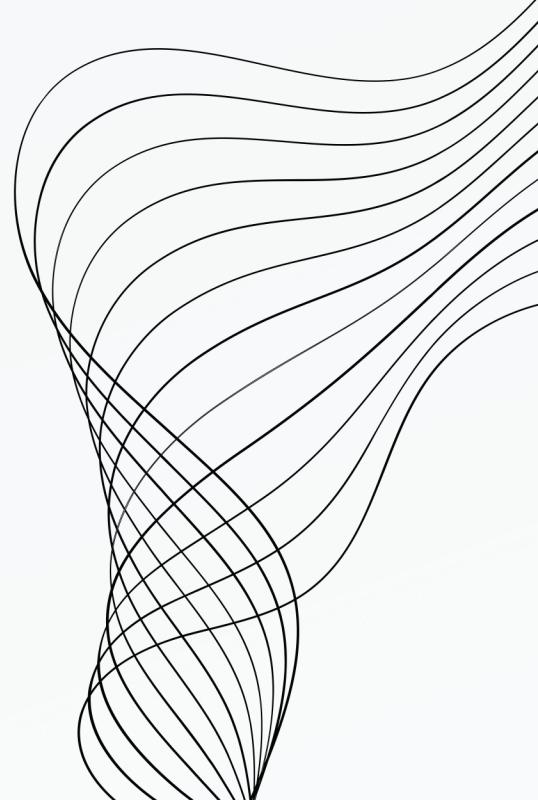


DISEÑO

**Javier Freire Otero
Adrián Perdiz Amoedo
Diego Pérez Martínez
Miguel Vila Rodríguez**



ÍNDICE

01

REQUISITOS NO FUNCIONALES

02

REQUISITOS FUNCIONALES

03

ALGORITMO

04

PRIORIDADES

05

ADELANTAMIENTOS

06

DIAGRAMAS DE FLUJO

07

PSEUDOCÓDIGO

08

MÉTRICAS

REQUISITOS NO FUNCIONALES

01

BASE DE DATOS MONOLÍTICA

02

SISTEMA DISTRIBUÍDO CON NÚMERO DINÁMICO DE NODOS

03

LOS NODOS SE COMUNICARÁN MEDIANTE PASO DE MENSAJES

04

EL SISTEMA NO SUFRIRÁ CAÍDAS Y NO SERÁ NECESARIO RECUPERAR EL ESTADO

05

LOS PROCESOS ESTARÁN EN SU SECCIÓN CRÍTICA UN TIEMPO FINITO

06

EL MANTENIMIENTO CONLLEVARÁ UNA PARADA TOTAL DEL SISTEMA

REQUISITOS FUNCIONALES

01

5 TIPOS DE PROCESOS: CONSULTAS,
RESERVAS, PAGOS, ADMINISTRACIÓN
Y ANULACIONES

02

LOS PROCESOS DE PAGOS Y
ANULACIONES SE EJECUTARÁN EN
EXCLUSIÓN MUTUA CON LOS
DEMÁS PROCESOS

03

LOS PROCESOS DE RESERVAS Y
ADMINISTRACIÓN SE EJECUTARÁN EN
EXCLUSIÓN MUTUA CON LOS DEMÁS
PROCESOS

04

PRIORIDADES:
1. ANULACIONES
2. PAGOS / ADMINISTRACIÓN
3. RESERVAS / CONSULTAS

05

ADELANTAMIENTOS: SÍ
• CUANDO HAYA N
CONSULTAS Y ENTRE UNA
PRIORITARIA, ADELANTARÁN

ALGORITMO

Tickets



Usaremos el algoritmo de Ricart-Agrawala(Tickets).

Cada uno tiene un número de ticket, el menor es el que entra

Más fácil poner N consultas a la vez(Lectores)
Más difícil hacer las prioridades.

