****

Tabla de contenidos.

[**Introducción**](#_1sgwxebsyqyc) **3**

[**Marco Teórico**](#_qwqjaa96sr9a) **4**

[**Desarrollo**](#_iugctm4j0qmd) **5**

[**Conclusiones**](#_xt6xbxyugj5) **8**

[**Anexo**](#_a4m417abnspz) **9**

# Introducción.

Para esta práctica se pidió que se desarrollará un programa que multiplique dos matrices de manera que indicando una cantidad de procesos específicos se asignen las multiplicaciones de manera equitativa a cada proceso. Para ello se utilizaran un poco de los conocimientos vistos en la anterior práctica y los casos específicos para el desarrollo del programa constan de dos ejemplos con matrices pequeñas variando el número de procesos y verificando que los resultados de las multiplicaciones de las matrices sean correctos de manera externa y otro ejemplo en donde se haga una multiplicación de matrices de gran tamaño en ambas se vera el tiempo de ejecución y la gráfica resultante de la multiplicaciones de matrices grandes.

Como sabemos la única condición para poder multiplicar matrices es que coincidan el número de columnas de la primera matriz sea igual al número de filas de la segunda matriz, así que cumpliendo esta condición podemos dar solución a cualquier tipo de matrices.

# 

# Una vez que se soluciona esta condición ahora tendremos que solucionar cómo distribuir el número de procesos de acuerdo al número de operaciones por lo que pueden existir tres casos diferentes el primero en donde el número de procesos sea igual al número de operaciones, el segundo donde hay más operaciones que procesos entonces a cada proceso se le tendrá que repartir un número de operaciones y en el caso que haya más procesos que operaciones entonces algunos procesos no serán utilizados.

Entonces para solucionar el problema de multiplicación de matrices con procesos tenemos que solucionar los problemas anteriores esto se verá desarrollado en las siguientes partes de la práctica.

# Marco Teórico.

# Los procesos son la parte fundamental de los sistemas operativos ya que se encargan de realizar todo lo que hace funcionar a una computadora así como cumplir con las solicitudes de los usuarios, los sistemas operativos de Linux nos permiten directamente asignar procesos para resolver operaciones por medio de comandos, permitiéndonos utilizar estos procesos para asignarles diferentes funciones y acortar el tiempo de ejecución al usar diversos procesos, uno de los comandos que nos permite ver este tiempo de ejecución es el de time.

# La sintaxis básica del comando Time de Linux es la siguiente:

# **time [-options] command**

# El cual se compone de los siguientes elementos:

# **time** – indica el uso del comando

# **[-options]** – nos permite agregar modificadores al comando

# **command** – es donde escribimos lo que queremos inspeccionar

# El uso de este comando nos da tres resultados:

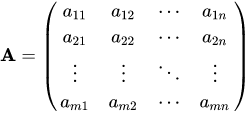
# **Real**: Es el tiempo transcurrido entre la ejecución y la finalización del proceso.

# **User**: El tiempo de CPU gastado en código de modo usuario (fuera del núcleo) durante la ejecución del proceso.

# **Sys**: El tiempo de CPU que transcurre en el núcleo al ejecutar el proceso.

# Este comando es ejecutado fuera del programa ya que de esa forma podremos ver el tiempo que se necesito para finalizar la ejecución de este, mientras que dentro del programa encontraremos las matrices.

Como sabemos las matrices son un arreglo bidimensional de valores, para representar una matriz se utiliza una letra mayúscula y definimos su tamaño por el número de filas y columnas diciendo que una matrices se compone de **mxn** m filas y n columnas



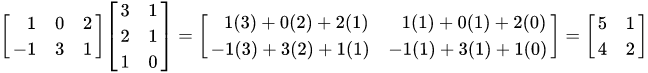
# También podemos realizar operaciones con estas matrices como lo son la suma o la multiplicación de matrices, pero para realizar estas se tiene que cumplir que el número de columnas de la primera matriz sea igual al número de filas de la segunda matriz, si esto no se cumple no se puede realizar la multiplicación.

Por lo que si se tienen dos matrices una de tamaño **mxn** y otra de tamaño **nxñ** al hacer la multiplicación entre ellas nos dará una matriz resultante de tamaño **mxñ.**

La manera de realizar esta operación se hace de la siguiente manera:

1.-Para empezar tenemos que multiplicar los elementos de la primera fila de la primera matriz con los elementos de la primera columna de la segunda matriz de manera que el primer valor de la primera fila se multiplica con el primer valor de la columna uno y se hace esto hasta terminar con todos los valores de la primera fila, estos valores que dan se suman y el resultado es primer resultado de la nueva matriz.

