Universidad Nacional de General Sarmiento

SISTEMAS OPERATIVOS Y REDES

Mini TP de Semáforos - **“El Gran Asado”**

Estudiantes:

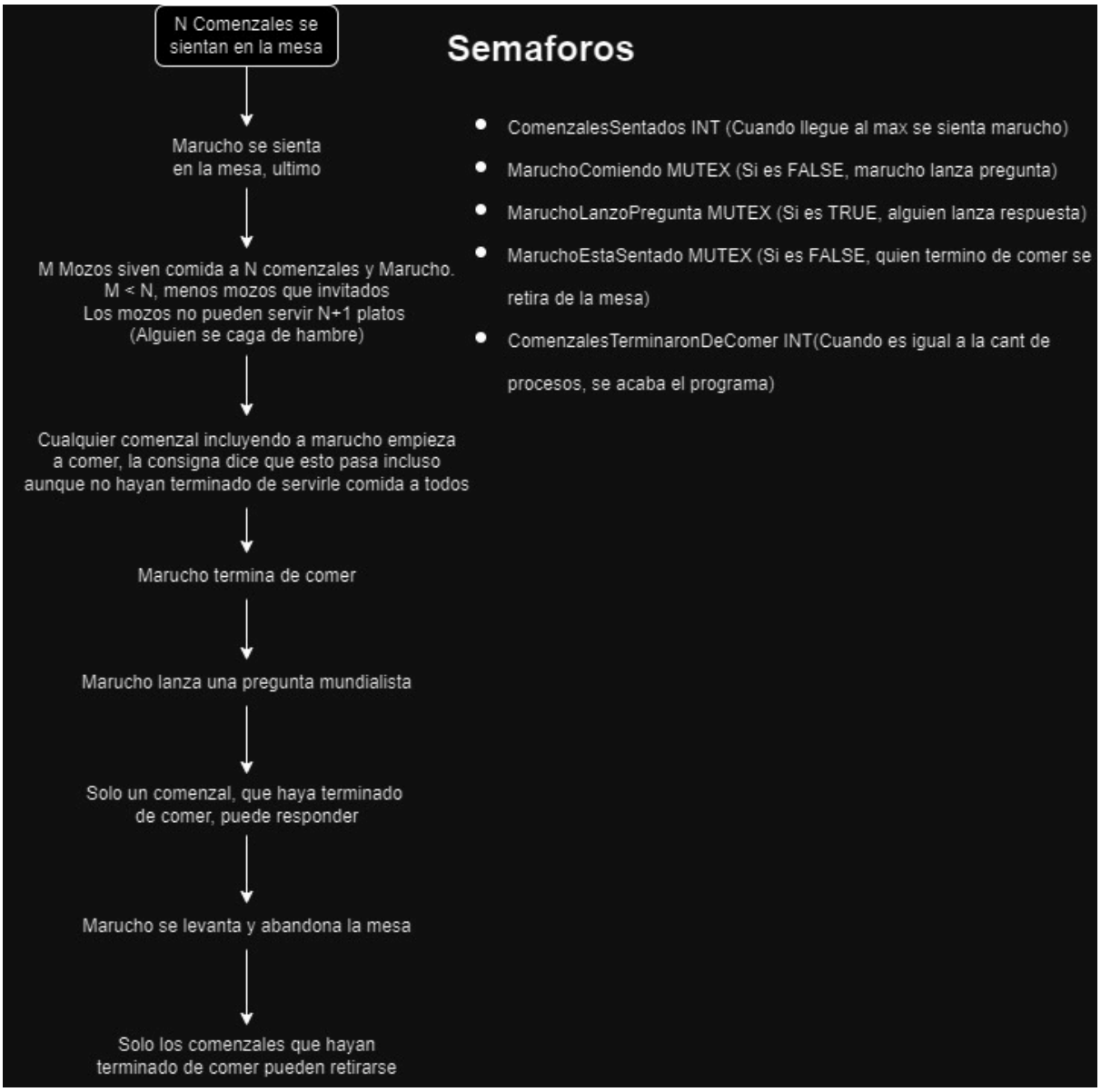
- Tomas Menegol
- Alan Rodriguez
- Leonardo Mendoza

