

# Overview

My goal of this assignment is to modify the original CPU scheduling from “RR priority scheduling” to “Lottery Scheduling”.

Lottery Scheduling selects the process which will run in the next round in a random way, which can avoid the starvation problem.

To reach the goals, I have decomposed the assignment into the following major parts:

- Identify the PCB  
In the file named “proc.h”, it designs the structure of the process, which is called Process Control Block (PCB). To implement the lottery scheduling, we should append a new item in it. In my assignment, I named it “numTickets” to indicate the tickets that a process has.
- Where to initialize the information of a process  
Processes can be initialized in two places, one is in “do\_fork” file, another is in “proc.c”. In my assignment, I initialize the information of a process in function “proc\_init”, which is the function in the “proc.c”.
- Where to implement the “Lottery Scheduling”  
Look closer to the file “proc.c”, we can find that there is a function called “pick\_proc”, which returns a process that will run next time. So, I choose to implement the “Lottery Scheduling” in this function.
- Dynamically adjust the “numTickets”  
I append a new function called “ProcessSetPriority” to dynamically adjust the “numTickets”. If the process has used all its time quantum, it will decrease the “numTickets” by 1, otherwise, it will increase the “numTickets” by 1. Furthermore, it will check if the “numTickets” is more than 10 or less than 1, if the situation happens, it will control the “numTickets” to avoid that

situation.

- Define the parameters

In my design, I define the maximum of the “numTickets” that the process has by MAX\_TICKETS, likewise MIN\_TICKETS. Also, when it needs to decrease or increase the “numTickets”, I define DECREASE\_TICKETS and INCREASE\_TICKETS. They are all in “proc.c”.

```
/* Lottery Scheduling */
#define MAX_TICKETS 10
#define MIN_TICKETS 1
#define DECREASE_TICKETS -1
#define INCREASE_TICKETS 1
```

## Major data structures

The major data structures which control the processes is “Queue”. There are several queues in minix, and in each queue, it is like a linked list.

## Major algorithms

The major algorithm is in the function “pick\_proc”. First, I will check all the process to see if it has used all its time quantum or not. If it has used all its time quantum, that means this process may be a CPU-bounded, so I will decrease the “numTickets” of it; otherwise means I/O-bounded, and I will increase their “numTickets”.

```
/* Check if the process has use all the time quantum or not */
for(q = 0; q < NR_SCHED_QUEUES; q++) {
    if(!(rp = rdy_head[q])) {
        continue;
    }
    assert(proc_is_runnable(rp));
    if (rp->p_rts_flags & RTS_NO_QUANTUM) { // decrease tickets
        ProcessSetPriority(rp, DECREASE_TICKETS);
    }
    else{
        ProcessSetPriority(rp, INCREASE_TICKETS); // increase tickets
    }
}
```

Next step is to implement the “Lottery Scheduling”.

In minix, there are several queues. If the number of a queue is smaller, that queue will have the higher priority. So, I decide to divide the queue into two part. One is for the system process; another is for user process. I will only implement the “Lottery Scheduling” in user process to assure that the system process always has the higher priority than user process.

```
/* System process has the higher priority */
for (q=0; q < 6; q++) {
    if(!(rp = rdy_head[q])) {
        TRACE(VF_PICKPROC, printf("cpu %d queue %d empty\n", cpuid, q));
        continue;
    }
    assert(proc_is_runnable(rp));
    if (priv(rp)->s_flags & BILLABLE)
        get_cpulocal_var(bill_ptr) = rp; /* bill for system time */
    return rp;
}
```

```
/* Do the Lottery Scheduling in User process*/
for(q = 6; q < NR_SCHED_QUEUES; q++) {
    if(!(rp = rdy_head[q])) {
        TRACE(VF_PICKPROC, printf("cpu %d queue %d empty\n", cpuid, q));
        continue;
    }

    while (rp != NULL) {
        if(rp->p_rts_flags == 0){
            total_tickets += rp->numTickets;
        }
        rp = rp->p_nextready;
    }
}
```

After calculating the total tickets in the queue, we then generate a random between “0” and “total\_tickets – 1”, and then check which process win the lottery. After finding that process, we will just return it. If there is no winner, we will simply return the “NULL” to indicate that CPU will idle next round.

## Difference between the original and the new one

You will need to investigate how the multilevel Minix scheduler works and how it differs from the lottery scheduler (a dynamic priority queue).

Minix3 uses **multilevel queuing system** for scheduling. Sixteen queues are defined, the lowest priority queue is used only by the **idle process**. User process start by default in a queue several levels higher than the lowest one. Server

processes have higher priorities than user processes and drivers have higher priorities than servers and clock and system task have highest priority.

The new CPU scheduling is called “Lottery Scheduling”, which is a probabilistic scheduling algorithm for the processes in an operating system.

Processes are scheduled in a random manner. Lottery scheduling can be preemptive or non-preemptive, it is preemptive in our assignment. It also solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has a non-zero probability of being selected at each scheduling operation. In this scheduling every process has some tickets, and the scheduler picks a random ticket and process having that ticket is the winner, and it is executed for a time slice and then another ticket is picked by the scheduler. These tickets represent the share of processes. A process having a higher number of tickets give it more chance to get chosen for execution.

