Algoritmo de Ricart-Agrawala - Solicitud de Turno

Este algoritmo se basa en el uso de un “ticket” pero sin utilizar un servidor de tickets asociado.

Para solicitar permiso, un proceso debe enviar un mensaje a todos los demás nodos usando su código de identificación. Hay dos posibles respuestas:

- Entra un Ticket previamente a querer entrar a la Sección Crítica: El proceso responde inmediatamente, aprobando su entrada.
- Entra un Ticket posteriormente a querer entrar a la Sección Crítica: El proceso pone en la cola la petición y responde después de hacer uso de la Sección Crítica.

Para poder entrar a la Sección Crítica, un proceso debe tener el permiso de todos los demás procesos.

Para cada nodo, tendremos dos métodos diferentes dentro del mismo proceso haciendo necesario el uso de *threads* para que el segundo pueda estar siempre activo:

- Main() →Gestiona la entrada y salida de la Sección Crítica del proceso y se ocupa de confirmar las peticiones después de salir de su Sección Crítica..
- Receive() → Procesa los mensajes recibidos, dando paso a su programa principal a la Sección Crítica o confirmando las peticiones de otros procesos.

Posibles problemas:

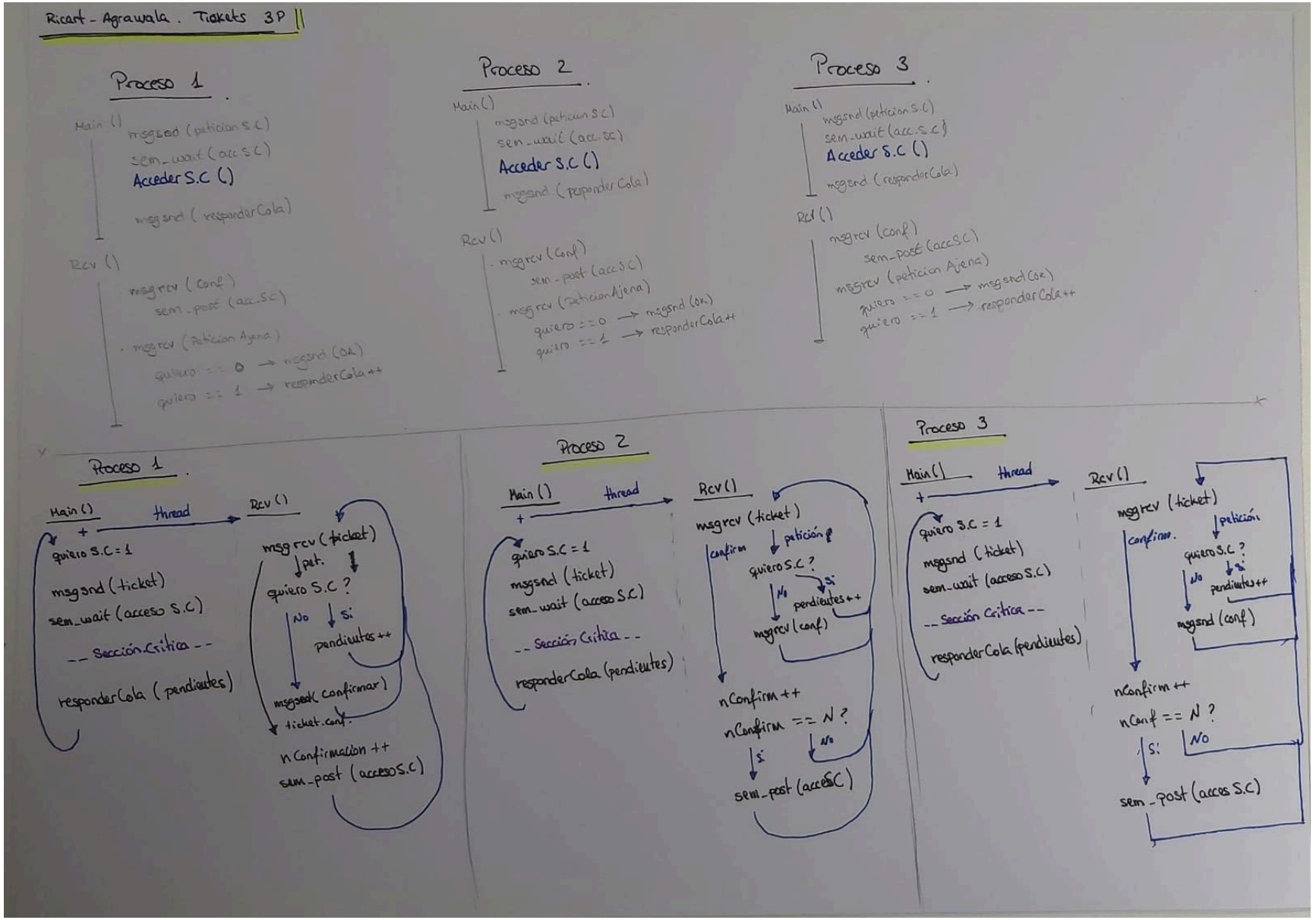
- ¿Qué pasa si dos procesos eligen el mismo ticket?