2.-El paso anterior se repite de manera que el valor que tendrá la posición Amn de la matriz resultante será igual a la multiplicación de la fila m por la columna n.



# 

# Las matrices son muy importantes en la computación ya que pueden ser utilizadas para resolver sistemas de ecuaciones y gracias a ellas es como una máquina puede resolver grandes operaciones y ecuaciones complejas en tiempo relativamente corto así como también se pueden realizar sistemas de simulaciones como los que se emplean para simular los efectos del calentamiento global que manejan grandes ecuaciones, incógnitas y factores.

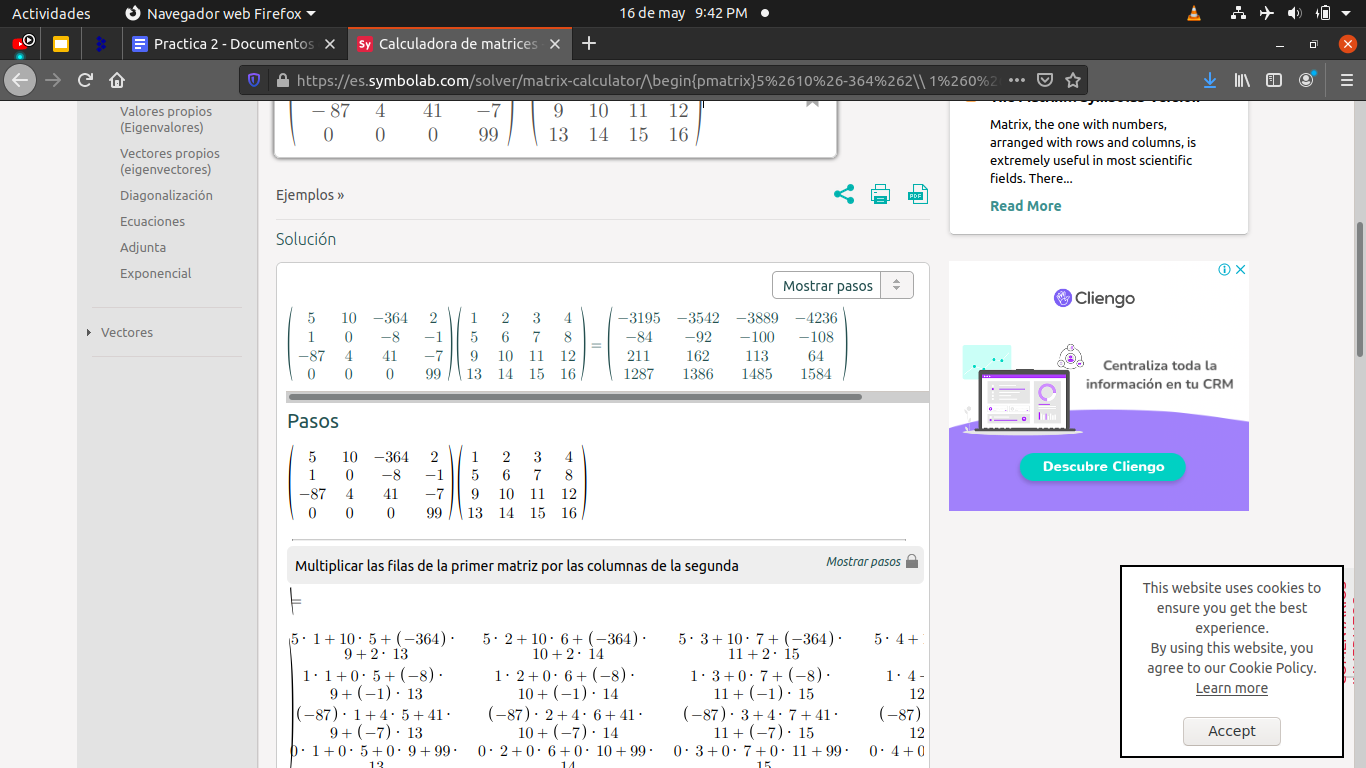
# 

# También son muy útiles para agilizar algunas operaciones algebraicas que de otro modo serían tediosas de resolver. Por ejemplo, calcular el valor n-ésimo (para un n muy grande) de la serie de fibonacci es impráctico por algoritmos recursivos, e iterativos. Lo mejor es optar por algoritmos que dividan el problema con matrices.