- Advantages of Lottery Process Scheduling:

1. Fairness: Lottery process scheduling is designed to be fair because every process has a chance of being selected. This ensures that no process is starved of CPU time, and it also helps to prevent monopolization of the CPU by a single process.
2. Flexibility: The lottery process scheduling algorithm is highly flexible and can be easily adapted to handle various scheduling requirements, such as real-time scheduling, multi-processor scheduling, and so on.
3. Simple implementation: The implementation of the lottery process scheduling algorithm is relatively simple compared to other scheduling algorithms, making it easier to develop and maintain.
4. Combining with other scheduling algorithms: Lottery scheduling can be combined with other scheduling algorithms such as round-robin or priority scheduling to achieve specific scheduling goals.

- Disadvantages of Lottery Process Scheduling:

1. Overhead: The lottery process scheduling algorithm requires additional overhead because it must generate lottery tickets for each process and

randomly select a winner. This overhead can be significant in systems with many processes.

2. Complexity: Although the implementation of the lottery process scheduling algorithm is relatively simple, the algorithm itself can be quite complex, especially when compared to other scheduling algorithms such as round-robin or FIFO.
3. Security: Lottery process scheduling is not a secure algorithm because it is based on randomness, which can be manipulated. An attacker could potentially gain an unfair advantage by manipulating the generation of lottery tickets, leading to security vulnerabilities.

## In-depth analysis and implementation

We declare an additional information in “proc.h” to indicate that how many tickets does the process have.

```
#if DEBUG_TRACE
    int p_schedules;
#endif

/* Lottery Scheduling */
int numTickets;
```

The file “proc.c” will include the file “proc.h”, which means that it will use the information in “proc.h”.

First, we will initialize the “numTickets” in every process to 5 in the function called “proc\_init”.

```

void proc_init(void)
{
    struct proc * rp;
    struct priv *sp;
    int i;

    /* Clear the process table. Announce each slot as empty and set up
     * mappings for proc_addr() and proc_nr() macros. Do the same for the
     * table with privilege structures for the system processes.
     */
    for (rp = BEG_PROC_ADDR, i = -NR_TASKS; rp < END_PROC_ADDR; ++rp, ++i) {
        rp->p_rts_flags = RTS_SLOT_FREE; /* initialize free slot */
        rp->p_magic = PMAGIC;
        rp->p_nr = i; /* proc number from ptr */
        rp->p_endpoint = _ENDPOINT(0, rp->p_nr); /* generation no. 0 */
        rp->p_scheduler = NULL; /* no user space scheduler */
        rp->p_priority = 0; /* no priority */
        rp->p_quantum_size_ms = 0; /* no quantum size */

        /* Lottery Scheduling */
        rp->numTickets = 5;

        /* arch-specific initialization */
        arch_proc_reset(rp);
    }
}

```

Next, in the function called “pick\_proc”, which returns the process that will run next round.

```

/*=====
 * pick_proc
 *=====*/
/* Lottery Scheduling */
static struct proc * pick_proc(void)
{
    register struct proc *rp; /* process to run */
    struct proc **rdy_head;
    int q; /* iterate over queues */
    int sum;
    total_tickets = 0;
    int winner;
    srandom(0);

    rdy_head = get_cpulocal_var(run_q_head);
    /* Check if the process has use all the time quantum or not */
    for(q = 0; q < NR_SCHED_QUEUES; q++) {
        if(!(rp = rdy_head[q])) {
            continue;
        }
        assert(proc_is_runnable(rp));
        if (rp->p_rts_flags & RTS_NO_QUANTUM) { // decrease tickets
            ProcessSetPriority(rp, DECREASE_TICKETS);
        }
        else{
            ProcessSetPriority(rp, INCREASE_TICKETS); // increase tickets
        }
    }
}

```

```

/* System process has the higher priority */
for (q=0; q < 6; q++) {
    if(! (rp = rdy_head[q])) {
        TRACE(VF_PICKPROC, printf("cpu %d queue %d empty\n", cpuid, q));
        continue;
    }
    assert(proc_is_runnable(rp));
    if (priv(rp)->s_flags & BILLABLE)
        get_cpulocal_var(bill_ptr) = rp; /* bill for system time */
    return rp;
}

/* Do the Lottery Scheduling in User process*/
for(q = 6; q < NR_SCHED_QUEUES; q++) {
    if(! (rp = rdy_head[q])) {
        TRACE(VF_PICKPROC, printf("cpu %d queue %d empty\n", cpuid, q));
        continue;
    }

    while (rp != NULL) {
        if(rp->p_rts_flags == 0){
            total_tickets += rp->numTickets;
        }
        rp = rp->p_nextready;
    }
}

```

```

if(total_tickets > 0) {
    winner = random() % total_tickets;
    for(q = 6; q < NR_SCHED_QUEUES; q++){
        rp = rdy_head[q];
        while (rp != NULL) {
            if (rp->p_rts_flags == 0){
                winner -= rp->numTickets;
                if(winner < 0){
                    TRACE(VF_PICKPROC, printf("found %s / %d on queue %d\n",
                        rp->p_name, rp->p_endpoint, q));

                    assert(proc_is_runnable(rp));
                    if (priv(rp)->s_flags & BILLABLE){
                        get_cpulocal_var(bill_ptr) = rp; /* bill for system time */
                    }
                    return rp;
                }
            }
            rp = rp->p_nextready;
        }
    }
}
return NULL;
}

