My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 _Estructura Struct Reference

Estructura estructura.

Public Attributes

- char **cadena** [100]
- int **numero**

3.1.1 Detailed Description

Estructura estructura.

Estructura.

Usaremos esta estructura para pasarsela a las funciones de los procesos

Parameters

cadena	Una cadena de 80 caracteres
numero	Un enterp

The documentation for this struct was generated from the following files:

- Ejercicio12a.c
- Ejercicio12b.c
- Ejercicio6.c

3.2 _Matriz Struct Reference

Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Public Attributes

- int **matriz** [MAX_MATRIZ]
- int **multiplicador**
- int **num**

3.2.1 Detailed Description

Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Parameters

matriz	Array de enteros que contienen la matriz
multiplicador	Numero por el que se multiplicara cada numero de la matriz
num	Numero de la matriz (1 o 2)

The documentation for this struct was generated from the following files:

- Ejercicio13.c
- Ejercicio13mod.c

Chapter 4

File Documentation

4.1 Ejercicio12a.c File Reference

Ejercicio 12 Apartado a.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <sys/time.h>
```

Include dependency graph for Ejercicio12a.c:

4.2 Ejercicio12b.c File Reference

Ejercicio 12 Apartado b.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <sys/time.h>
```

Include dependency graph for Ejercicio12b.c:

Classes

- struct _Estructura

  Estructura estructura.

Typedefs

- typedef struct _Estructura Estructura

  Estructura estructura.

Functions

- int es_primo (int num)

  es_primo
- void * comprueba_primos (void *estructura)

  comprueba_primos
- int main (int argc, char *argv[ ])

  Main.

4.2.1 Detailed Description

Ejercicio 12 Apartado b.

Author

Miguel Angel Sanchez y Juan Velasco

4.2.2 Typedef Documentation

4.2.2.1 Estructura

```
typedef struct _Estructura Estructura
```

Estructura estructura.

Usaremos esta estructura para pasarsela a las funciones de los procesos

4.2.3 Function Documentation

4.2.3.1 comprueba_primos()

```
void* comprueba_primos (
            void * estructura )
```

comprueba_primos

Esta funcion calcula los numeros primos hasta un numero de primos concreto

Parameters

estructua	Puntero a una estructura Estructura (pasado como void)

Returns

pthread_exit

4.2.3.2 es_primo()

```
int es_primo (
            int num )
```

es_primo

Esta funcion comprueba si el numero pasado es un primo o no

Parameters

num	Numero a comprobar que es primo

Returns

-1 (No es) o 0 (Es)

4.2.3.3 main()

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

Main.

Este main comprueba el tiempo que tarda en ejecutar con 100 hilos la funcion comprueba_primos hasta un numero n que se pasa como parametro.

Parameters

argc	Numero de parametros que se pasan
argv	Parametros que se pasan

Returns

EXIT_SUCCESS o EXIT_FAILURE

4.3 Ejercicio13.c File Reference

Ejercicio 13.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <sys/time.h>
```

Include dependency graph for Ejercicio13.c:

Classes

- struct _Matriz

  Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Macros

- #define **MAX_CHAR** 500
- #define **MAX_MATRIZ** 25

Typedefs

- typedef struct _Matriz Matriz

  Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Functions

- void * multiplica_matriz (void *matriz)

  multiplica_matriz Esta funcion multiplica la matriz que se le pasa por un numero entero
- int main (void)

  Main.

Variables

- int tam

  Variable global tam Usaremos esta variable para que podamos compartir el tamaño entre los hilos.

4.3.1 Detailed Description

Ejercicio 13.

Author

Miguel Angel Sanchez y Juan Velasco

4.3.2 Typedef Documentation

4.3.2.1 Matriz

```
typedef struct _Matriz Matriz
```

Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Parameters

matriz	Array de enteros que contienen la matriz
multiplicador	Numero por el que se multiplicara cada numero de la matriz
num	Numero de la matriz (1 o 2)

4.3.3 Function Documentation

4.3.3.1 main()