Ventajas/Inconvenientes



PRIORIDADES

Prioridad #1

- Anulaciones

Prioridad #2

- Pagos
- Administración

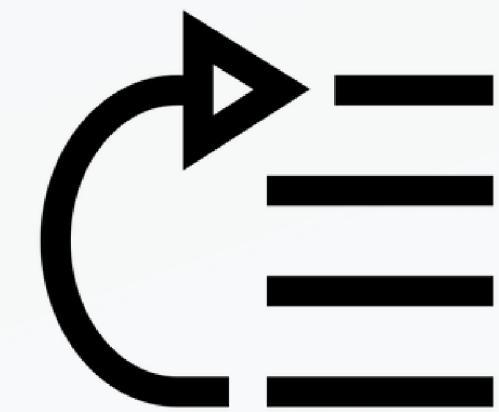
Prioridad #3

- Reservas
- Consultas

1. Anulaciones se consideran en el primer nivel puesto que son finitas y no deberían sufrir inanición en ningún caso.
2. En el segundo nivel de prioridad, tenemos pagos y administración. No ponemos pagos como más prioritario, ya que queremos minimizar problemas relacionados con que pasen todas las ventas antes que modificaciones/cancelaciones de eventos.
3. También consideramos reservas y consultas en un mismo nivel, porque no creemos que sea buena idea posponer consultas por una reserva que no nos asegura nada.

ADELANTAMIENTOS

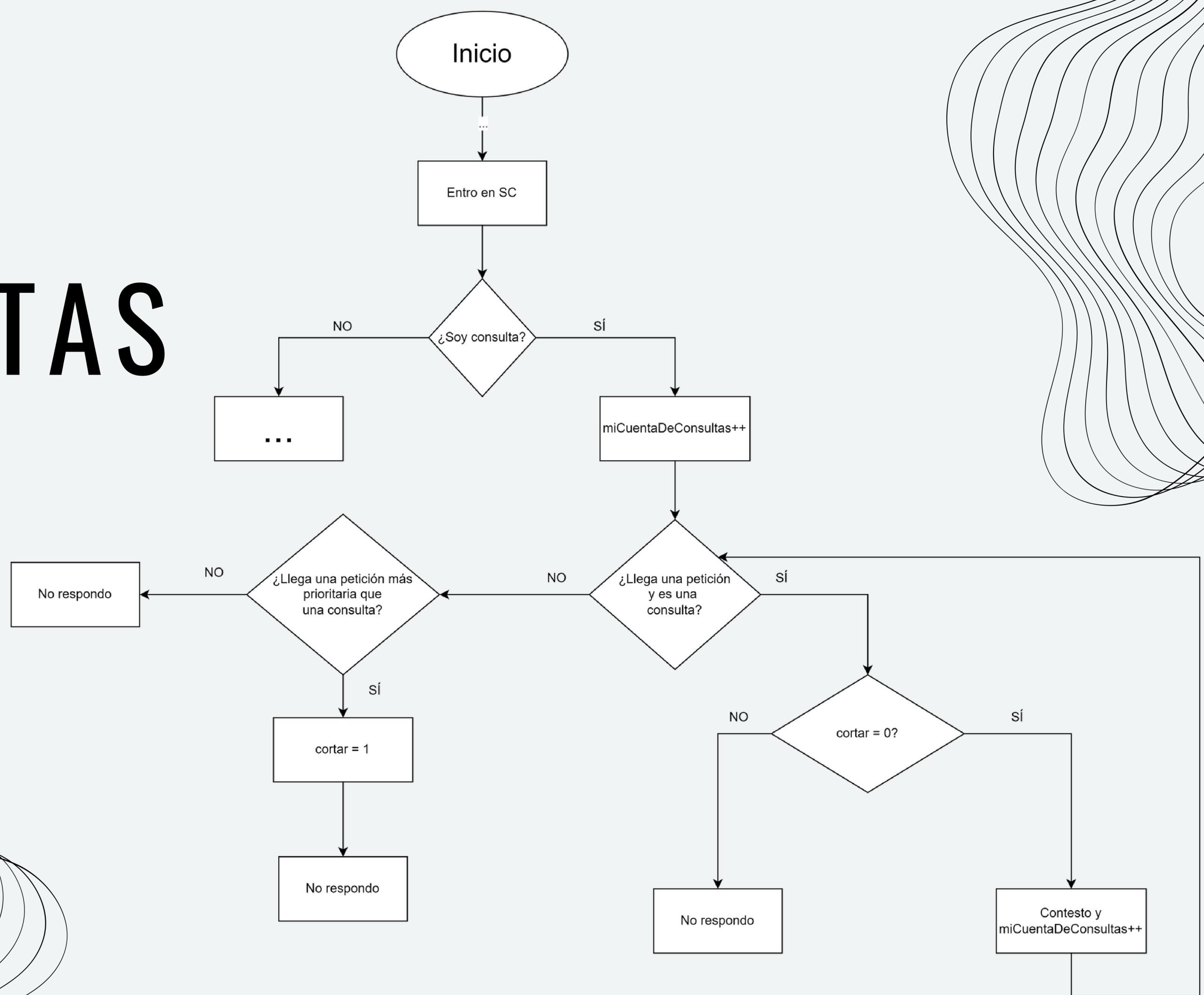
- Los adelantamientos se realizarán únicamente sobre las consultas.
- Si hay un número de consultas menor o igual que las N consultas máximas concurrentes y llega un proceso más prioritario, no se dejará pasar a las consultas a la SC para que entre este proceso.
- Si esto no se hiciera de esta forma, podrían seguir entrando nuevas consultas (hasta el n° máximo concurrentes) y pospondríamos indefinidamente el proceso prioritario.