```

Notice that it appears a new function called “ProcessSetPriority”, which accounts for dynamically adjust the “numTickets” by the time quantum information. Also, it will check if the “numTickets” of the process is more than 10 or less than 1.

```

/*=====
 *          ProcessSetPriority          *
 *=====*/
/* Lottery Scheduling */
void ProcessSetPriority(register struct proc *rp, int ntickets){

    /* If ntickets is positive, add that number into numTickets*/
    if(ntickets > 0){
        /*makes sure that the process can't have over MAX_TICKETS tickets.*/
        if((MAX_TICKETS - rp->numTickets) < ntickets || rp->numTickets == MAX_TICKETS){
            rp->numTickets = MAX_TICKETS;
        }

        else{
            /*Add nTickets to the process's tickets*/
            rp->numTickets = rp->numTickets + ntickets;
            /*Increase the total number of tickets*/
            total_tickets = total_tickets + ntickets;
        }
    }
}

```

```

/*If ntickets is negative, subtract that number of tickets*/
else{
    /*Reverse ntickets, for easiness*/
    ntickets = ntickets * -1;

    /*Ensure that the number of tickets is greater than MIN_TICKETS always*/
    if(rp->numTickets < ntickets || rp->numTickets == MIN_TICKETS){
        rp->numTickets = MIN_TICKETS;
    }

    else{
        /*Subtract ntickets from the process's tickets*/
        rp->numTickets = rp->numTickets - ntickets;
        /*Decrement the total number of tickets*/
        total_tickets = total_tickets - ntickets;
    }
}
}

```

Notice that there are two ways to calculate the “total\_tickets” in the queue. First method is calculated in the function “pick\_proc” every time when it is called, another method is to add the “total\_tickets” in the function called “enqueue” in “proc.c” and decrease the “total\_tickets”. In my assignment, I choose the first method to calculate how many “total\_tickets” are there in the queue.



```

/* Do the Lottery Scheduling in User process*/
for(q = 6; q < NR_SCHED_QUEUES; q++) {
    if(!(rp = rdy_head[q])) {
        TRACE(VF_PICKPROC, printf("cpu %d queue %d empty\n", cpuid, q));
        continue;
    }

    /* Calculate the total tickets in the ready queue */
    while (rp != NULL) {
        if(rp->p_rts_flags == 0){
            total_tickets += rp->numTickets;
        }
        rp = rp->p_nextready;
    }
}

```

## The existing functions to be modified

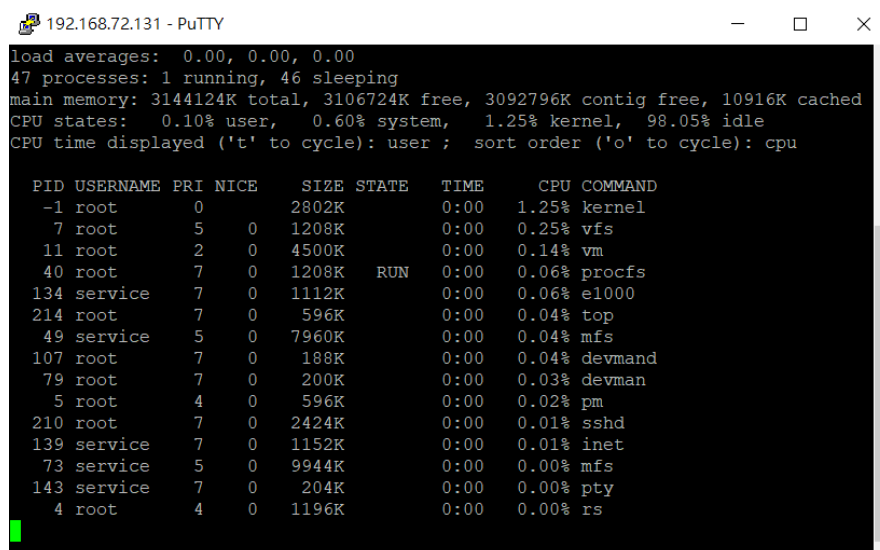
We modify the existing function called “pick\_proc”, and append a new function called “ProcessSetPriority”.

## Test plan

We use “top” to check the performance of the kernel and the running process, which means that we check how many CPU utilization a process and the kernel has.

## Original CPU Scheduling

When we input the command called “top”, it shows the detail performance of the computer.



```

192.168.72.131 - PuTTY
load averages:  0.00, 0.00, 0.00
47 processes: 1 running, 46 sleeping
main memory: 3144124K total, 3106724K free, 3092796K contig free, 10916K cached
CPU states:  0.10% user,  0.60% system,  1.25% kernel,  98.05% idle
CPU time displayed ('t' to cycle): user ;  sort order ('o' to cycle): cpu

  PID USERNAME  PRI NICE   SIZE STATE   TIME    CPU COMMAND
   -1 root        0     0   2802K      0:00  1.25% kernel
    7 root        5     0   1208K      0:00  0.25% vfs
   11 root        2     0   4500K      0:00  0.14% vm
   40 root        7     0   1208K   RUN   0:00  0.06% procfs
  134 service    7     0   1112K      0:00  0.06% e1000
  214 root        7     0    596K      0:00  0.04% top
   49 service    5     0   7960K      0:00  0.04% mfs
  107 root        7     0    188K      0:00  0.04% devmand
   79 root        7     0    200K      0:00  0.03% devman
    5 root        4     0    596K      0:00  0.02% pm
  210 root        7     0   2424K      0:00  0.01% sshd
  139 service    7     0   1152K      0:00  0.01% inet
   73 service    5     0   9944K      0:00  0.00% mfs
  143 service    7     0    204K      0:00  0.00% pty
    4 root        4     0   1196K      0:00  0.00% rs

```

To enhance the high CPU usage, I use the code from this website.