```
int main (
            void  )
```

Main.

El main sera el encargado de crear las dos estructuras Matrices, pidiendo por pantalla los datos deseados: tamaño, multiplicador y la matriz Despues, genera los hilos donde se realizaran las operaciones.

Returns

EXIT_SUCCESS o EXIT_FAILURE

Creamos las dos estructuras matriz

Introducimos los datos del tamaño y vemos si concuerdan con los que nos pide el enunciado. Hemos admitido tambien una matriz de 1 x 1

Introducimos ambos multiplicadores

Introducimos ambas matrices Aclarar que se pueden meter como aparecen en el enunciado de la practica, pero si se ponen numeros de mas, no seran multiplicados y pueden dar errores de memoria. Si se introducen presionando intro despues de cada numero solo se dejara poner los numeros correctos. Tenemos que añadir otra vez m∗m numeros

Por ultimo, creamos los threads pasandole sus matrices correspondientes

4.3.3.2 multiplica_matriz()

```
void* multiplica_matriz (
            void * matriz )
```

multiplica_matriz Esta funcion multiplica la matriz que se le pasa por un numero entero

Parameters

matriz	Puntero a void que contiene un puntero de tipo Matriz

Returns

void∗

Cambiamos el puntero al tipo original

Dado que es una matriz de tamaño m, tenemos que iterar el array hasta m∗m. Lo guardamos en la misma matriz ya que no nos dicen nada de que haya que guardarla.

Cuando alcanzamos el tamaño de fila deseado (que es la iteracion + 1), pasamos a imprimir la fila correspondiente iterando los nuevos valores de la matriz. Tenemos que poner fflush para que pueda imprimirse todo el buffer junto Por ultimo, aumentamos en uno la fila en la que estamos

4.4 Ejercicio13mod.c File Reference

Ejercicio 13 mod.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include <sys/time.h>
```

Include dependency graph for Ejercicio13mod.c:

Classes

- struct _Matriz

  Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Macros

- #define **MAX_CHAR** 500
- #define **MAX_MATRIZ** 25

Typedefs

- typedef struct _Matriz Matriz

  Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Functions

- void * multiplica_matriz (void *matriz)

  multiplica_matriz Esta funcion multiplica la matriz que se le pasa por un numero entero
- int main (void)

  Main.

Variables

- int tam

  Variable global tam Usaremos esta variable para que podamos compartir el tamaño entre los hilos.
- int fila1 = 0

  Variables fila1 y fila2 Usaremos estas dos variables para que cada hilo sepa por donde va el otro.
- int **fila2** = 0

4.4.1 Detailed Description

Ejercicio 13 mod.

Author

Miguel Angel Sanchez y Juan Velasco

4.4.2 Typedef Documentation

4.4.2.1 Matriz

```
typedef struct _Matriz Matriz
```

Estructura Matriz Guardaremos en esta estructura todos los datos referentes a la matriz. Nota: hemos decidido guardar la matriz en un array en vez de en una matriz como tal. Esto cambiara un poco la forma de proceder en la funcion de multiplica_matriz.

Parameters

matriz	Array de enteros que contienen la matriz
multiplicador	Numero por el que se multiplicara cada numero de la matriz
num	Numero de la matriz (1 o 2)

4.4.3 Function Documentation

4.4.3.1 main()

```
int main (
             void  )
```

Main.

El main sera el encargado de crear las dos estructuras Matrices, pidiendo por pantalla los datos deseados: tamaño, multiplicador y la matriz Despues, genera los hilos donde se realizaran las operaciones.

Returns

EXIT_SUCCESS o EXIT_FAILURE

Creamos las dos estructuras matriz

Introducimos los datos del tamaño y vemos si concuerdan con los que nos pide el enunciado. Hemos admitido tambien una matriz de 1 x 1

Introducimos ambos multiplicadores

Introducimos ambas matrices Aclarar que se pueden meter como aparecen en el enunciado de la practica, pero si se ponen numeros de mas, no seran multiplicados y pueden dar errores de memoria. Si se introducen presionando intro despues de cada numero solo se dejara poner los numeros correctos. Tenemos que añadir otra vez m∗m numeros En este caso asignaremos el numero de la matriz a su variable. Con esto, podremos saber a que fila tenemos que mirar para saber por donde va el otro hilo

