

## Gestor de Memoria - Práctica 3 OS

Generado por Doxygen 1.15.0



<b>1 Índice de estructuras de datos</b>	<b>1</b>
1.1 Estructuras de datos	1
<b>2 Índice de archivos</b>	<b>3</b>
2.1 Lista de archivos	3
<b>3 Documentación de estructuras de datos</b>	<b>5</b>
3.1 Referencia de la estructura Memoria	5
3.1.1 Descripción detallada	5
3.1.2 Documentación de campos	5
3.1.2.1 cant_particiones	5
3.1.2.2 particiones	5
3.1.2.3 ultimo_indice_asignado	6
3.2 Referencia de la estructura Particion	6
3.2.1 Descripción detallada	6
3.2.2 Documentación de campos	6
3.2.2.1 dir_inicio	6
3.2.2.2 estado	6
3.2.2.3 nombre_proceso	6
3.2.2.4 tamano	7
3.3 Referencia de la estructura Proceso	7
3.3.1 Descripción detallada	7
3.3.2 Documentación de campos	7
3.3.2.1 en_memoria	7
3.3.2.2 finalizado	7
3.3.2.3 mem_requerida	8
3.3.2.4 nombre	8
3.3.2.5 t_ejecucion	8
3.3.2.6 t_final	8
3.3.2.7 t_llegada	8
3.3.2.8 t_restante	8
<b>4 Documentación de archivos</b>	<b>9</b>
4.1 Referencia del archivo src/ficheros.h	9
4.1.1 Descripción detallada	9
4.1.2 Documentación de funciones	9
4.1.2.1 cargar_procesos()	9
4.1.2.2 guardar_estado()	10
4.1.2.3 limpiar_log()	10
4.2 ficheros.h	11
4.3 Referencia del archivo src/sim_engine.h	11
4.3.1 Descripción detallada	12
4.3.2 Documentación de enumeraciones	12

---

4.3.2.1 TipoAlgo . . . . .	12
4.3.3 Documentación de funciones . . . . .	12
4.3.3.1 alinear_size() . . . . .	12
4.3.3.2 asignar_proceso() . . . . .	13
4.3.3.3 avanzar_tiempo() . . . . .	13
4.3.3.4 buscar_hueco() . . . . .	14
4.3.3.5 compactar() . . . . .	14
4.3.3.6 inicializar_memoria() . . . . .	14
4.3.3.7 liberar_proceso() . . . . .	15
4.3.3.8 mostrar_estado() . . . . .	15
4.3.3.9 ocupar_memoria() . . . . .	15
4.4 sim_engine.h . . . . .	16
<b>Índice alfabético</b>	<b>17</b>

# Capítulo 1

## Índice de estructuras de datos

### 1.1. Estructuras de datos

Lista de estructuras con breves descripciones:

<a href="#">Memoria</a>	Estructura que representa la memoria del sistema . . . . .	<a href="#">5</a>
<a href="#">Particion</a>	Que representa una partición de memoria . . . . .	<a href="#">6</a>
<a href="#">Proceso</a>	Estructura que representa un proceso . . . . .	<a href="#">7</a>



## Capítulo 2

# Índice de archivos

### 2.1. Lista de archivos

Lista de todos los archivos documentados y con breves descripciones:

src/ <a href="#">ficheros.h</a>	
Declaraciones para la gestión de ficheros en la simulación de memoria . . . . .	9
src/ <a href="#">sim_engine.h</a>	
Declaraciones del motor de simulación de gestión de memoria . . . . .	11





## Capítulo 3

# Documentación de estructuras de datos

### 3.1. Referencia de la estructura Memoria

Estructura que representa la memoria del sistema.

```
#include <sim_engine.h>
```

#### Campos de datos

- [Particion](#) [particiones](#) [[MAX\\_PARTICIONES](#)]
- int [cant\\_particiones](#)
- int [ultimo\\_indice\\_asignado](#)

#### 3.1.1. Descripción detallada

Estructura que representa la memoria del sistema.