#####

<https://cobweb.cs.uga.edu/~maria/classes/4730-Fall-2009/project4mufix-scheduler/longrun.c>

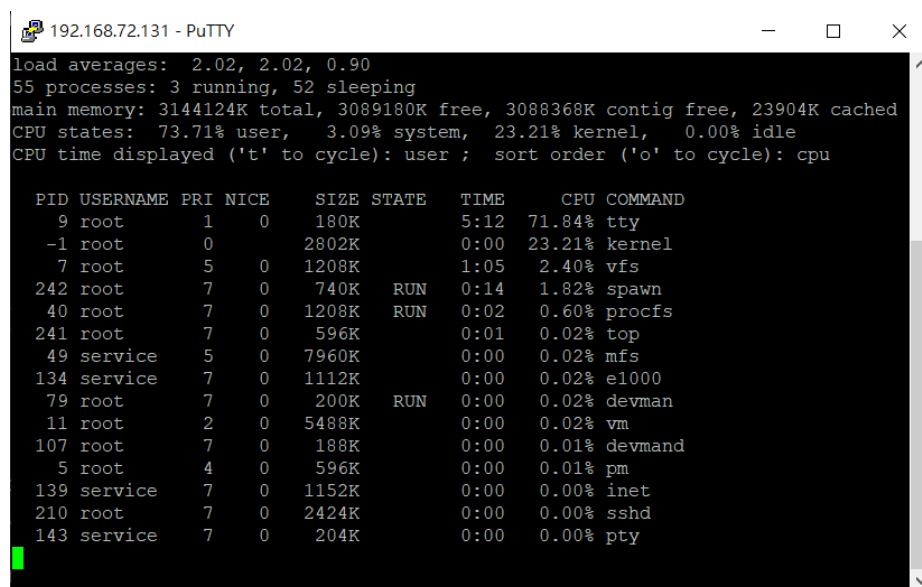
#####

In my directory, it is named “longrun.c”. You can put it in the folder wherever you want.

To run this code, please to do the following steps:

1. `clang longrun.c -o longrun`
2. `./longrun 0 100 10000000`

After running the “longrun.c”, the usage of CPU is as following:



```
192.168.72.131 - PuTTY
load averages:  2.02, 2.02, 0.90
55 processes: 3 running, 52 sleeping
main memory: 3144124K total, 3089180K free, 3088368K contig free, 23904K cached
CPU states: 73.71% user,  3.09% system, 23.21% kernel,  0.00% idle
CPU time displayed ('t' to cycle): user ;  sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE  SIZE  STATE  TIME   CPU COMMAND
    9 root      1    0   180K      5:12  71.84% tty
   -1 root      0      2802K      0:00  23.21% kernel
    7 root      5    0  1208K      1:05   2.40% vfs
  242 root      7    0   740K    RUN   0:14   1.82% spawn
   40 root      7    0  1208K    RUN   0:02   0.60% procfs
  241 root      7    0   596K      0:01   0.02% top
   49 service   5    0  7960K      0:00   0.02% mfs
  134 service   7    0  1112K      0:00   0.02% e1000
   79 root      7    0   200K    RUN   0:00   0.02% devman
   11 root      2    0  5488K      0:00   0.02% vm
  107 root      7    0   188K      0:00   0.01% devmand
    5 root      4    0   596K      0:00   0.01% pm
  139 service   7    0  1152K      0:00   0.00% inet
  210 root      7    0  2424K      0:00   0.00% sshd
  143 service   7    0   204K      0:00   0.00% pty
```

Generate kernel use statistics (100 times)

The mean of CPU that the kernel use is 25.4945% ~= 25.49%

## Lottery Scheduling

When we input the command called “top”, it shows the detail performance of the computer.

```
192.168.72.131 - PuTTY
load averages: 0.00, 0.96, 0.31
47 processes: 1 running, 46 sleeping
main memory: 3144124K total, 3106744K free, 3092792K contig free, 10880K cached
CPU states: 0.09% user, 0.54% system, 0.94% kernel, 98.43% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE   SIZE STATE   TIME    CPU COMMAND
    -1 root      0     0   2806K      0:00 0.94% kernel
     7 root      5     0   1204K      0:22 0.22% vfs
    11 root      2     0   4512K      0:00 0.16% vm
    40 root      7     0   1208K      0:00 0.07% procfs
   215 root      7     0    596K      0:00 0.05% top
    49 service    5     0   7960K      0:00 0.04% mfs
   107 root      7     0    188K      0:00 0.03% devmand
    79 root      7     0    200K      0:00 0.02% devman
   134 service    7     0   1112K      0:00 0.02% e1000
     5 root      4     0    596K      0:00 0.01% pm
   139 service    7     0   1152K      0:00 0.00% inet
   211 root      7     0   2424K      0:00 0.00% sshd
   143 service    7     0    204K      0:00 0.00% pty
     4 root      4     0   1196K      0:00 0.00% rs
   173 root      7     0    212K      0:00 0.00% dhcpcd
```

