Lottery scheduling

Lottery scheduling is a probabilistic scheduling algorithm for processes in an operating system. Processes are each assigned some **number of lottery tickets**, and the scheduler draws a **random** ticket to select the next process. The distribution of tickets need not be uniform; **granting a process more tickets provides it a relative higher chance of selection**. This technique can be used to approximate other scheduling algorithms, such as Shortest job next and Fair-share scheduling.

Implementation

1. Rename the parameter called deadline of the proc struct in /usr/src/kernel/proc.h and call it “tickets”

```
struct proc {
    struct stackframe_s p_reg; /* process' registers saved in stack frame
*/
    struct segframe p_seg; /* segment descriptors */
    proc_nr_t p_nr; /* number of this process (for fast access) */
    struct priv *p_priv; /* system privileges structure */
    volatile u32_t p_rts_flags; /* process is runnable only if zero */
    volatile u32_t p_misc_flags; /* flags that do not suspend the process
*/

    int tickets; /*current process, quantity of tickets*/

    char p_priority; /* current process priority */
    u64_t p_cpu_time_left; /* time left to use the cpu */
```

2. Modify the kernel call /usr/src/kernel/system/do_setedf.c :

```
#include "kernel/system.h"
#include <minix/endpoint.h>
int do_setedf(struct proc * caller, message * m_ptr){
    struct proc *p;
    int proc_nr = 0;
    if (!isokendpt(m_ptr->m1_i3, &proc_nr))
        return EINVAL;
    p = proc_addr(proc_nr);
    p->tickets = m_ptr->m1_i2;
    printf("do_setedf.c. %d\n",m_ptr->m1_i2);
    return(OK);
}
```

3. Create a global variable total_qty_tickets in the file /usr/src/kernel/glo.h **without initializing it**

```
extern struct kmessages kmessages;        /* diagnostic messages in
kernel */
extern struct loadinfo loadinfo;          /* status of load average */
extern struct minix_kerninfo minix_kerninfo;

EXTERN struct k_randomness krandom;     /* gather kernel random
information */
EXTERN int total_qty_tickets; /*quantity of tickets in all processes*/
vir_bytes minix_kerninfo_user;

#define kmess kmessages
#define kloadinfo loadinfo
```

4. En este punto ir a /usr/src/releasetools y ejecutar los comandos
   make services
   make install
   make hdboot
   Reiniciar la maquina y entrar en la opcion de minix por defecto (2)
5. Initialize the global variable **total_qty_tickets** in the file /usr/src/kernel/glo.h

```
extern struct kmessages kmessages;        /* diagnostic messages in
kernel */nJW =
extern struct loadinfo loadinfo;          /* status of load average */
extern struct minix_kerninfo minix_kerninfo;

EXTERN struct k_randomness krandom;     /* gather kernel random
information */
EXTERN int total_qty_tickets=0; /*quantity of tickets in all
processes*/
vir_bytes minix_kerninfo_user;

#define kmess kmessages
#define kloadinfo loadinfo
```

6. Modify the enqueue function (/usr/src/kernel/proc.c void enqueue) to accumulate the quantity of tickets:

```
assert(proc_is_runnable(rp));
assert(q >= 0);
rdy_head = get_cpu_var(rp->p_cpu, run_q_head);
rdy_tail = get_cpu_var(rp->p_cpu, run_q_tail);
if(rp->tickets>0)
    total_qty_tickets=total_qty_tickets+(rp->tickets);
```

7. Add the process to the queue sorted in **ascending** order (the process with fewer tickets at the head)

```
. . . .
/* Now add the process to the queue. */
if (!rdy_head[q]) { /* add to empty queue */
    rdy_head[q] = rdy_tail[q] = rp; /* create a new queue */
    rp->p_nextready = NULL; /* mark new end */
}else { /* add to tail of queue */
    struct proc *start = rdy_head[q];
    //Case that needs to be added to the head
    if(rdy_head[q]->tickets > rp->tickets){
        rp->p_nextready = rdy_head[q];
        rdy_head[q] = rp;
    }else{//Find position to insert
       // COMPLETE THE IMPLEMENTATION
    }
}
. . . .
```

Hint:
-At this point take a **snapshot** of the machine.
-**Compile and reboot** the machine to check if there is any Error. (/usr/src/releasetools: make services, make hdboot...)
-Execute **test_lottery_a.c** to be sure that processes are finishing in ascending order of tickets.