#### 3.1.2. Documentación de campos

##### 3.1.2.1. [cant\\_particiones](#)

```
int Memoria::cant_particiones
```

Cantidad de particiones actuales

##### 3.1.2.2. [particiones](#)

```
Particion Memoria::particiones[MAX\_PARTICIONES]
```

Array de particiones de memoria

### 3.1.2.3. ultimo\_indice\_asignado

```
int Memoria::ultimo_indice_asignado
```

Último índice asignado en particiones

La documentación de esta estructura está generada del siguiente archivo:

- src/[sim\\_engine.h](#)

## 3.2. Referencia de la estructura Particion

que representa una partición de memoria.

```
#include <sim_engine.h>
```

### Campos de datos

- int [dir\\_inicio](#)
- int [tamano](#)
- int [estado](#)
- char [nombre\\_proceso](#) [10]

### 3.2.1. Descripción detallada

que representa una partición de memoria.

### 3.2.2. Documentación de campos

#### 3.2.2.1. dir\_inicio

```
int Particion::dir_inicio
```

Dirección de inicio de la partición

#### 3.2.2.2. estado

```
int Particion::estado
```

Estado de la partición 0 -> HUECO || 1 -> PROCESO

#### 3.2.2.3. nombre\_proceso

```
char Particion::nombre_proceso[10]
```

Nombre del proceso asignado o "HUECO" si está libre

#### 3.2.2.4. tamaño

```
int Particion::tamano
```

Tamaño de la partición

La documentación de esta estructura está generada del siguiente archivo:

- [src/sim\\_engine.h](#)

### 3.3. Referencia de la estructura Proceso

Estructura que representa un proceso.

```
#include <sim_engine.h>
```

#### Campos de datos

- char [nombre](#) [10]
- int [t\\_llegada](#)
- int [mem\\_requerida](#)
- int [t\\_ejecucion](#)
- int [t\\_final](#)
- int [t\\_restante](#)
- bool [en\\_memoria](#)
- bool [finalizado](#)

#### 3.3.1. Descripción detallada

Estructura que representa un proceso.

#### 3.3.2. Documentación de campos

##### 3.3.2.1. en\_memoria

```
bool Proceso::en_memoria
```

Indica si el proceso está en memoria

##### 3.3.2.2. finalizado

```
bool Proceso::finalizado
```

Indica si el proceso ha finalizado

#### 3.3.2.3. mem\_requerida

```
int Proceso::mem_requerida
```

Memoria requerida por el proceso

#### 3.3.2.4. nombre

```
char Proceso::nombre[10]
```

Nombre del proceso

#### 3.3.2.5. t\_ejecucion

```
int Proceso::t_ejecucion
```

Tiempo de ejecución del proceso

#### 3.3.2.6. t\_final

```
int Proceso::t_final
```

Tiempo de finalización del proceso

#### 3.3.2.7. t\_llegada

```
int Proceso::t_llegada
```

Tiempo de llegada del proceso

#### 3.3.2.8. t\_restante

```
int Proceso::t_restante
```

Tiempo restante de ejecución

La documentación de esta estructura está generada del siguiente archivo:

- [src/sim\\_engine.h](#)

## Capítulo 4

# Documentación de archivos

### 4.1. Referencia del archivo src/ficheros.h

Declaraciones para la gestión de ficheros en la simulación de memoria.

```
#include "sim_engine.h"
```

#### Funciones

- int [cargar\\_procesos](#) (const char \*ruta, [Proceso](#) procesos[])
- void [guardar\\_estado](#) (const char \*ruta, [Memoria](#) \*m, int instante)
- void [limpiar\\_log](#) (const char \*ruta)

#### 4.1.1. Descripción detallada

Declaraciones para la gestión de ficheros en la simulación de memoria.