After running the “longrun.c”, the usage of CPU is as following:

```
192.168.72.131 - PuTTY
load averages: 0.13, 0.97, 0.32
48 processes: 2 running, 46 sleeping
main memory: 3144124K total, 3106612K free, 3092792K contig free, 10884K cached
CPU states: 69.28% user, 3.14% system, 27.59% kernel, 0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE   SIZE STATE   TIME    CPU COMMAND
     9 root      1     0    180K      1:53 68.18% tty
    -1 root      0     0   2806K      0:00 27.59% kernel
     7 root      5     0   1204K      0:22 2.95% vfs
   217 root      7     0    720K      0:00 1.04% longrun
    11 root      2     0   4544K      0:00 0.08% vm
    40 root      7     0   1208K      0:00 0.03% procfs
    79 root      7     0    200K      0:00 0.03% devman
    49 service    5     0   7960K      0:00 0.03% mfs
   215 root      7     0    596K      0:00 0.03% top
   107 root      7     0    188K      0:00 0.02% devmand
     5 root      4     0    596K      0:00 0.01% pm
   134 service    7     0   1112K      0:00 0.00% e1000
    21 service    1     0     52K      0:00 0.00% pckbd
     4 root      4     0   1196K      0:00 0.00% rs
   139 service    7     0   1152K      0:00 0.00% inet
```

Generate kernel use statistics (100 times)

The mean of CPU that the kernel use is 25.40275% ≈ 25.40%

## Conclude the settings of the various parameters

There are several parameters in my assignment, they are as follows:

1. Time quantum: Don't change.
2. Maximum numTickets

3. Minimum numTickets
4. Decrease numTickets
5. Increase numTickets

```
/* Lottery Scheduling */
#define MAX_TICKETS 10
#define MIN_TICKETS 1
#define DECREASE_TICKETS -1
#define INCREASE_TICKETS 1
```

- Maximum numTickets = 10 & Minimum numTickets = 1:

192.168.72.131 - PuTTY

```
load averages: 0.57, 0.12, 0.04
48 processes: 4 running, 44 sleeping
main memory: 3144124K total, 3106612K free, 3092792K contig free, 10888K cached
CPU states: 72.28% user, 2.59% system, 25.14% kernel, 0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu
```

PID	USERNAME	PRI	NICE	SIZE	STATE	TIME	CPU	COMMAND
9	root	1	0	180K		0:08	71.80%	tty
-1	root	0		2806K		0:00	25.14%	kernel
7	root	5	0	1204K		0:01	2.49%	vfs
216	root	7	0	720K	RUN	0:00	0.44%	longrun
11	root	2	0	4540K		0:00	0.04%	vm
40	root	7	0	1208K	RUN	0:00	0.03%	procfs
214	root	7	0	596K		0:00	0.02%	top
49	service	5	0	7960K		0:00	0.01%	mfs
107	root	7	0	188K	RUN	0:00	0.01%	devmand
79	root	7	0	200K	RUN	0:00	0.01%	devman
5	root	4	0	596K		0:00	0.01%	pm
134	service	7	0	1112K		0:00	0.00%	e1000
143	service	7	0	204K		0:00	0.00%	pty
210	root	7	0	2424K		0:00	0.00%	sshd
139	service	7	0	1152K		0:00	0.00%	inet

- Maximum numTickets = 100 & Minimum numTickets = 1:

192.168.72.131 - PuTTY

```
load averages: 0.93, 0.18, 0.06
48 processes: 4 running, 44 sleeping
main memory: 3144124K total, 3106620K free, 3092792K contig free, 10880K cached
CPU states: 69.78% user, 3.59% system, 26.64% kernel, 0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu
```

PID	USERNAME	PRI	NICE	SIZE	STATE	TIME	CPU	COMMAND
9	root	1	0	180K		0:20	69.02%	tty
-1	root	0		2806K		0:00	26.64%	kernel
7	root	5	0	1212K		0:03	2.85%	vfs
210	root	7	0	580K	RUN	0:00	0.71%	longrun
40	root	7	0	1208K	RUN	0:00	0.54%	procfs
11	root	2	0	4532K		0:00	0.13%	vm
215	root	7	0	596K		0:00	0.03%	top
49	service	5	0	7960K		0:00	0.03%	mfs
79	root	7	0	200K	RUN	0:00	0.02%	devman
107	root	7	0	188K		0:00	0.01%	devmand
134	service	7	0	1112K		0:00	0.01%	e1000
5	root	4	0	596K		0:00	0.01%	pm
4	root	4	0	1196K		0:00	0.00%	rs
139	service	7	0	1152K		0:00	0.00%	inet
211	root	7	0	2424K		0:00	0.00%	sshd

- Maximum numTickets = 1000 & Minimum numTickets = 1:

```

192.168.72.131 - PuTTY
load averages:  4.43, 1.10, 0.36
48 processes: 3 running, 45 sleeping
main memory: 3144124K total, 3106540K free, 3092792K contig free, 10944K cached
CPU states: 70.59% user,  3.95% system, 25.46% kernel,  0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE  SIZE STATE   TIME    CPU COMMAND
    9 root      1   0    180K      1:00 69.44% tty
   -1 root      0      2806K      0:00 25.46% kernel
    7 root      5   0    1208K      0:10  3.86% vfs
  215 root      7   0     720K    RUN  0:03  1.12% longrun
   11 root      2   0    4540K      0:00  0.05% vm
   40 root      7   0    1208K    RUN  0:00  0.03% procfs
  216 root      7   0     596K      0:00  0.02% top
   49 service   5   0    7960K      0:00  0.00% mfs
  134 service   7   0    1112K      0:00  0.00% e1000
    5 root      4   0     596K      0:00  0.00% pm
  208 root      7   0    2424K      0:00  0.00% sshd
  143 service   7   0     204K      0:00  0.00% pty
  139 service   7   0    1152K      0:00  0.00% inet
    4 root      4   0    1204K      0:00  0.00% rs
   79 root      7   0     200K    RUN  0:00  0.00% devman