Por ultimo, creamos los threads pasandole sus matrices correspondientes

4.4.3.2 multiplica_matriz()

```
void* multiplica_matriz (
             void * matriz )
```

multiplica_matriz Esta funcion multiplica la matriz que se le pasa por un numero entero

Parameters

matriz	Puntero a void que contiene un puntero de tipo Matriz

Returns

void*

Cambiamos el puntero al tipo original

Dado que es una matriz de tamaño m, tenemos que iterar el array hasta m*m. Lo guardamos en la misma matriz ya que no nos dicen nada de que haya que guardarla.

Cuando alcanzamos el tamaño de fila deseado (que es la iteracion + 1), pasamos a imprimir la fila correspondiente iterando los nuevos valores de la matriz. Tenemos que poner fflush para que pueda imprimirse todo el buffer junto Por ultimo, aumentamos en uno la fila1 o fila2 en la que estamos dependiendo del hilo.

4.5 Ejercicio4a.c File Reference

Ejercicio 4 Apartado a.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio4a.c:

Macros

- #define NUM_PROC 6

  Main.

Functions

- int **main** (void)

4.5.1 Detailed Description

Ejercicio 4 Apartado a.

Author

Miguel Angel Sanchez y Juan Velasco

4.5.2 Macro Definition Documentation

4.5.2.1 NUM_PROC

```
#define NUM_PROC 6
```

Main.

Este main crea procesos hijo cada vez que i es congruente con dos

EXIT_SUCCESS o EXIT_FAILURE

4.6 Ejercicio4b.c File Reference

Ejercicio 4 Apartado b.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio4b.c:

Macros

- #define NUM_PROC 6

  Main.

Functions

- int **main** (void)

4.6.1 Detailed Description

Ejercicio 4 Apartado b.

Author

Miguel Angel Sanchez y Juan Velasco

4.6.2 Macro Definition Documentation

4.6.2.1 NUM_PROC

```
#define NUM_PROC 6
```

Main.

Se generan procesos hijos cuando i es congruente con 2 pero esta vez el padre esperara a que termine un hijo (cuando llegue al wait) para terminar

Returns

EXIT_SUCCESS

4.7 Ejercicio4moda.c File Reference

Ejercicio 4 Apartado a modificado.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio4moda.c:

Macros

- #define NUM_PROC 6

  Main.

Functions

- int **main** (void)

4.7.1 Detailed Description

Ejercicio 4 Apartado a modificado.

Author

Miguel Angel Sanchez y Juan Velasco

4.7.2 Macro Definition Documentation

4.7.2.1 NUM_PROC

```
#define NUM_PROC 6
```

Main.

Realizamos lo mismo que el apartado a pero enseñamos de quien es cada hijo y su id o el id del padre

Returns

EXIT_SUCCES

4.8 Ejercicio4modb.c File Reference

Ejercicio 4 Apartado b.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio4modb.c:

Macros

- #define NUM_PROC 6

  Main.

Functions

- int **main** (void)

4.8.1 Detailed Description

Ejercicio 4 Apartado b.

Author

Miguel Angel Sanchez y Juan Velasco

4.8.2 Macro Definition Documentation

4.8.2.1 NUM_PROC

```
#define NUM_PROC 6
```

Main.

Realiza lo mismo que el apartado b pero en este caso mostramos el id del hijo y su padre o solo el id del padre

4.9 Ejercicio5a.c File Reference

Ejercicio 5 Apartado a.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio5a.c:

Macros

- #define **NUM_PROC** 6

Functions

- int main (void)

  Main En este apartado, haremos que cada proceso genere a su vez solo un proceso cuando i no sea congruente con 2.

4.9.1 Detailed Description

Ejercicio 5 Apartado a.

Author

Miguel Angel Sanchez y Juan Velasco

4.9.2 Function Documentation

4.9.2.1 main()

```
int main (
            void  )
```

Main En este apartado, haremos que cada proceso genere a su vez solo un proceso cuando i no sea congruente con 2.