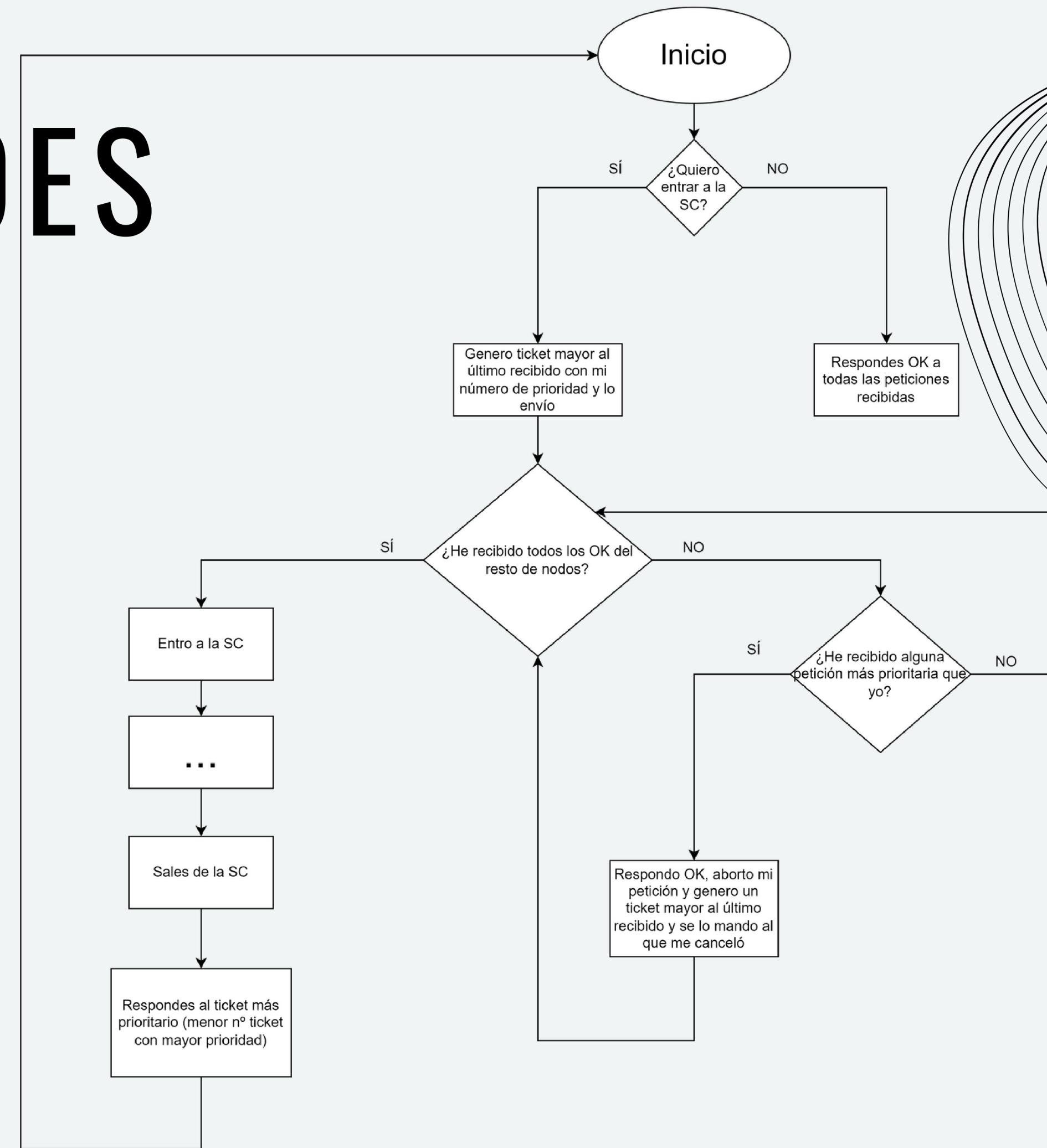
DIAGRAMAS DE FLUJO



N CONSULTAS



PRIORIDADES



PSEUDOCÓDIGO

```
int abortado = 0, id_nodos_pend[N-1] = {0},  
num_pend = 0, quiero = 0, max_ticket = 0,  
dentro = 0;  
int mi_id, mi_ticket, soyConsulta,  
mi_prioridad, miCuentaDeConsultas = 0;
```

MAIN

```
int id_aux = 0, i = 0, id_nodos[N-1] = {1235, 1236, ...};  
while (1) {  
    .....;  
    quiero = 1;  
    mi_ticket = max_ticket + 1;  
    for (i = 0; i < N-1; i++) { send(REQUEST, id_nodos[i], {mi_id, mi_ticket, mi_prioridad}); }  
    for (i = 0; i < N-1; i++) { receive(REPLY, &id_aux); } //recibo todos los OK  
    for (i = 0; i < abortado; i++) { receive(REPLY, &id_aux); }  
    //esperas el ok del que te había cancelado  
    dentro = 1;  
SECCIÓN CRÍTICA;  
    dentro = 0;  
    quiero = 0;  
    miCuentaDeConsultas = 0;  
    for (i = 0; i < num_pend; i++) { send(REPLY, id_nodos_pend[i], mi_id); }  
    num_pend = 0;  
}
```

RECEPTOR

```
int id_nodo_origen = 0, ticket_origen = 0;  
while (1) {  
    receive(REQUEST, {&id_nodo_origen, &ticket_origen});  
    max_ticket = MAX(max_ticket, ticket_origen);  
    if (NOT quiero) {  
        send(REPLY, id_nodo_origen, mi_id);  
        continue; }  
    if(ticket_origen.prioridad > mi_prioridad) {  
        if(NOT dentro) { abortar_mi_peticion(); }  
        if(soyConsulta AND dentro) { miCuentaDeConsultas = N-1; }  
    } else if(ticket_origen.prioridad = mi_ticket.prioridad AND  
              (ticket_origen < mi_ticket OR  