```

- Maximum numTickets = 10 & Minimum numTickets = 5:

```

192.168.72.131 - PuTTY
load averages:  2.80, 0.55, 0.18
48 processes: 4 running, 44 sleeping
main memory: 3144124K total, 3106608K free, 3092792K contig free, 10888K cached
CPU states: 70.03% user,  3.10% system, 26.87% kernel,  0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE  SIZE STATE   TIME    CPU COMMAND
    9 root      1   0     180K      0:43 69.24% tty
   -1 root      0      2806K      0:00 26.87% kernel
    7 root      5   0    1212K      0:07  2.98% vfs
  211 root      7   0     580K    RUN  0:02  0.74% longrun
   40 root      7   0    1208K    RUN  0:00  0.04% procfs
   11 root      2   0    4536K      0:00  0.04% vm
  216 root      7   0     596K      0:00  0.02% top
   49 service   5   0    7960K      0:00  0.02% mfs
  107 root      7   0     188K    RUN  0:00  0.01% devmand
   79 root      7   0     200K    RUN  0:00  0.01% devman
    5 root      4   0     596K      0:00  0.01% pm
  134 service   7   0    1112K      0:00  0.01% e1000
  139 service   7   0    1152K      0:00  0.00% inet
  212 root      7   0    2424K      0:00  0.00% sshd
    4 root      4   0    1196K      0:00  0.00% rs

```

- Maximum numTickets = 100 & Minimum numTickets = 5:

```

192.168.72.131 - PuTTY
load averages:  2.05, 0.46, 0.15
48 processes: 2 running, 46 sleeping
main memory: 3144124K total, 3106620K free, 3092792K contig free, 10880K cached
CPU states: 67.29% user,  3.52% system, 29.20% kernel,  0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE  SIZE STATE   TIME    CPU COMMAND
    9 root      1   0     180K      0:52 66.63% tty
   -1 root      0      2806K      0:00 29.20% kernel
    7 root      5   0    1208K      0:10  2.77% vfs
   40 root      7   0    1208K    RUN  0:00  0.66% procfs
  211 root      7   0     580K    RUN  0:02  0.61% longrun
   49 service   5   0    7960K      0:00  0.02% mfs
   11 root      2   0    4536K      0:00  0.02% vm
  216 root      7   0     596K      0:00  0.02% top
   79 root      7   0     200K      0:00  0.02% devman
  107 root      7   0     188K      0:00  0.02% devmand
    5 root      4   0     596K      0:00  0.01% pm
  134 service   7   0    1112K      0:00  0.01% e1000
  212 root      7   0    2424K      0:00  0.00% sshd
  139 service   7   0    1152K      0:00  0.00% inet
  143 service   7   0     204K      0:00  0.00% pty

```

- Maximum numTickets = 1000 & Minimum numTickets = 5:

```

192.168.72.131 - PuTTY
load averages: 3.20, 0.61, 0.20
48 processes: 3 running, 45 sleeping
main memory: 3144124K total, 3106568K free, 3092792K contig free, 10932K cached
CPU states: 68.31% user, 3.41% system, 28.28% kernel, 0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE  SIZE STATE   TIME    CPU COMMAND
    9 root        1    0   180K      0:33 67.19% tty
   -1 root        0      2806K      0:00 28.28% kernel
    7 root        5    0  1208K      0:06 3.37% vfs
  215 root        7    0   720K    RUN   0:01 0.89% longrun
  210 root        7    0  2424K      0:00 0.22% sshd
  134 service     7    0  1112K      0:00 0.01% e1000
   11 root        2    0  4536K      0:00 0.01% vm
   40 root        7    0  1208K    RUN   0:00 0.01% procfs
  214 root        7    0   596K      0:00 0.01% top
  139 service     7    0  1152K      0:00 0.00% inet
   49 service     5    0  7960K      0:00 0.00% mfs
    5 root        4    0   596K      0:00 0.00% pm
    4 root        4    0  1196K      0:00 0.00% rs
  107 root        7    0   188K    RUN   0:00 0.00% devmand
   79 root        7    0   200K      0:00 0.00% devman

```

- Maximum numTickets = 100 & Minimum numTickets = 1 & Increase numTickets 10 & Decrease numTickets 5:

```

192.168.72.131 - PuTTY
load averages: 0.49, 0.09, 0.03
48 processes: 5 running, 43 sleeping
main memory: 3144124K total, 3106568K free, 3092792K contig free, 10932K cached
CPU states: 70.08% user, 3.35% system, 26.56% kernel, 0.00% idle
CPU time displayed ('t' to cycle): user ; sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE  SIZE STATE   TIME    CPU COMMAND
    9 root        1    0   180K      0:09 69.42% tty
   -1 root        0      2806K      0:00 26.56% kernel
    7 root        5    0  1208K      0:02 2.65% vfs
  215 root        7    0   720K    RUN   0:00 0.63% longrun
   40 root        7    0  1208K    RUN   0:00 0.58% procfs
   11 root        2    0  4536K      0:00 0.07% vm
   49 service     5    0  7960K      0:00 0.02% mfs
  214 root        7    0   596K      0:00 0.02% top
   79 root        7    0   200K    RUN   0:00 0.01% devman
  107 root        7    0   188K    RUN   0:00 0.01% devmand
  134 service     7    0  1112K      0:00 0.01% e1000
    5 root        4    0   596K      0:00 0.01% pm
  210 root        7    0  2424K      0:00 0.00% sshd
  139 service     7    0  1152K      0:00 0.00% inet
    4 root        4    0  1196K      0:00 0.00% rs