Docente: Mariano Vargas

Comisión: SorC1

Grupo N°: 8

1) Esquema inicial del código:



Semáforos:
Funciones adicionales

manuchoCome()
{
printf(“Manucho come”);
sleep(5)
printf(“Manucho termino de comer”);
}

invitadoCome()
{
printf(“Invitado %d come”)
sleep(5)
printf(“Invitado %d termino de comer”);
}

(A modo de aclaración, estas funciones, junto con el comportamiento de los mozos se juntaron en una única función)

Procesos adicionales

Inivitado_Responde - Único, solo un invitado responde -
{
p(sem_invitado_disponible) /*Si hay un invitado que terminó de comer se ejecuta - en particular, como este es un proceso único, solo un invitado responde*/

p(sem_se_lanzo_pregunta) // Si se lanzo la pregunta
Lanzar_respuesta_mundialista() //Se lanza la respuesta
v(sem_se_lanzo_respuesta) //Se indica que se lanzó la respuesta
}

TodosSentados()
{
p(sem_comensales_sentados)
p(sem_comensales_sentados)
p(sem_comensales_sentados)
(...) // Repite n veces hasta asegurarse de que todos los comensales están sentados

v(sem_manucho_sentado) // Ahora manucho se sienta al último

}

Procesos Principales

Invitado - Este modelo se repite por los n invitados -
{
Sentarse(invitado) //Los invitados se sientan sin esperar a nadie
v(sem_comensales_sentados) // Incrementa la cantidad de comensales sentados

p(sem_plato_servido) //Si el mozo sirvió toma el recurso
invitadoCome()
v(sem_invitado_disponible) //aumenta el acumulador de invitados que pueden responder, en particular debido al proceso único **InvitadoResponde** sólo uno responderá.

p(sem_manucho_se_levanto) //Una vez manucho se levantó los invitados pueden irse
Levantarse()
v(sem_manucho_se_levanto) //Devuelve el semáforo para que los demás invitados puedan accederlo, o en su defecto termine la ejecución
}

Manucho{
p(sem_manucho_se_sienta) // Espera a que todos se sienten
Sentarse(manucho)
v(sem_manucho_sentado) //indica que manucho ya está sentado

p(sem_plato_servido)
ManuchoCome()
Lanzar_Pregunta_mudialista()
v(sem_se_lanzo_pregunta)

p(sem_se_lanzo_respuesta)
Enojarse()
Levantarse()
v(sem_manucho_se_levanto) //Indica que manucho se levantó para que el resto pueda hacerlo
}

Mozos {
p(sem_manucho_sentado) //Antes de servir espera que manucho se siente

p(sem_mozos_disponibles) //Decrementa la cantidad de mozos disponibles
p(invitados_sentados) //Espera que haya al menos un invitado sentado

Servircomida() // Tanto Manucho como los invitados compiten por el cpu

v(sem_plato_servido) //Incremento los platos servidos
v(sem_mozos_disponibles) //Una vez se sirvió se incrementa la cantidad de mozos
}

2) En nuestra solución se divide en 2 grupos de threads, manucho y N invitados. Los mozos son representados como un recurso finito, o sea, un semáforo.

Los semáforos utilizados fueron los siguientes:

- sem_comensales_sentados

Usado en:
- Invitado: Cada vez que un invitado se sienta, el semáforo se incrementa.
- TodosSentados: Verifica que todos los invitados se hayan sentados.

- sem_manucho_se_sienta

Usado en:
- Manucho: Controla que Manucho sea el último en sentarse.