Debemos seleccionar un criterio de desempate. Por ejemplo el ID de proceso.

- ¿Qué pasa si un proceso “rápido” tiene tendencia a elegir números de ticket bajos?

Implementar en el método Receive() una variable que lleve la cuenta del número de ticket más alto y la actualice.

Propuesta implementación Tickets 3 procesos



Implementación

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <math.h>
#include <sys/msg.h>    // Msgget
#include <sys/ipc.h>    // Msgget
#include <pthread.h>    // Hilos
#include <semaphore.h>  // Semaforos


// Ricard-Agrawaka minimo
#define nUsuarios 3

typedef struct {

   int mtype;
   int idOrigen;
   float ticket;

} msg;


key_t clave;

sem_t entradaSeccionCritica;
float ticket, minTicket = 0;
int quieroEntrar = 0, miID, colaNodos [nUsuarios - 1], pendientes = 0;


float findGreater (float, float);
void* receive();
void* inicilizarBuzonSemaforo ();
```

```
int main (int argc, char* argv[]) {

   msg mensajeOut;
   pthread_t idHilo;
   int error, idNodos [nUsuarios - 1];


   // Recogemos la clave para el msgget()
   clave = atoi (argv[1]);


   // Inicializamos el Buzon de comunicaciones y el semaforo de paso
   inicilizarBuzonSemaforo();


   // Recogemos los ID de buzones de los otros procesos
   for (int i = 0; i < (nUsuarios - 1); i++) {

       printf ("Introduzca el ID del buzon del usuario %i: ", i);
       scanf ("%i", &idNodos[i]);

   }

   // Creamos el hilo de receive()
   error = pthread_create (&idHilo, NULL, receive, NULL);

   if ( error == -1 ) {

       printf ("Error durante la creacion del hilo del metodo
recepcion. Cerrando programa.\n");
       exit (-1);

   }

   printf ("Hilo creado correctamente.\n\n");


   while (1) {

       sleep (1);
       fflush (stdin);

       printf ("Esperando activacion.\n");
```

```
        while(getchar() != 'n');


        srand( (unsigned) time (NULL) );

        // Generamos un numero aleatorio entre mminTicket y (minTicket +
1)
        ticket = minTicket + (float) ((rand() % 1000) / 1000.0f);


        quieroEntrar = 1;


        // Definimos el mensaje para los otros nodos
        mensajeOut.mtype = 1;
        mensajeOut.ticket = ticket;
        mensajeOut.idOrigen = miID;

        // Realizamos las peticiones para entrar a la Seccion Critica.
        for (int i = 0; i < (nUsuarios - 1); i++ ) {

            do {

                error = msgsnd (idNodos[i], &mensajeOut,
sizeof(mensajeOut), 0);

                if (error == -1) {

                    printf ("Error durante el envio de peticiones.\n
Reintentado.\n\n");