```

- Maximum numTickets = 1000 & Minimum numTickets = 1 & Increase numTickets 100 & Decrease numTickets 50:

```
192.168.72.131 - PuTTY
load averages:  1.50, 0.29, 0.09
48 processes: 2 running, 46 sleeping
main memory: 3144124K total, 3106612K free, 3092792K contig free, 10884K cached
CPU states: 67.13% user,  2.73% system, 30.14% kernel,  0.00% idle
CPU time displayed ('t' to cycle): user ;  sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE   SIZE STATE   TIME    CPU COMMAND
    9 root      1  0    180K      0:18 66.81% tty
   -1 root      0      2806K      0:00 30.14% kernel
    7 root      5  0    1204K      0:03  2.57% vfs
  215 root      7  0    720K   RUN  0:00  0.29% longrun
   11 root      2  0   4536K      0:00  0.07% vm
   40 root      7  0   1208K   RUN  0:00  0.03% procfs
  214 root      7  0    596K      0:00  0.02% top
  134 service    7  0   1112K      0:00  0.02% e1000
   79 root      7  0    200K      0:00  0.01% devman
   49 service    5  0   7960K      0:00  0.01% mfs
   73 service    5  0   9896K      0:00  0.01% mfs
   32 service    7  0    188K      0:00  0.01% at_wini
  107 root      7  0    188K      0:00  0.00% devmand
    5 root      4  0    596K      0:00  0.00% pm
  139 service    7  0   1152K      0:00  0.00% inet
```

- Maximum numTickets = 1000 & Minimum numTickets = 1 & Increase numTickets 10 & Decrease numTickets 50:

```
192.168.72.131 - PuTTY
load averages:  0.43, 0.08, 0.02
48 processes: 4 running, 44 sleeping
main memory: 3144124K total, 3106624K free, 3092792K contig free, 10880K cached
CPU states: 72.14% user,  3.56% system, 24.29% kernel,  0.00% idle
CPU time displayed ('t' to cycle): user ;  sort order ('o' to cycle): cpu

  PID USERNAME PRI NICE   SIZE STATE   TIME    CPU COMMAND
    9 root      1  0    180K      0:05 70.88% tty
   -1 root      0      2806K      0:00 24.29% kernel
    7 root      5  0    1204K      0:01  3.47% vfs
  215 root      7  0    720K   RUN  0:00  1.25% longrun
   11 root      2  0   4536K      0:00  0.05% vm
   40 root      7  0   1208K   RUN  0:00  0.02% procfs
  134 service    7  0   1112K      0:00  0.01% e1000
  214 root      7  0    596K      0:00  0.01% top
   49 service    5  0   7960K      0:00  0.00% mfs
    5 root      4  0    596K      0:00  0.00% pm
  139 service    7  0   1152K      0:00  0.00% inet
  208 root      7  0   2424K      0:00  0.00% sshd
  107 root      7  0    188K      0:00  0.00% devmand
   79 root      7  0    200K   RUN  0:00  0.00% devman
  143 service    7  0    204K   RUN  0:00  0.00% pty
```

If I set the MAX\_TICKETS to about 100 or 1000, every time “pick\_proc” is called, it will calculate the “total\_tickets”, so the number is bigger and bigger, and PuTTY, which uses ssh to connect to the minix OS, will slow down obviously. The followings are the impact of these parameters:

#### 1. Impact of Maximum numTickets and Minimum numTickets

- ◆ Range 10 to 1,000 for Maximum numTickets and 1 to 5 for Minimum numTickets: These settings determine the range of tickets that any process can hold. A wider range (e.g., 1 to 1000) offers greater flexibility and can be used to prioritize processes more distinctly. A narrower range (e.g., 5 to 10) limits this flexibility, potentially making the scheduling less responsive to changes in

process priority or workload characteristics.

## 2. Impact of Increase numTickets and Decrease numTickets

- ◆ Fixed increments and decrements (e.g., 10 and 5, 100 and 50):  
These settings influence how quickly a process can gain or lose priority. Larger increments and decrements allow faster adaptation to changing conditions but can lead to instability or "thrashing" where processes frequently oscillate between high and low priority. Smaller values provide smoother but slower adjustments.

## 3. Operational and Performance Considerations

- ◆ When Maximum numTickets is set to higher values (100 or 1000):  
This leads to a significant increase in total\_tickets computation as observed with the slowdown in PuTTY via SSH. This implies that high values can cause overhead due to more complex calculations and larger numbers to manage, impacting system performance, especially in environments with limited processing power or during high I/O operations.

