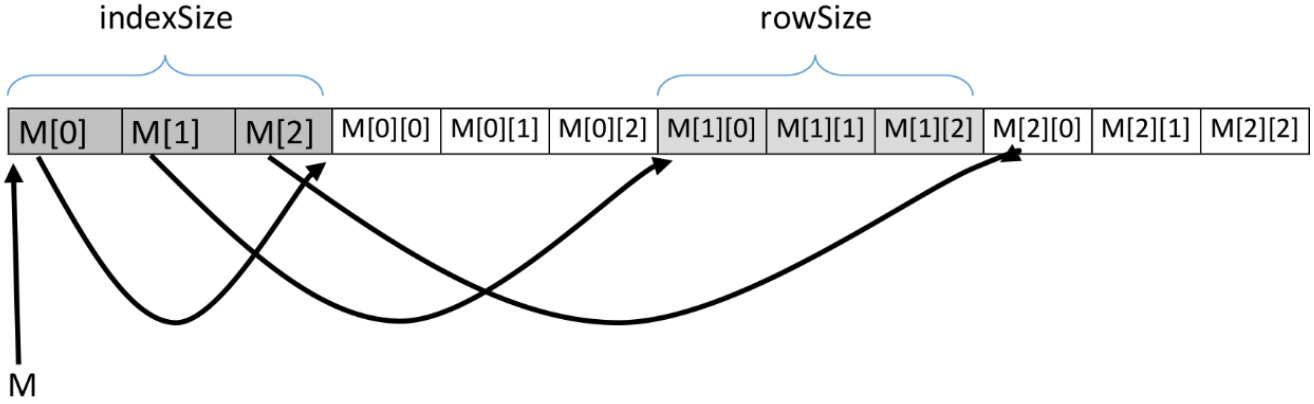
Taller – Memoria compartida
Objetivo: Manejo de matrices en segmentos de memoria compartida tipo Unix System V (No POSIX)

Si ha desarrollado la habilidad del manejo de vectores en segmentos de memoria compartida, un paso natural es el manejo de vectores multidimensionales en un único segmento. En este contexto, recuerde que una forma típica del manejo de matrices dinámicas mediante el uso de las utilidades que disponen los lenguajes para esto, tales como (en C): *Malloc, calloc, realloc* y *free*, puede ser así:

```
double **m;
m = (double **) calloc(rows, sizeof(double *));
for( i = 0; i< rows; i++)
  m[i] = (double *) calloc(cols, sizeof(double));
```

Bajo este enfoque las direcciones de memoria almacenas en *m* y en cada fila *m[i]*, aunque almacenados en el segmento *Heap* compartido con el segmento de la pila *SS*, no necesariamente son consecutivos en el espacio virtual de memoria, tal como sí sería usando una declaración típica de la forma *double m[rows][cols]*. Esto dificultaría un almacenamiento en un único segmento compartido que es continuo por definición. Por lo tanto, esta forma no es adecuada. Una alternativa seguirá el siguiente esquema:



Considere las siguientes funciones:

La función *sizeof_dm(…)* permite determinar el tamaño que se debe reservar en el segmento de memoria compartida para una matriz de elementos con tamaño *sizeElement* de *rows* filas por *cols* columnas.

```
unsigned int sizeof_dm(int rows, int cols, size_t sizeElement){
        size_t size;
    size  = rows * sizeof(void *); //indexSize
    size += (cols * rows * sizeElement);  //Data size
        return size;
}
```

Así, un llamado típico de esta función para crear una matriz de *double* de 100 filas por 100 columnas sería:

int size = sizeof_dm(100, 100, sizeof(double))

Una vez se ha estimado el tamaño que requerirá el segmento de memoria compartida, lo siguiente es solicitar la creación del segmento mediante *shmget* y mapear el segmento a la memoria local mediante *shmat.*

Para poder crear el esquema de la figura anterior se debe crear el segmento de direccionamiento (la parte IndexSize), esta sección configura los punteros al interior del segmento de memoria compartida para cada una de las filas al interior. Note que M[0] apunta al inicio de los elementos de la primera fila, cada uno de los *cols* elementos M[0][0], M[0][1]...,M[0][*cols*]. Lo mismo para M[1], M[2] hasta M[*rows*].

Este índice se crea con la función *créate_index*, esta función recibe la matriz, el número de filas y columnas, y el tamaño de los elementos.

```
void create_index(void **m, int rows, int cols, size_t sizeElement){
        int i;
        size_t sizeRow = cols * sizeElement;
        m[0] = m + rows;
        for(i=1; i<rows; i++){
                m[i] = (m[i-1] + sizeRow);
        }
}
```

De tal forma que el siguiente conjunto de instrucciones permitiría direccionar un vector 2-dimendional (matriz) en un conjunto continuo de memoria que resida en un segmento de memoria compartida. El valor *sizeElement* deberá ser el tamaño del tipo de dato del que será la matriz.

```
double **matrix = NULL;
int Rows=0, Cols = 0;
size_t sizeMatrix = sizeof_dm(Rows,Cols,sizeof(double));
int shmId = shmget(IPC_PRIVATE, sizeMatrix, IPC_CREAT|0600);
matrix = shmat(shmId, NULL, 0);
create_index((void*)matrix, Rows, Cols, sizeof(double));
```

Acceso a elementos de la matriz
Después de configurar la matriz en el segmento de memoria compartida, es posible acceder a los elementos mediante la notación de matrices tradicional del lenguaje C.

```
// Accediendo a los elementos de la matriz para asignar valores
for (int i = 0; i < Rows; i++) {
    for (int j = 0; j < Cols; j++) {
        matrix[i][j] = (i * Cols) + j;  //Asignar un valor de ejemplo basado en la posición
    }
}

// Imprimir los elementos de la matriz para verificar su correcta asignación
for (int i = 0; i < Rows; i++) {
    for (int j = 0; j < Cols; j++) {
        printf("%f ", matrix[i][j]);
    }
    printf("\n");
}
```

Una vez finalizado el uso, se debe liberar correctamente el segmento de memoria utilizado:

```
// Desconectar el segmento de memoria compartida
shmdt(matrix);

// Eliminar el segmento de memoria compartida
shmctl(shmId, IPC_RMID, NULL);
```

Entorno de desarrollo y herramientas
Sistema Operativo: Unix-like, como Linux o macOS.
Compilador: GCC (GNU Compiler Collection).
IDE Recomendado: (opcional) cualquier editor de texto como VSCode, Sublime, o un entorno integrado como CLion o Eclipse CDT.

Dependencias y Headers:
Incluir los headers necesarios al inicio del archivo fuente, como:

```
#include <stdio.h>      // Para funciones de entrada/salida como printf
#include <stdlib.h>     // Para malloc, calloc, free
#include <sys/shm.h>    // Para funciones de manejo de memoria compartida
#include <sys/ipc.h>    // Necesario para flags IPC
```

Finalmente, el programa se puede compilar usando la instrucción:
gcc -o memoria_compartida programa.c

Practica de uso:
Escriba un programa que implemente las funcionalidades descritas anteriormente y permita compartir una matriz en un segmento de memoria compartida. El programa deberá permitir que el proceso padre escriba el contenido de la matriz asignando un valor cualquiera a cada posición de la matriz.

El proceso hijo deberá imprimir la matriz por pantalla. Para la sincronización utilice las facilidades de señales mediante la función *pause, signal* y *kill*.