               (ticket_origen == mi_ticket AND (id_nodo_origen < mi_id))) {  
        send(REPLY, id_nodo_origen, mi_id);  
    }
```

RECEPTOR

```
} else if(!soyConsulta) id_nodos_pend[num_pend++] = id_nodo_origen;
if (dentro AND soyConsulta AND ticket_nodo_origen.soyConsulta
AND miCuentaDeConsultas < N-1) {
miCuentaDeConsultas++;
send(REPLY, id_nodo_origen, mi_id);
} else { // caso en el que se recibe consulta pero ya ha habido N-1
id_nodos_pend[num_pend++] = id_nodo_origen; }

abortar_mi_peticion(){
send(REPLY, id_nodo_origen, mi_id); // OK al que te canceló
mi_ticket = max_ticket + 1;
send(REQUEST, id_nodo_origen, {mi_id, mi_ticket, mi_prioridad}); //petición solo a ese
abortado ++; }
```

MISMO NODO:

Solo mandamos un REQUEST por cada prioridad, si llega otro proceso de la misma prioridad se queda esperando en un semáforo de paso.

Una vez tengamos los REPLY necesarios , le damos paso al proceso más prioritario del nodo, este entrará en la SC y al salir comprobará:

- Si no hay ningún otro nodo que quiera, le da paso a su nodo(si hay)
- Si hay esperando en otro nodo, seleccionamos al más prioritario priorizando nuestro nodo solo N veces, siendo N los ADELANTAMIENTOS



MÉTRICAS

- TIEMPO EFECTIVO EN SECCIÓN CRÍTICA: EL TIEMPO QUE ESTÁ EL PROCESO DENTRO DE LA MISMA
- TIEMPO DE ESPERA PROMEDIO PARA ACCEDER A LA SECCIÓN CRÍTICA: CUÁNTO TIEMPO TIENE QUE ESPERAR UN PROCESO, DE MEDIA, PARA ENTRAR EN LA SECCIÓN CRÍTICA
- NÚMERO DE MENSAJES ENVIADOS POR NODO

MÉTRICAS