# 

# Desarrollo.

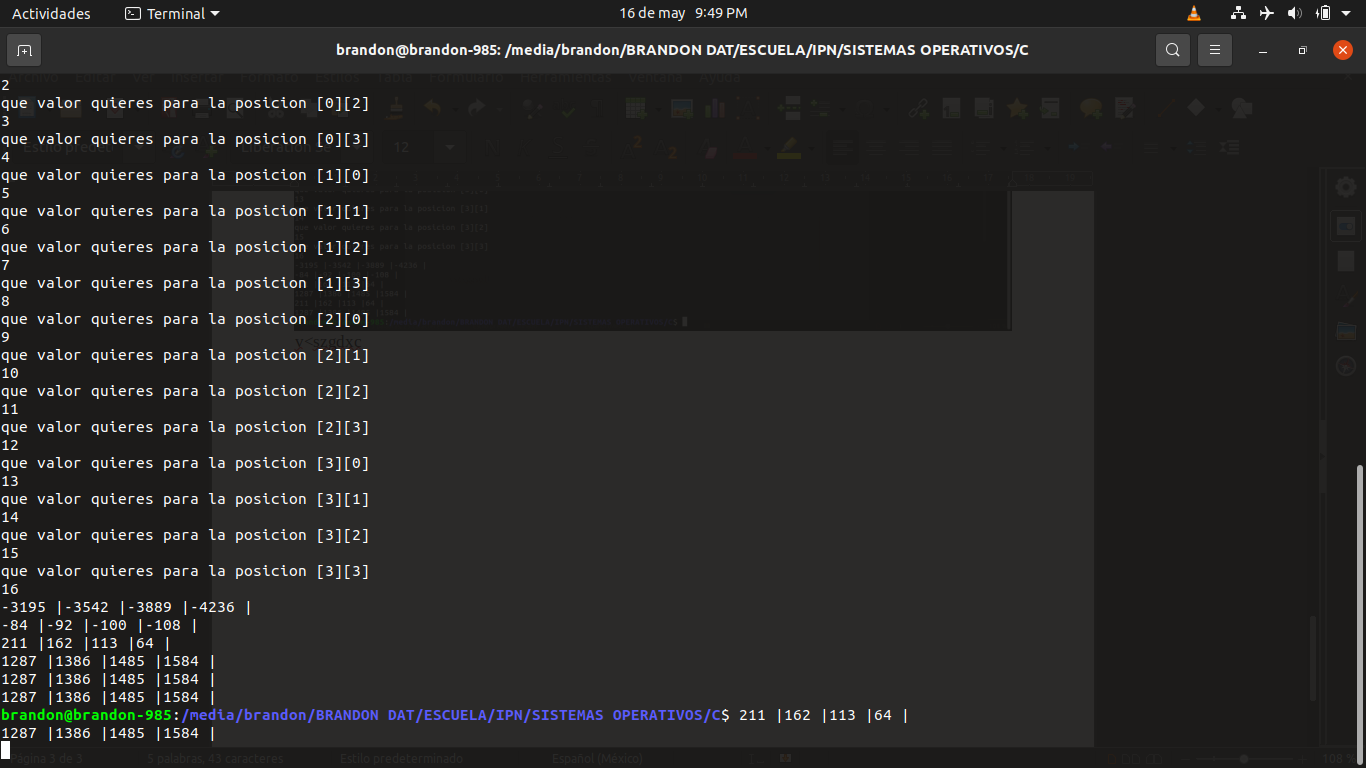
1) Se realizó la siguiente multiplicación de matrices:



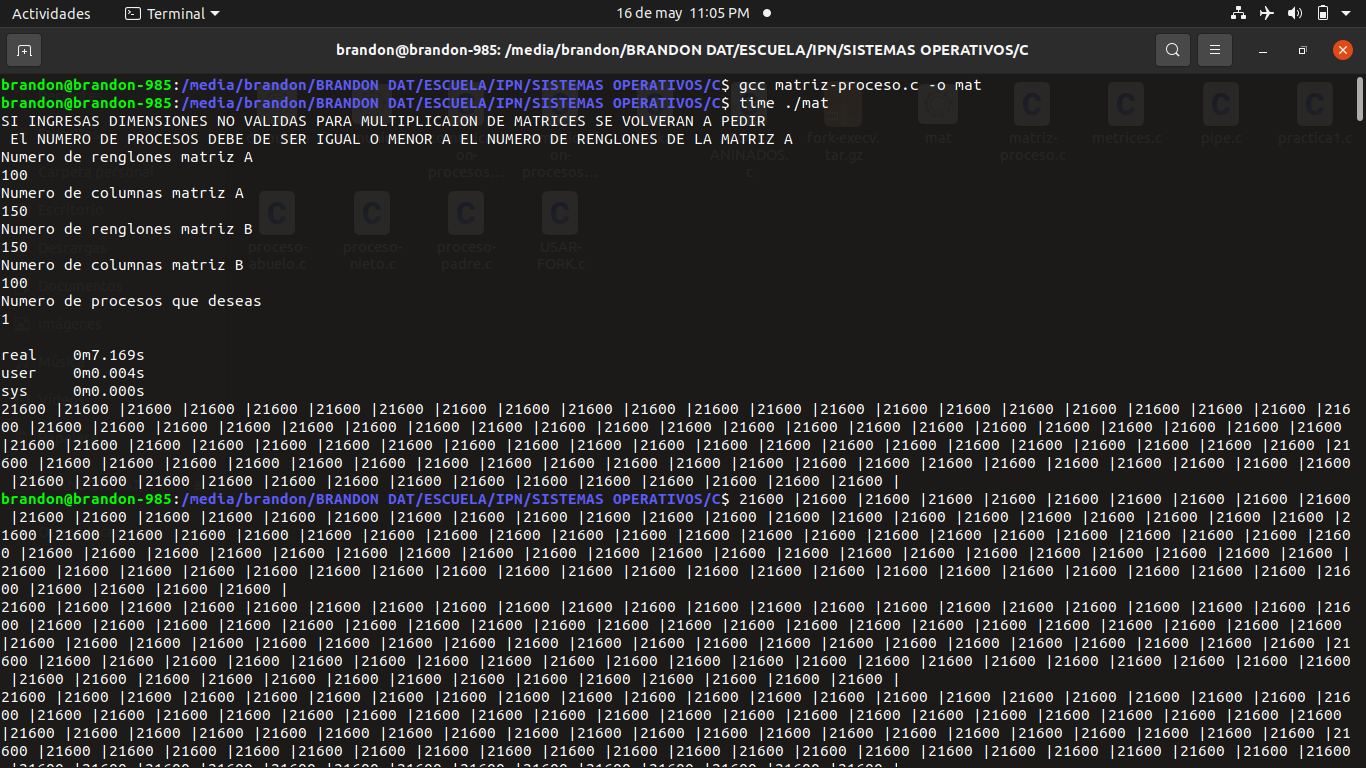
Programa en ejecución.