- sem_manucho_sentado

Usado en:

- Manucho: Una vez que Manucho se sienta, el semáforo se incrementa
- Mozos: Espera que Manucho esté sentado antes de servir la comida.

- sem_plato_servido

Función: Controla cuándo se sirve la comida, permitiendo que Manucho y los invitados coman.

Usado en:
- Invitado: Espera a que le sirvan la comida antes de comer.
- Manucho: Espera a que le sirvan la comida antes de comer.
- Mozos: Una vez servido el plato, el semáforo se incrementa.

- sem_invitado_disponible

Usado en:
- Invitado: Después de comer, el semáforo se incrementa indicando que está disponible para responder la pregunta.

- Invitado_Responde: Espera a que haya un invitado disponible para lanzar la respuesta.

- sem_se_lanzo_pregunta

Usado en:
- Manucho: Lanza la pregunta e incrementa el semáforo.
- Invitado_Responde: Espera a que Manucho lance la pregunta para responder.

- sem_se_lanzo_respuesta

Usado en:
- Invitado_Responde: Una vez que un invitado responde, incrementa este semáforo.
- Manucho: Espera a que alguien responda antes de continuar con su proceso.

- sem_manucho_se_levanto

Usado en:
- Invitado: Espera a que Manucho se levante para poder hacerlo también.
- Manucho: Una vez que se levanta, incrementa el semáforo.

- sem_mozos_disponibles

Usado en:
- Mozos: Se decrementa cuando un mozo sirve y se incrementa cuando queda disponible nuevamente.

3) Si, puede producirse inanición durante el momento de servir debido a que siempre hay menos mozos que invitados. Esto resulta en que algún comensal se mantenga esperando a que lo sirvan mucho más tiempo que sus pares más afortunados, quienes pudieron acceder al recurso y ser servidos.

Adjuntamos el código (**path: sorc1g18/entregableSemaforos/elGranAsado.c**)

- Resaltamos que, la implementación está sujeta a cambios lógicos debido al desarrollo en conjunto del programa, desde ya nos disculpamos por las molestias.