# Compare with Linux

CFS (Completely Fair Scheduler) 是 Linux 操作系統中的進程調度器，其設計原理可以總結為模擬一個“理想的、精確的多任務 CPU”在真實硬件上的表現。CFS 不使用隊列（queue）作為數據結構，而是採用紅黑樹（red-black tree）。這種結構利用每個任務的虛擬運行時間（virtual runtime）來組織樹，並在每次調度時選擇具有最小虛擬運行時間的任務來執行，從而達到公平性。在 CFS 中，每個進程的運行時間被記錄在進程描述符的 `p->se.vruntime` 中，調度器則根據紅黑樹中的最小虛擬運行時間來選擇下一個要運行的進程，以確保每個進程都有公平的 CPU 時間分配。

input the command called “top”, it shows the detail performance of the computer.



```
top - 16:27:02 up 2 min, 1 user, load average: 1.50, 0.78, 0.30
Tasks: 325 total, 1 running, 324 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.5 us, 0.3 sy, 0.0 ni, 99.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3870.6 total, 1973.9 free, 968.1 used, 928.7 buff/cache
MiB Swap: 3898.0 total, 3898.0 free, 0.0 used. 2621.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1238	owen	20	0	4483120	280720	143244	S	0.7	7.1	0:09.83	gnome-shell
1936	owen	20	0	650788	71388	55228	S	0.7	1.8	0:21.35	gnome-terminal-
593	root	20	0	244272	8956	7676	S	0.3	0.2	0:00.64	vmtoolsd
1612	owen	20	0	141160	37920	29600	S	0.3	1.0	0:00.87	vmtoolsd
2079	owen	20	0	13740	4224	3328	R	0.3	0.1	0:00.25	top
1	root	20	0	101156	11772	8316	S	0.0	0.3	0:01.30	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_flushwq
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
7	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0-events
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-ttm
9	root	20	0	0	0	0	I	0.0	0.0	0:00.05	kworker/0:1-events
10	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/u256:0-events_unbound
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
12	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_kthread
13	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_rude_kthread
14	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_tasks_trace_kthread
15	root	20	0	0	0	0	S	0.0	0.0	0:00.11	ksoftirqd/0
16	root	20	0	0	0	0	I	0.0	0.0	0:00.14	rcu_preempt
17	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
18	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	idle_inject/0
19	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
21	root	-51	0	0	0	0	S	0.0	0.0	0:00.00	idle_inject/1

After running the “longrun.c”, the usage of CPU is as following :

```
top - 16:36:21 up 11 min, 1 user, load average: 1.50, 0.85, 0.49
Tasks: 310 total, 2 running, 308 sleeping, 0 stopped, 0 zombie
%Cpu(s): 28.3 us, 60.4 sy, 0.0 ni, 11.0 id, 0.2 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3870.6 total, 243.2 free, 1712.1 used, 1915.3 buff/cache
MiB Swap: 3898.0 total, 3898.0 free, 0.0 used. 1823.3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4248	owen	20	0	2776	1536	1536	R	69.4	0.0	0:05.86	longrun
1936	owen	20	0	646288	67036	50748	S	67.4	1.7	1:01.29	gnome-t+
243	root	20	0	0	0	0	I	29.6	0.0	0:15.01	kworker+
1238	owen	20	0	4540964	322784	173404	S	4.0	8.1	0:32.77	gnome-s+
240	root	20	0	0	0	0	I	3.0	0.0	0:04.91	kworker+
1404	owen	20	0	339056	15616	13568	S	0.3	0.4	0:00.16	goa-ide+
1612	owen	20	0	141436	38176	29600	S	0.3	1.0	0:01.89	vmtoolsd
2079	owen	20	0	13740	4224	3328	R	0.3	0.1	0:01.99	top
1	root	20	0	167988	13180	8316	S	0.0	0.3	0:01.92	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par+
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_fl+
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.11	kworker+
9	root	20	0	0	0	0	I	0.0	0.0	0:00.18	kworker+
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_perc+

To compare Minix 3.3.0 to Linux, particularly in terms of scheduling and system performance for I/O-bound and CPU-bound applications, we need to look at

their respective scheduling strategies, kernel architectures, and design principles.

### 1. Kernel Architecture:

- Minix 3.3.0: Minix uses a microkernel architecture which aims to keep the core functionality of the kernel minimal and executes most of the operating system services like drivers and file systems in user space. This design can enhance the system's reliability and make it easier to manage and secure, but it might introduce overheads in performance due to the increased context switches between user and kernel mode.
- Linux: Linux uses a monolithic kernel architecture where the core system services such as device drivers, file system management, and network stacks run in the kernel space. This can potentially offer better performance for CPU-bound applications due to fewer context switches.

### 2. Scheduling:

- Minix 3.3.0: Minix used a simple round-robin scheduling for processes within the multiple priority level. However, the focus in Minix is generally on simplicity and modularity, which might not always provide the best performance for more demanding computational tasks.
- Linux: Linux uses the Completely Fair Scheduler (CFS), which is highly sophisticated and aims to provide fair CPU time to runnable tasks using a red-black tree to dynamically calculate and adjust the virtual runtime of processes. This scheduler is very effective for both CPU-bound and I/O-bound workloads, balancing load and managing resources efficiently to optimize performance.

### 3. I/O and CPU-bound Applications:

- For I/O-bound applications, the microkernel architecture of Minix might incur some performance penalties due to the overhead of message passing and context switches. Linux, with its monolithic design and advanced I/O scheduling mechanisms like CFQ (Completely Fair Queuing), typically provides better throughput and responsiveness.
- For CPU-bound applications, the efficiency of Linux's scheduler in

managing and allocating CPU time shines, especially with recent improvements in load balancing across multiple CPUs and CPU cores. Minix, while capable, may lag behind in raw computational performance due to its simpler scheduler and the overhead introduced by its microkernel architecture.