Returns

EXIT_SUCCESS

4.10 Ejercicio5b.c File Reference

Ejercicio 5 Apartado b.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio5b.c:

Macros

- #define **NUM_PROC** 6

Functions

- int main (void)

4.10.1 Detailed Description

Ejercicio 5 Apartado b.

Author

Miguel Angel Sanchez y Juan Velasco

4.10.2 Function Documentation

4.10.2.1 main()

```
int main (
            void  )
```

Main En este apartado haremos que el proceso padre genere tres hijos y espere a que terminen los tres para terminar

Returns

EXIT_SUCCESS

4.11 Ejercicio6.c File Reference

Ejercicio 6.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

Include dependency graph for Ejercicio6.c:

Classes

- struct _Estructura

  Estructura estructura.

Typedefs

- typedef struct _Estructura Estructura

  Estructura.

Functions

- int main (void)

  Main.

4.11.1 Detailed Description

Ejercicio 6.

Author

Miguel Angel Sanchez y Juan Velasco

4.11.2 Typedef Documentation

4.11.2.1 Estructura

```
typedef struct _Estructura Estructura
```

Estructura.

Parameters

cadena	Una cadena de 80 caracteres
numero	Un enterp

4.11.3 Function Documentation

4.11.3.1 main()

```
int main (
            void  )
```

Main.

En este ejercicio crearemos en el proceso padre memoria para un puntero a nuestra Estructura y veremos que se crea una copia para el hijo donde pediremos los parametros por teclado. Tambien veremos que no se peude acceder desde el apdre a esta informacion

Returns

EXIT_SUCCESS

4.12 Ejercicio8.c File Reference

Ejercicio 8 de la practica.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
```

Include dependency graph for Ejercicio8.c:

Functions

- void ejecutar (char *exe, char *mode)

  Funcion ejecutar.
- int main (int argc, char *argv[ ])

  Main del programa.

4.12.1 Detailed Description

Ejercicio 8 de la practica.

Author

Miguel Angel Sanchez y Juan Velasco

4.12.2 Function Documentation

4.12.2.1 ejecutar()

```
void ejecutar (
            char * exe,
            char * mode )
```

Funcion ejecutar.

Esta funcion ejecuta el comando con la función que se han pasado por los parametros

Parameters

exe	Comando a ejecutar
mode	Modo de ejecucion

Returns

EXIT_SUCCES o EXIT_FAILURE

Dado que no sabemos que path vamos a necesitar hemos procedido de la siguiente manera: Tras leer la documentacion descubrimos que estas funciones, si no hacen nada, devuelven un parametro y si funciona correctamente acaba con la ejecucion del proceso que lo lleve a cabo. Por tanto, si falla el primero el segundo sera el correcto mientras que si el primero es el correcto acabara la funcion. Sucede lo mismo con execv

Sin embargo, estas funciones buscan el path automaticamente y con una sola ejecucion es suficiente. Sucede lo mismo con execvp

4.12.2.2 main()

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

Main del programa.

Generamos un proceso padre que a la vez invoca tantos procesos hijos como comandos se pasen. Estos procesos hijos ejecutan cada uno de los comandos con la funcion indicada con el parametro de entrada.

Parameters

argc	Numero de parametros de entrada incluyendo el nombre del programa
argv	Paramtreos de entrada

Returns

EXIT_SUCCESS o EXIT_FAILURE

4.13 Ejercicio9.c File Reference

Ejercicio 9.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
```

Include dependency graph for Ejercicio9.c:

Macros

- #define NUM_PROC 4

  Main.
- #define **MAX** 256

Functions

- int **main** (void)

4.13.1 Detailed Description

Ejercicio 9.

Author

Miguel Angel Sanchez y Juan Velasco

4.13.2 Macro Definition Documentation

4.13.2.1 NUM_PROC

```
#define NUM_PROC 4
```

Main.

Este main crea 4 procesos hijo que se encargan de hacer diferentes calculos. El padre obtiene los operandos por pantalla y se los pasa a los hijos utilizando tuberias. Los hijos devuelven el resultado al padre y este lo imprime.

EXIT_SUCCESS o EXIT_FAILURE

Index