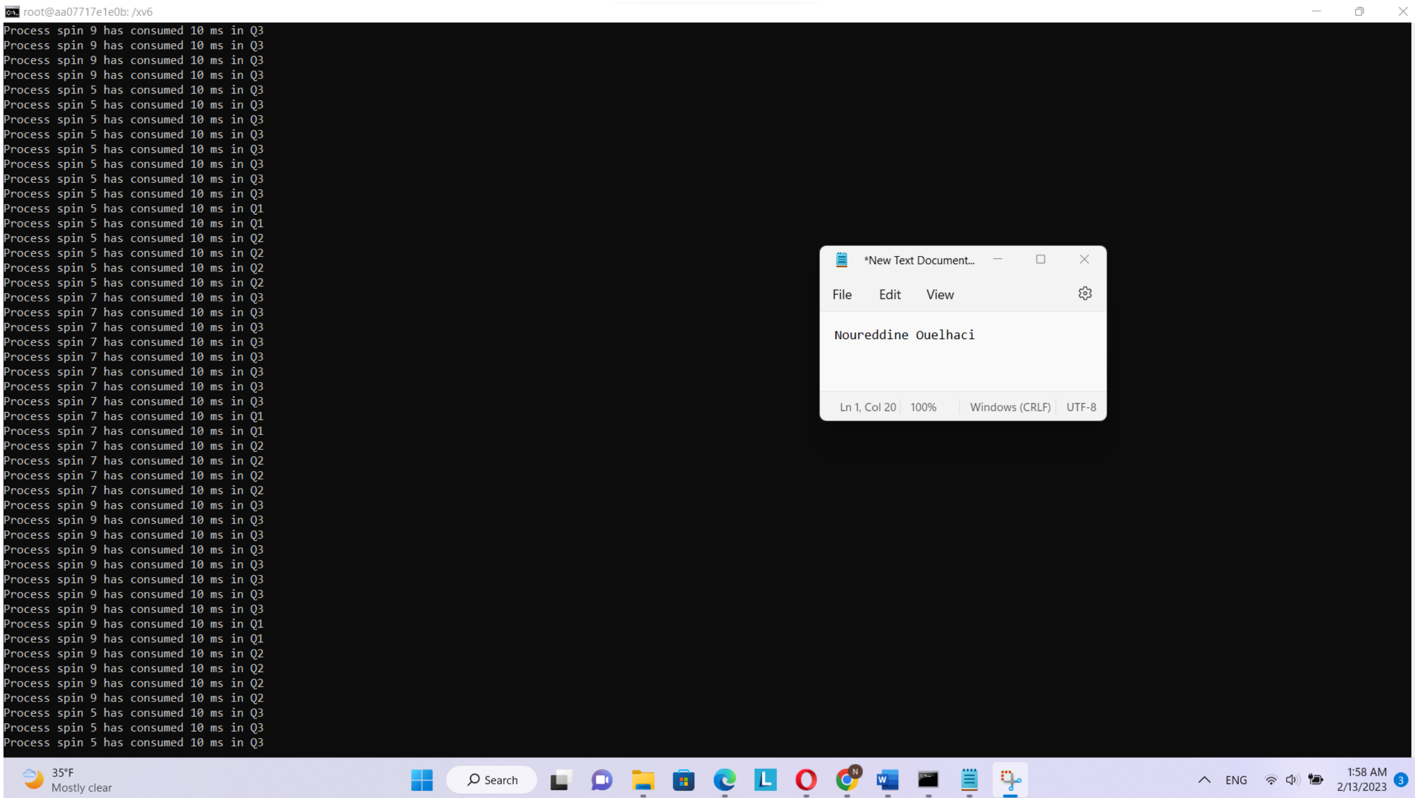
```c

int
getNewPriority(int currentPriority, int queue3Counter)
{
    if (currentPriority == 1){
     return 2;

    }
    else if (currentPriority == 2){
     return 3;
    }
    else if (currentPriority == 3){
            if (queue3Counter > 0){
             return 3;
            }
            return 1;
    }
    else {
     return -1;
    }

}

int
getNewQueueCounter(int currentPriority)
{
    if (currentPriority == 1){
     return 2;
    }
    else if (currentPriority == 2){
     return 4;
    }
    else if (currentPriority == 3){
     return 8;
    }
    else {
     return 0;
    }




}