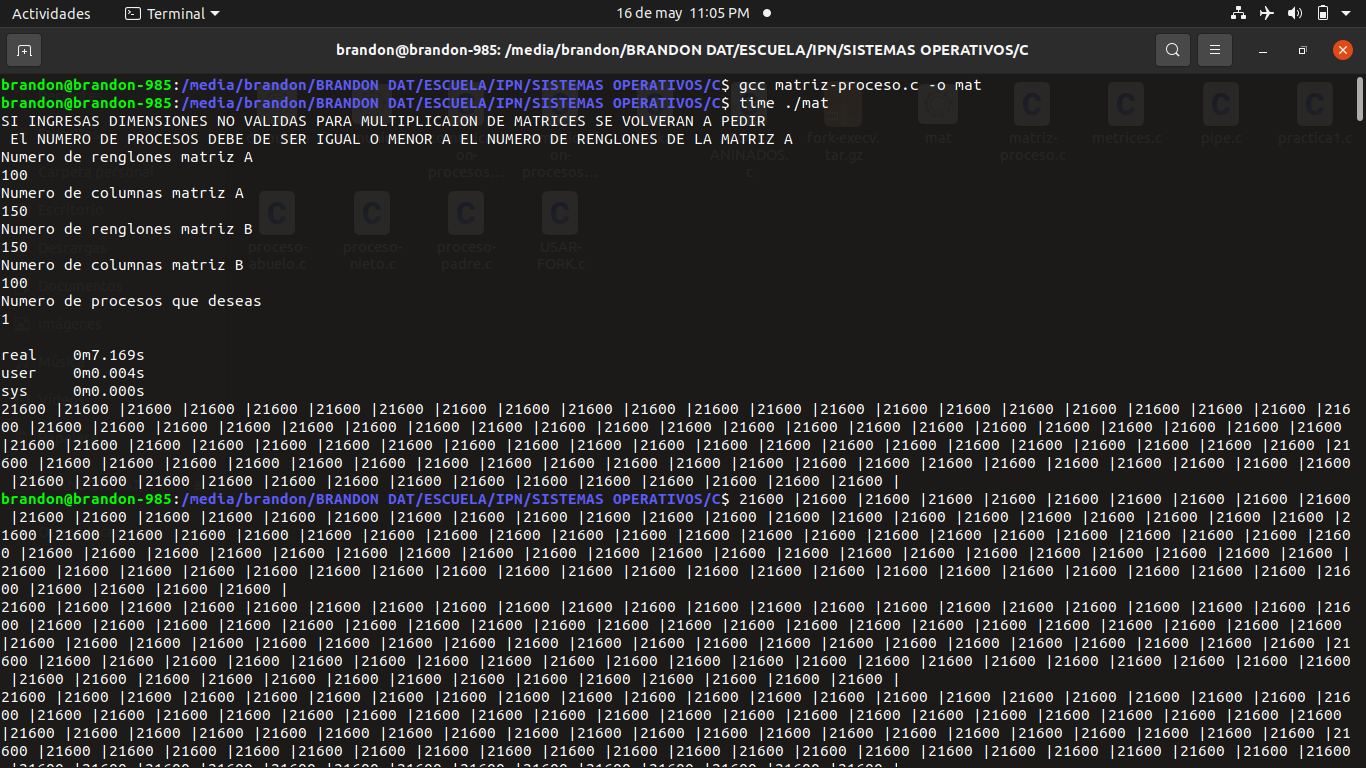
Resultados de la multiplicación de Matrices.



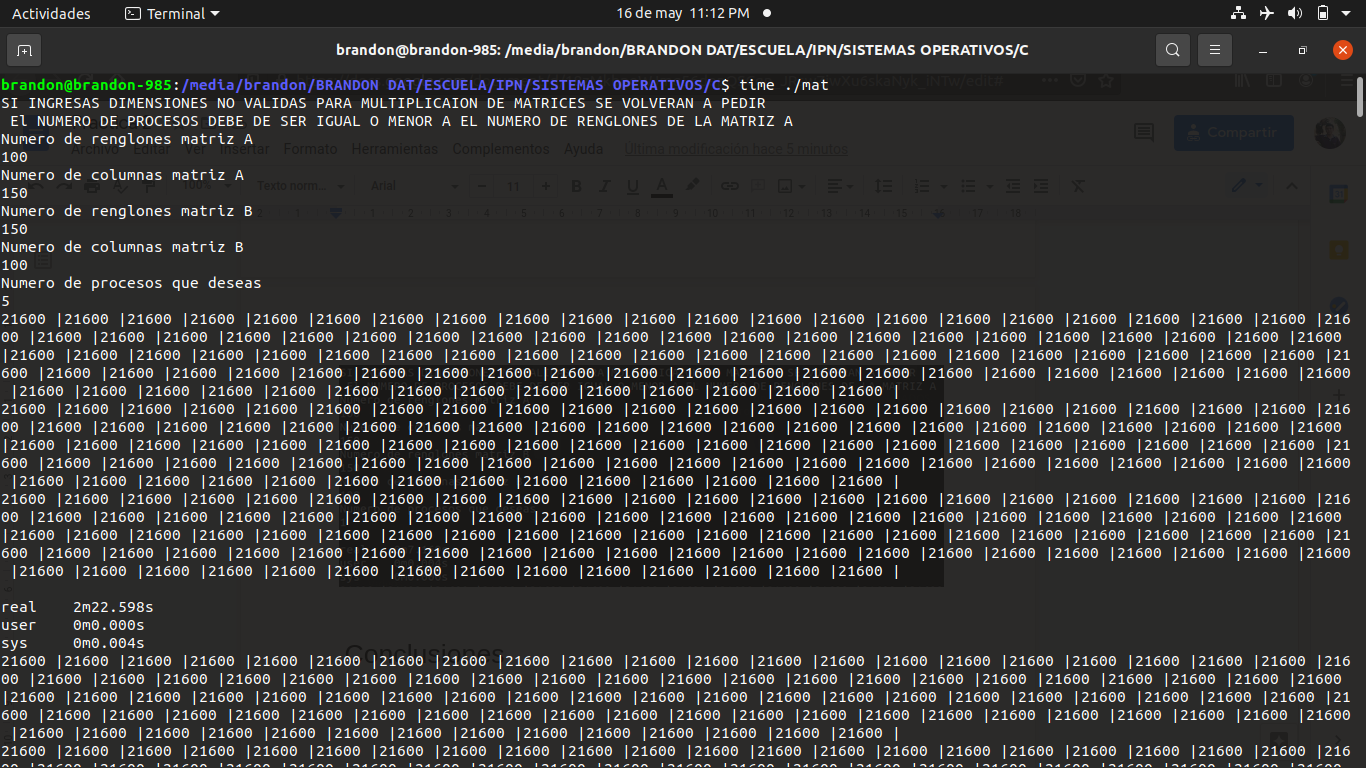
2) Se realizó la siguiente multiplicación de matrices con los siguientes parámetros:



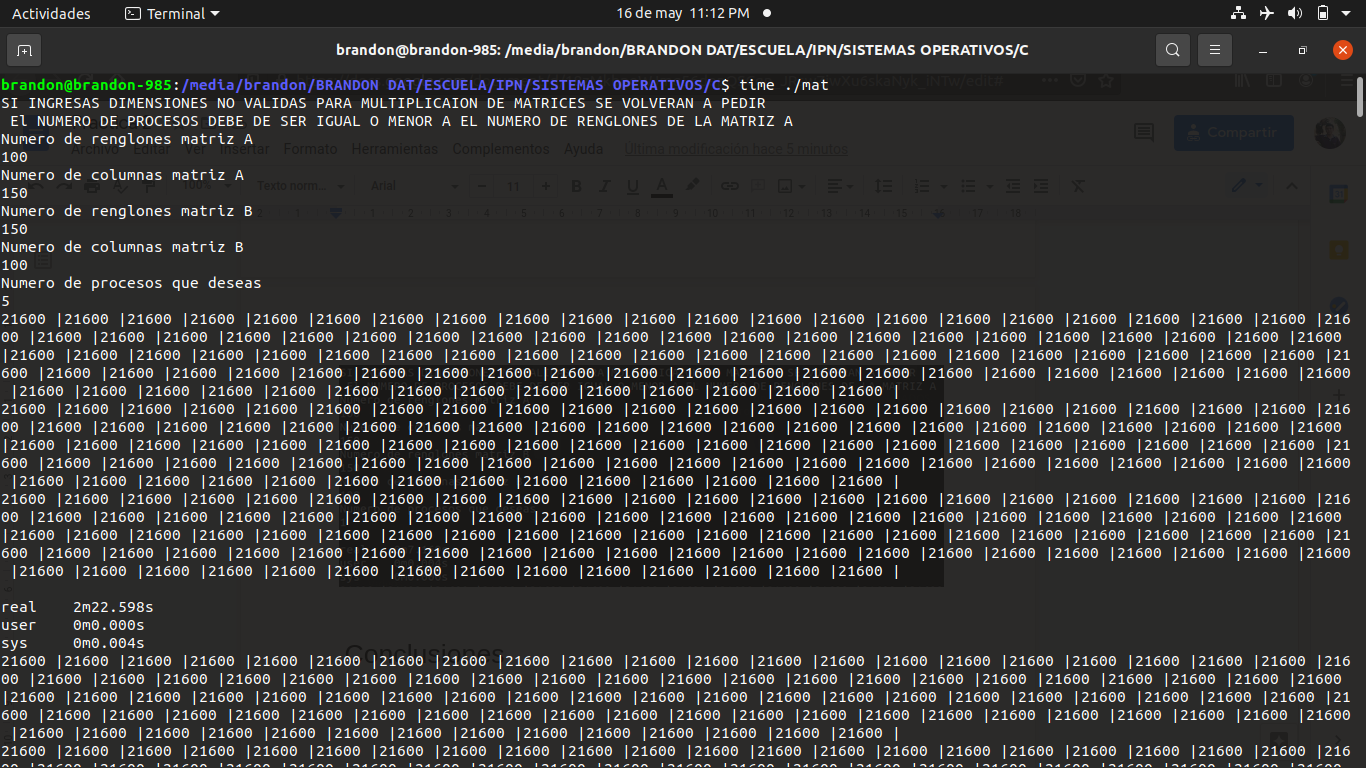
Tiempos de salida:



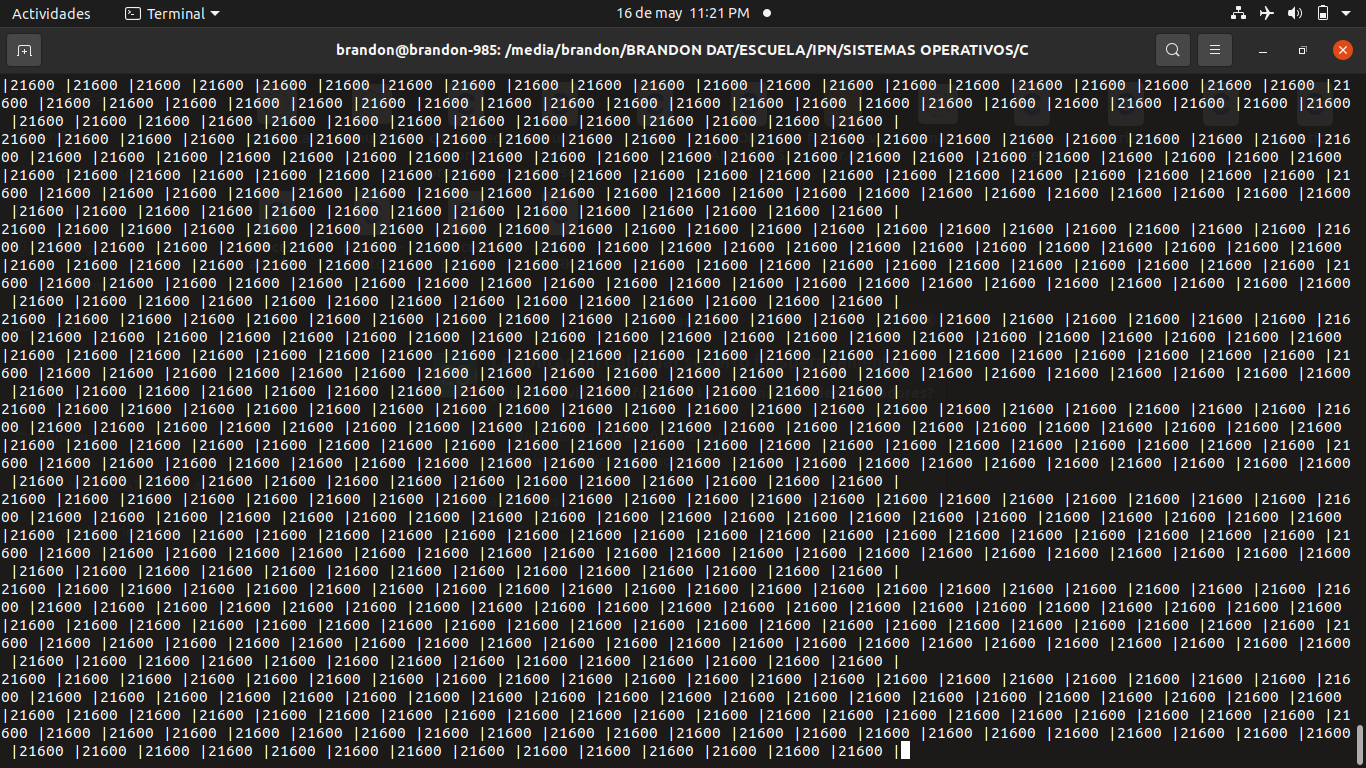
3) Se realizó la siguiente multiplicación de matrices con los siguientes parámetros:



Tiempos de salida:



En este caso podemos observar que una matriz muy grande con bastantes procesos, se comporta de manera ***exponencial***.



# Conclusiones.

Casiano Granados Brandon Antonio:

Cuando ejecutamos un algoritmo en ocasiones es eficiente utilizar más de un proceso para concluirlo, sin embargo el utilizar demasiados procesos generaría que nuestro programa se comporte de manera exponencial.

Otro de los problemas es la espera de los procesos con wait o waitpid, los cuales solo esperan a un solo proceso hijo lo que es difícil saber en qué momento es el adecuado para imprimir el resultado de las matrices.