Código:

```
# include <stdio.h>
# include <unistd.h>
```

```
# include <stdlib.h>
# include <pthread.h>
# include <semaphore.h>
# include <string.h>

int m = 3; //Cant de mozos
int n = 4; //Cant de invitados
int comensalesSentados = 0; //Cant de comensales sentados al inicio
int comensalesLibres = 0;

//Semáforos a utilizar
sem_t manuchoSentado;
sem_t manuchoComiendo;
sem_t todosComensalesSentados;
sem_t manuchoLanzoPregunta;
sem_t comensalLanzaRespuesta;
sem_t mozosDisponibles;
sem_t manuchoSeLevanto;

void servirComida(int  id)
{
	sem_wait(&manuchoSentado);	//La primera vez se espera a que marucho se siente para empezar a servir.
	sem_post(&manuchoSentado);

	char comensal[20];
	if(id == 0){
		strcpy(comensal, "Manucho");	//El string comensal vale "Manucho" si el Id que me llego es 0
	}else{
		snprintf(comensal, sizeof(comensal), "invitado %d", id);
	}

	sem_wait(&mozosDisponibles);	// Espera hasta que un mozo esté disponible (disminuye el semáforo)
	printf("Mozo está sirviendo a %s.\n", comensal);
	sleep(3);			// Simula el tiempo que el mozo tarda en servir al invitado
	sem_post(&mozosDisponibles);	// Una vez que se sirvió el plato libero al mozo.
	printf("%s está comiendo.\n", comensal);
	sleep(5);			// Simulamos que el invitado está comiendo
	printf("%s ha terminado de comer.\n", comensal);

	if(id != 0){
		comensalesLibres++;	//Aumentó el contador de comenzales que terminaron de comer
```

```
        }else{                      //Si el que terminó de comer es marucho
                sem_post(&manuchoComiendo);
        }
}

void* levantarse(int  id)
{
        sem_wait(&manuchoSeLevanto);
        printf("Invitado %d se levantó de la mesa.\n", id);
        sem_post(&manuchoSeLevanto);
}

void* sentarse(int  id)
{
        comensalesSentados++;
        printf("Invitado %d se sentó.\n", id);
        if(comensalesSentados == n){
                sem_post(&todosComensalesSentados);
        }
}

void* lanzar_pregunta()
{
        //Manucho es el único que lanza pregunta
        sem_wait(&manuchoComiendo);             //Cuando termine de comer manucho
        printf("Manucho: Quien consideran que es el próximo campeón del mundo???\n");
        sem_post(&manuchoLanzoPregunta);
}

void* lanzar_respuesta(int id)
{
        if(comensalesLibres == 1){
                sem_wait(&manuchoLanzoPregunta);//Si ya se hizo la pregunta
                printf("Comensal %d: Francia tiene potencial.\n", id);
                sem_post(&comensalLanzaRespuesta);
        }
}

void* sentarse_manucho()
{
        sem_wait(&todosComensalesSentados);     //Necesito saber si todos los comenzales
se sentaron //Quiero ese recurso //Bueno se entiende
        printf("Manucho se sentó.\n");
        sem_post(&manuchoSentado);
}
```

```
void* enojarse()
{
	//El único que se enoja en nuestro problema es manucho Manucho se enoja cuando se lanza la respuesta
	sem_wait(&comensalLanzaRespuesta);
	printf("Manucho: -Se enoja por la respuesta!-\n");
	printf("Manucho: -Se levanta de la mesa!-\n");
	sem_post(&manuchoSeLevanto);	//Manucho se levantó de mesa!
}

//Comportamiento invitados.
void* invitados(void* arg)
{
	int invitado_id = *(int*)arg;	//Busco el ID de cada thread
	sentarse(invitado_id);
	servirComida(invitado_id);
	lanzar_respuesta(invitado_id);
	levantarse(invitado_id);
	pthread_exit(NULL);		//Thread termina su laburo
}

//Comportamiento de Manucho.
void* manucho(void* arg)
{
	sentarse_manucho();
	servirComida(0);
	lanzar_pregunta();
	enojarse();
	pthread_exit(NULL);
}

int main () {
	//Defino los threads
	pthread_t manuchot;

	pthread_t hilos_invitados[n];	//Array de mozos
	int id_invitados[n];		//Array que va a contener los ID de los threads de mozos

	//Inicializo semáforos y les defino su primer estado
	sem_init(&manuchoSentado, 0, 0);	//Este semáforo se va a usar para que los mozos esperen a que marucho se siente para servir
	sem_init(&todosComensalesSentados, 0 ,0);
	sem_init(&manuchoComiendo,0,0);
	sem_init(&manuchoLanzoPregunta,0,0);
```

```
	sem_init(&comensalLanzaRespuesta,0,0);
	sem_init(&mozosDisponibles, 0, m);	//Mozos disponibles dado por M
	sem_init(&manuchoSeLevanto, 0, 0);

	//Hilo para manucho
	pthread_create(&manuchot, NULL, *manucho, NULL);

	for (int i = 0; i < n; i++)	//Instancio los mozos y los mandó a hacer su función "principal"
	{
		id_invitados[i] = i + 1;//Conforme se van asignando threads, voy armando sus IDs desde 1 a N
		pthread_create(&hilos_invitados[i], NULL, *invitados, &id_invitados[i]); //Inicializo los invitados, los mandó a hacer su funcion, y como parametro le paso su id
	}

	for (int i = 0; i < n; i++)	//Mató a los invitados (Solo se da cuando salieron de la función principal "invitados")
	{
		pthread_join(hilos_invitados[i], NULL);
	}

	pthread_join(manuchot,NULL);	//Mato a manucho

	printf("Nos vemos! \n");

	return 0;
}
```