Sincronización: proceso por el cual se lográ coordinar las actividades de 2 o mas hilos. Se usa cuando 2 o mas hilos necesitan acceso a un recurso compartido que solo puede ser usado por un hilo a la vez

Exclusion Mutua: actividad que realiza el S.O. para evitar que 2 o mas procesos ingresen al mismo tiempo a un area de datos compartidos o accedan a un mismo recurso.

Sección crítica: porcion de codigo en un programa a la que se accede en un recurso compartido y no debe ser accedido por mas de 1 hilo a la vez.

Abrazo mortal: Cuando en un conjunto de procesos cada uno de ellos espera un suceso que solo puede originar otro proceso del mismo conjunto

Espera activa: cuando un proceso repetidamente verifica una condicion, o si el ingreso a una sección critica esta habilitada.

Mora Guzmán José Antonio

API Semaforos

#include <semaphore.h> ⟶ Libreria

int sem_init(sem_t *sem, int pshared, unsigned int value);
↳ inicializa semaforo

int sem_destroy(sem_t *sem);
↳ libera recursos asociados al semaforo

Sem_t *sem_open(const char *name, int oflag)
↳ le da nombre a un semaforo

int sem_close(sem_t *sem) → cierra el descriptor SEM

int sem_unlink(const char *name)
↳ Quita el nombre al semaforo

int sem_wait(sem_t *sem)
↳ Decrementa el semaforo y si es <0 bloquea el proceso

int sem_post(sem_t *sem)
↳ desbloquea el semaforo

int sem_getvalue(sem_t *restrict_sem,
int *restrict_sval)
↳ obtiene el valor del Semaforo y lo almacena en sval

Mora Guzman Jose Antonio

Productor - Consumidor.

Codigo del productor

```
#include <sys/mman.h>
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<fcntl.h>
#include<unistd.h>
#define Max_buffer=1024
#define datosaproducir=100000
void productor(void);
sem_t *huecos;
sem_t *elementos;
int *buffer; /*puntero al buffer de numeros enteros */
int main(void){
   int shd;   /*se crean e inician semaforos */
huecos=sem_open("HUECOS",O_CREAT,00666,1024);
elementos=sem_open("elementos",O_CREAT,00666,0);
if(huecos==SEM_FAILED||elementos==SEM_FAILED){
   perror("ERROR al abrir semaforo");
   exit(1);
}
/*Se crea el segmento de mem.compartida usado como buffer circular*/
shd=open("BUFFER",O_CREAT|O_TRUNC|O_RDWR,00666);
  if(shd==-1){
    perror("ERROR en open shd");
     exit(1);
}ftruncate(shd,(MAx_buffer*sizeof(int)));
buffer=mmap(NULL,(Max_buffer*sizeof(int)),PROT_WRITE,
        MAP_SHARED,shd,0);
if(buffer==MAP_FAILED){
   perror("ERROR en mmap");
    exit(1);
}
  productor(); //llamamos a funcion Productor
```

```
//Se libera el buffer
munmap(buffer, Max_buffer *sizeof(int));
close(shd);
unlink("BUFFER");
//CERRAMOS y Destruimos semaforos
sem_close(huecos);
sem_close(elementos);
unlink("HUECOS");
unlink("elementos");
  return 0;
}
//funcion productor
void productor(void){
 int dato; //dato a producir
 int posicion=0;//pos. donde insertar elemento
 int j;
for(j=0;j<datosaproducir;j++){
   dato=j;
   printf("Produce %d\n",dato);
   sem_trywait(huecos);//un hueco menos
    buffer[posicion]=dato;
    posicion=(posicion+1)%Max_buffer;//nueva posición
     sem_post(elementos);//un elemento mas
  }
   return;
  }
```

CONSUMIDOR:

```
#include<sys/mman.h>
#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<fcntl.h>

#define MAX_BUFFER=1024 //tamaño del buffer
#define Datosaconsumir=100000//datos a consumir

void consumidor(void);

sem_t *huecos;
sem_t *elementos;
int *buffer // buffer numeros enteros

int main(void){
  int shd;
  //abrimos semaforos
  huecos=sem_open("HUECOS",0);
  elementos=sem_open("ELEMENTOS",0);
  if(huecos==SEM_FAILED||elementos==SEM_FAILED){
      perror("Error al abrir semaforos");
      exit(1);
  }
  // Se abre memoria compartida usada como buffer circular
    shd=open("BUFFER",O_RDONLY);
    if(shd==-1){
        perror("ERROR en open shd");
        exit(0);
    }
  buffer=(int*)mmap(NULL,MAX_BUFFER*sizeof(int),
        PROT_READ,MAP_SHARED,shd,0);
    if(buffer==NULL){
        perror("ERROR en mmap");
        exit(1);
    }
  consumidor(); // llamamos a funcion consumidor
  // Se libera buffer
  munmap(buffer,MAX_BUFFER*sizeof(int));
   close(shd);
  // cerramos semaforos
  sem_close(huecos);
   sem_close(elementos);
    exit(0);
}
```

```
// funcion consumidor

void consumidor(void){
 int dato; //dato a consumir
 int posicion=0; //posicion del dato a consumir
 int j;
 for(j=0; j<Datosaconsumir; j++){
  dato=j
  sem_wait(elementos); //un elemento menos
  dato=buffer[posicion];
  printf("consume %d\n", dato);
  posicion=(posicion+1) % MAX_BUFFER; //nueva posicion
  sem_post(huecos); //un hueco mas
 }
   return;
 }
```