                }

            } while ( error != 0);


        }

        printf ("Peticiones enviadas correctamente.\n");


        // Esperamos al semaforo de paso
```

```
        sem_wait (&entradaSeccionCritica);

        printf ("Se ha recibito respuesta a todas las peticiones.\n\n");
        printf ("El nodo ha entrado en la Seccion Critica.\n");

        // Para salir definimos el caracter s (para tener tiempo de
probar las peticiones cuando un proceso se encuentra en su SC)
        while(getchar() != 's');
        printf ("El nodo ha salido de la Seccion Critica.\n\n");


        // Definimos el mensaje para los otros nodos
        mensajeOut.mtype = 2;
        mensajeOut.idOrigen = miID;

        quieroEntrar = 0;


        // Una vez fuera de la Seccion Critica,
        // Respondemos todas las peticiones que estaban pendientes
        printf ("Enviando confirmaciones pendientes.\n");

        for (int i = 0; i < pendientes; i++) {

            do {

                error = msgsnd (idNodos[i], &mensajeOut,
sizeof(mensajeOut), 0);

                if (error == -1) {

                    printf ("Error durante el envio de peticiones.\n
Reintentado.\n\n");

                }


            } while ( error != 0);

            printf ("Confirmacion enviada.\n");

        }
```

```
        // ? pendientes = 0;

    }

    return 0;
}




float findGreater (float n1, float n2) {
    return (n1 > n2) ? n1 : n2;
}


void* inicilizarBuzonSemaforo () {

    int error;

    // Inicializamos el buzon
    miID = msgget (clave, IPC_CREAT|0777);

    if (miID == -1) {

        printf ("\nError creando el buzon. Cerrando programa.\n");
        exit(-1);

    }

    printf ("Buzon con ID %i, creado correctamente.\n", miID);


    // Inicializamos el semaforo de paso
    error = sem_init(&entradaSeccionCritica, 0 ,0);

    if (error == -1) {

        printf ("\nError inicializando el semaforo de paso. Cerrando
programa.\n");
        exit(-1);

    }
```

```
   return 0;
}


void* receive () {

   int error, nConfirmaciones;
   msg mensajeIn, mensajeOut;



   while (1) {

       error = msgrcv (miID, &mensajeIn, sizeof(mensajeIn), 0, 0);

       if (error == -1) {

           printf ("Error durante el recibimiento de peticiones.\n
Reintentado.\n\n");

       }
       else if (mensajeIn.mtype == 2) {

           printf ("Confirmacion recibida.\n");

           nConfirmaciones++;

           if (nConfirmaciones == (nUsuarios - 1)) {

               // Accionamos el semanforo de paso
               sem_post (&entradaSeccionCritica);

               // Reiniciamos el contador de Confirmaciones
               nConfirmaciones = 0;

           }

       }
       else if (quieroEntrar == 1) {

           // Guardamos la peticion en la cola de pendientes
           colaNodos [pendientes] = mensajeIn.idOrigen;
```

```
            pendientes++;

        }
        else if ( (quieroEntrar == 0) || ((ticket > mensajeIn.ticket) ||
(ticket == mensajeIn.ticket && miID > mensajeIn.idOrigen)) ) {

            // Recojemos y actualizamos el valor del ticket

            minTicket = findGreater (minTicket, mensajeIn.ticket);

            mensajeOut.idOrigen = miID;
            mensajeOut.ticket = 0;
            mensajeOut.mtype = 2;


            // Confirmamos la peticion entrante

            do {

                error = msgsnd (mensajeIn.idOrigen, &mensajeOut,
sizeof(mensajeOut), 0);

                if (error == -1) {

                    printf ("Error durante el recibimiento de
peticiones.\n Reintentado.\n\n");

                }

                printf ("Confirmacion enviada.\n");

            } while (error != 0);

        }

    }



}
```