8. To complete the lottery algorithm is necessary to generate random numbers, which is going to be the winner ticket (the owner of the ticket is the process to be execute).
   Implement the following function to **generate random numbers** (at the beginning of the file /usr/src/kernel/proc.c)

```
static void idle(void);
unsigned short lfsr = 0xACE1u;
unsigned bit;

unsigned rand()
{
    bit  = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 5) ) & 1;
    return lfsr =  (lfsr >> 1) | (bit << 15);
}
/**
 * Made public for use in clock.c (for user-space scheduling)
   static int mini_send(struct proc *caller_ptr, endpoint_t dst_e,
message
   *m_ptr, int flags);
*/
```

Note: Why stdlib.h can't be imported in proc.c to use its rand function?

9. Modify the **pick_proc** function to select a process based on the lottery and not just the head of the queue.

```
/*===========================================================================*
 *                                pick_proc                                  *
 *===========================================================================*/
static struct proc * pick_proc(void)
{
/* Decide who to run now.  A new process is selected an returned.
 * When a billable process is selected, record it in 'bill_ptr', so that the
 * clock task can tell who to bill for system time.
 * This function always uses the run queues of the local cpu!
 */
```

Hint:

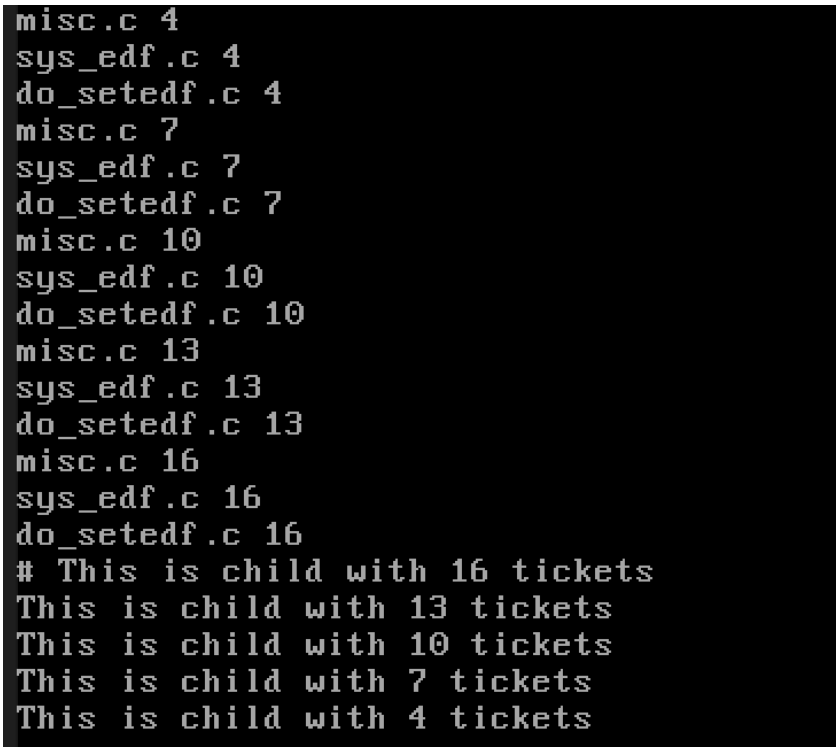
```
idProceso Ganador(){
 int suma = 0;
 int papeletaGanadora;
 idProceso proceso = preparados.primero;            /* se usa la cola de preparados */
 papeletaGanadora = Sorteo(numPapeletasTotales);          /* se realiza el sorteo */
 while ( suma < papeletaGanadora) {                    /* ¿quién tiene la papeleta? */
   suma = suma + tablaDescriptores[proceso].numPapeletas;
   if (suma < papeletaGanadora)
    proceso = tablaDescriptores[proceso].siguiente;
}
 return idProceso                                          /* Este proceso es el ganador */
}
```

Se saca el proceso ganador de la cola de preparados, se decrementa el número de papeletas que tiene asignadas del total.

```
   numPapeletasTotales = numPapeletasTotales - tablaDescriptores[proceso].numPapeletas;
```

- Sorteo(numPapeletasTotales) in our case is rand() % total_qty_tickets
- tablaDescriptores[proceso].siguiente in our case is rp->p_nextready
-return idProceso in our case is return rp
-Only apply the algorithm if(rp->tickets && rp->tickets>0)
-Remember to take a snapshot of the machine before compile the system.

Example of an output:



Since this is a lottery algorithm, the output could be different for each test