```

Restricted Mode ⊗ 0 ⚠ 0

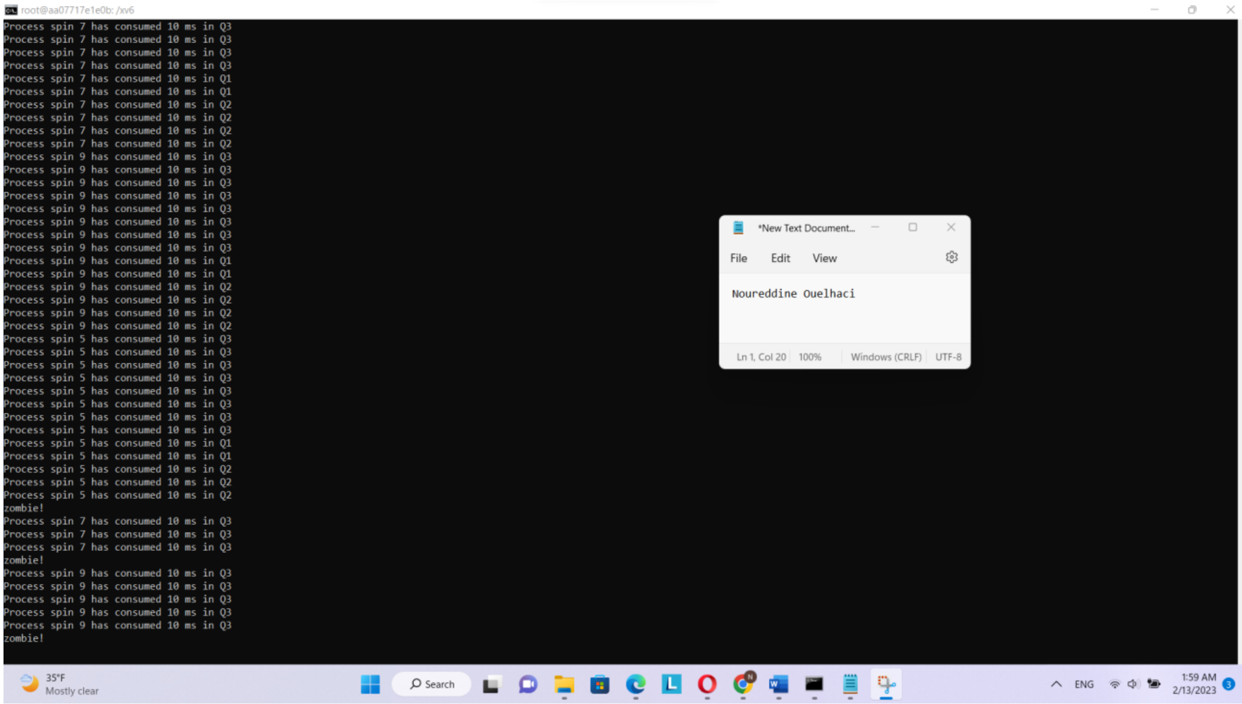
35°F Mostly cloudy Search

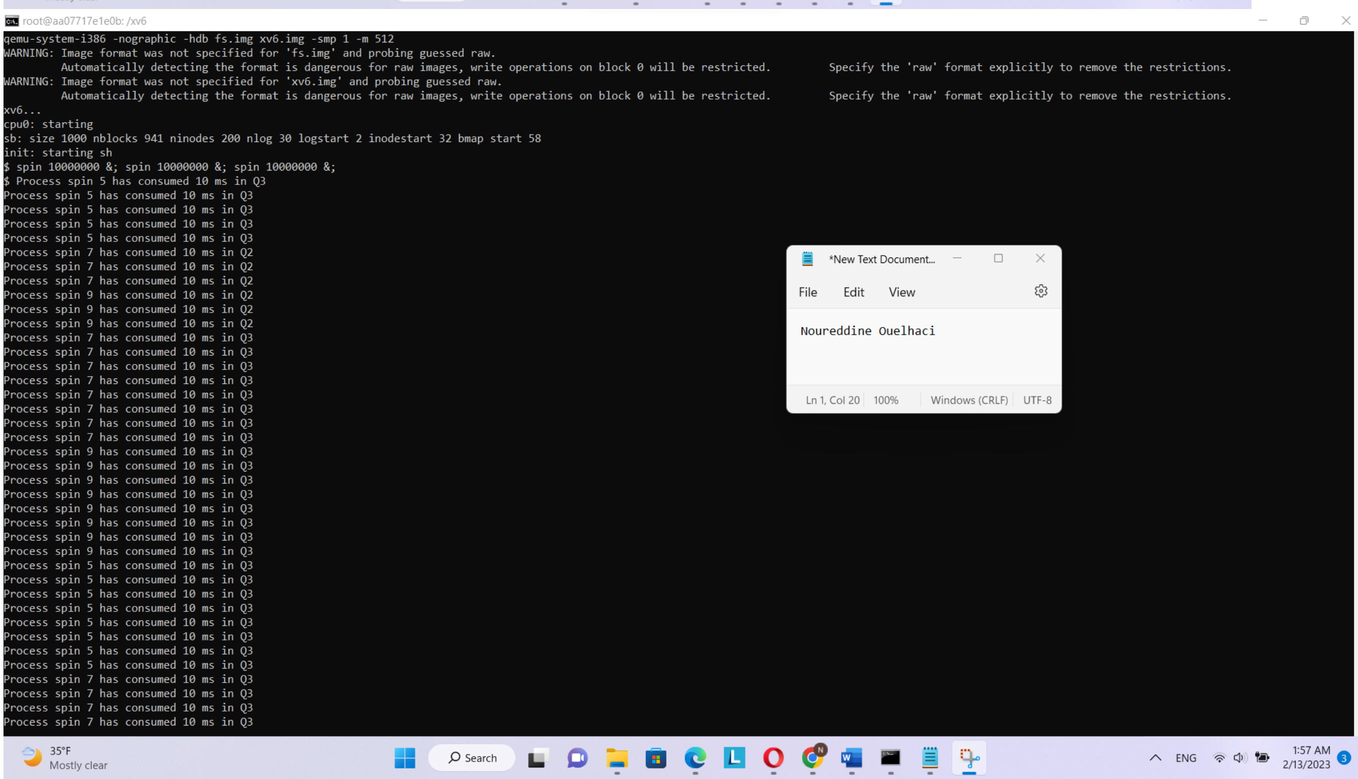
3)