Viorato Lozada Osmar:

La práctica nos permitió visualizar la implementación de los procesos mediante la resolución de una multiplicación de matrices. Se generaron problemas en la comunicación de procesos por lo que realizó una implementación diferente además de una perspectiva diferente a la que se venían manejando los procesos. Aunque al principio no veía muy clara la utilidad de generar una copia exacta de un proceso, ahora sé que con la lógica correcta, se puede dividir la carga de un algoritmo y que cada proceso haga diferentes secciones.

# Anexo.

Código de implementación.

#include <unistd.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

int pideDat(int\*, int\*, int\*, int\*, int\*);

int divideTareas(int, int, int, int, int\*\*, int\*\*, int\*\*);

int\*\* creaMat(int, int);

int llenaMat(int \*\*, int, int);

int main()

{

int r, c, r2, c2, cantProcesos, \*\*a, \*\*b, \*\*res;

pideDat(&r, &c, &r2, &c2, &cantProcesos);

a=creaMat(r, c);

if(a == NULL)

exit(1);

b=creaMat(r2, c2);

if(b == NULL)

exit(1);

res=creaMat(r, c2);

if(res == NULL)

exit(1);

llenaMat(a, r, c);

llenaMat(b, r2, c2);

divideTareas(r, c, c2, cantProcesos, a, b, res);

free(a);

free(b);

free(res);

return 0;

}

int pideDat(int \*re, int \*co, int \*re2, int \*co2, int \*cant)

{

do

{

puts("SI INGRESAS DIMENSIONES NO VALIDAS PARA MULTIPLICAION DE MATRICES SE VOLVERAN A PEDIR \n El NUMERO DE PROCESOS DEBE DE SER IGUAL O MENOR A EL NUMERO DE RENGLONES DE LA MATRIZ A");

puts("Numero de renglones matriz A");

scanf("%d",re);

puts("Numero de columnas matriz A");

scanf("%d",co);

puts("Numero de renglones matriz B");

scanf("%d",re2);

puts("Numero de columnas matriz B");

scanf("%d",co2);

puts("Numero de procesos que deseas");

scanf("%d",cant);

}while(!(\*co == \*re2) && \*cant<=\*re);

return 0;

}

int divideTareas(int ren, int col, int col2, int proces, int \*\*matA, int \*\*matB, int \*\*matC)

{

if(matA == NULL)

exit(1);

if(matB == NULL)

exit(1);

if(matC == NULL)

exit(1);

int cantProcesXFila = ren/proces;

int residuoProcesXFila = ren%proces;

int ini=0, fin, i=0, espPro, f, k;

pid\_t espera;

int indice[proces\*2];

while (ini<ren)

if(residuoProcesXFila > 0)

{

fin = ini+cantProcesXFila;

indice[i]=ini;

indice[i+1]=fin;

ini += cantProcesXFila+1;

residuoProcesXFila--;

i+=2;

}

else

{

fin = ini+(cantProcesXFila-1);

indice[i]=ini;

indice[i+1]=fin;

ini += cantProcesXFila;

i+=2;

}

for(i=0; i<(proces\*2); i+=2)

switch(fork())

{

case -1:

perror("NO SE LOGRO CREAR EL PROCESO");

exit(1);

break;

case 0:

fin = indice[i+1];

for(ini = indice[i]; ini<=fin; ini++)

{

for(f=0; f<col2; f++)

{

matC[ini][f]=0;

for(k=0; k<col; k++)

matC[ini][f]+= matA[ini][k] \* matB[k][f];

printf("%d |", matC[ini][f]);

}

printf("\n");

}

break;

default:

waitpid(espera, &espPro, 0);

break;

}

return 0;

}

int\*\* creaMat(int ren, int col)

{

int cent, cent2;

int \*\*temp = (int \*\*) malloc(sizeof(int\*)\*ren);

for(cent=0; cent<ren; cent++)

temp[cent] = (int \*)malloc(sizeof(int)\*col);

return temp;

}

int llenaMat(int \*\*mat, int renglon, int columna)

{

if(mat == NULL)

exit(1);

int p, q;

for(p=0; p<renglon; p++)

for(q=0; q<columna; q++)

{

printf("que valor quieres para la posicion [%d][%d] \n",p, q);

scanf("%d",&mat[p][q]);

}

return 0;

}