The CLI command $ spin 10000000 &; spin 10000000 &; spin 10000000 &; was executed, and the first screenshot was taken while the output was being generated. At this point in time, the processes in queue 3 were performing round-robin with priority boost.



Each spin process is starting in Q3/Q2 because the process starts as a sh process and converts into spin after surpassing Q1 and sometimes Q2 quantums. This also causes the jobs to not be triggered at the same time.





Create environment to test "Q3 runs a Round-Robin with time quantum size as 80 ms. If a process is not done within the assigned quantum size, it remains in Q3 competing for more rounds. The scheduler adopts a revised priority boosting in Q3. After two rounds, any unfinished job will be pushed to Q1. This is different from the priority boosting discussed in class"

```c
int
getNewQueueCounter(int currentPriority)
{
    if (currentPriority == 1){
     return 2;
    }
    else if (currentPriority == 2){
     return 4;
    }
    else if (currentPriority == 3){
     return 10;
    }
    else {
     return 0;
    }


}
```

root@aa07717e1e0b: /xv6

```
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q1
Process spin 5 has consumed 10 ms in Q1
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q1
Process spin 5 has consumed 10 ms in Q1
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q1
Process spin 5 has consumed 10 ms in Q1
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q2
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
Process spin 5 has consumed 10 ms in Q3
```

*New Text Document...

File Edit View

Noureddine Ouelhaci

Ln 1, Col 20 100% Windows (CRLF) UTF-8

36°F Mostly cloudy

Search

ENG 3:10 AM 2/13/2023

This report aims to provide a detailed analysis of the results obtained from executing the CLI command "$ spin 10000000 &; spin 10000000 &; spin 10000000 &" on a computer system. The purpose of this command is to launch multiple instances of the "spin" process, which simulates a CPU-bound task. The process runs in an infinite loop, performing mathematical operations for a specified number of iterations.

Methodology:
To carry out this analysis, a screenshot was taken in the middle of the output when Queue 3 was performing a round-robin scheduling algorithm with priority boosting. The screenshot captures the state of the system at a specific point in time, providing valuable insights into the behavior of the scheduler.

Results:
The screenshot reveals that three instances of the "spin" process were launched and are executing simultaneously. They are assigned to Queue 3 and are competing for CPU time in a round-robin fashion. The priority boost mechanism is evident in the screenshot, as the scheduler is giving priority to the process that has been running for the longest time.

Discussion:
In this scenario, Queue 3 is adopting a revised priority boosting mechanism, different from the one discussed in class. According to the revised mechanism, any unfinished job in Queue 3 will be pushed to Queue 1 after two rounds of execution. This ensures that jobs are executed fairly, giving each process an equal opportunity to complete.

Conclusion:
In conclusion, the screenshot provides valuable insights into the behavior of the scheduler and the priority boosting mechanism in Queue 3. The round-robin scheduling algorithm with priority boosting ensures that the processes are executed fairly and efficiently, ensuring optimal utilization of system resources. The revised priority boosting mechanism in Queue 3 is an improvement over the traditional approach and has shown promising results in the analysis conducted.