Autor

Julian Hinojosa Gil

Fecha

2025

#### 4.1.2. Documentación de funciones

##### 4.1.2.1. cargar\_procesos()

```
int cargar_procesos (  
    const char * ruta,  
    Proceso procesos[])
```

Carga los procesos desde un archivo de texto.

**Parámetros**

---

<i>ruta</i>	Ruta del archivo de entrada
<i>procesos</i>	Array donde se almacenarán los procesos cargados

Devuelve

cantidad de procesos cargados

#### 4.1.2.2. guardar\_estado()

```
void guardar_estado (
    const char * ruta,
    Memoria * m,
    int instante)
```

Guarda el estado de la memoria en un archivo de texto.

##### Parámetros

<i>ruta</i>	Ruta del archivo de salida
<i>m</i>	Puntero a la estructura de memoria a inicializar
<i>instante</i>	Instante de tiempo actual

Devuelve

nada

#### 4.1.2.3. limpiar\_log()

```
void limpiar_log (
    const char * ruta)
```

Limpia el contenido de un archivo de texto.

##### Parámetros

<i>ruta</i>	Ruta del archivo a limpiar
-------------	----------------------------

Devuelve

nada

## 4.2. ficheros.h

[Ir a la documentación de este archivo.](#)

```
00001 #ifndef FICHEROS_H
00002 #define FICHEROS_H
00003
00004 #include "sim_engine.h"
00005
00012
00019 int cargar_procesos(const char* ruta, Proceso procesos[]);
00020
00028 void guardar_estado(const char* ruta, Memoria *m, int instante);
00029
00035 void limpiar_log(const char* ruta);
00036
00037 #endif // FICHEROS_H
```

## 4.3. Referencia del archivo src/sim\_engine.h

Declaraciones del motor de simulación de gestión de memoria.

```
#include <stdbool.h>
```

### Estructuras de datos

- struct [Proceso](#)  
*Estructura que representa un proceso.*
- struct [Particion](#)  
*que representa una partición de memoria.*
- struct [Memoria](#)  
*Estructura que representa la memoria del sistema.*

### defines

- #define **MEMORIA\_TOTAL** 2000  
*Tamaño total de la memoria simulada.*
- #define **UNIDAD\_MINIMA** 100  
*Unidad mínima de asignación de memoria.*
- #define **MAX\_PARTICIONES** 50  
*Máximo número de particiones en la memoria.*
- #define **MAX\_PROCESOS** 100  
*Máximo número de procesos en la simulación.*

### Enumeraciones

- enum [TipoAlgo](#) { [ALGO\\_PRIMER\\_HUECO](#) , [ALGO\\_SIGUIENTE\\_HUECO](#) }  
*Tipos de algoritmos de asignación de memoria.*

## Funciones

- void `inicializar_memoria` (`Memoria *m`)
- void `mostrar_estado` (`Memoria *m`)
- int `ocupar_memoria` (`Memoria *m`, int `indice_hueco`, `Proceso p`)
- void `compactar` (`Memoria *m`)
- bool `liberar_proceso` (`Memoria *m`, char `*nombre_proceso`)
- int `buscar_hueco` (`Memoria *m`, int `mem_requerida`, `TipoAlgo tipo_algo`)
- int `alinear_size` (int `size`)
- bool `asignar_proceso` (`Memoria *m`, `Proceso p`, `TipoAlgo tipo_algo`)
- void `avanzar_tiempo` (`Memoria *m`, `Proceso procesos[]`, int `num_procesos`, int `*reloj_actual`, `TipoAlgo algo`, const char `*ruta_log`)

### 4.3.1. Descripción detallada

Declaraciones del motor de simulación de gestión de memoria.

#### Autor

Julian Hinojosa Gil

#### Fecha

2025

### 4.3.2. Documentación de enumeraciones

#### 4.3.2.1. TipoAlgo

enum `TipoAlgo`

Tipos de algoritmos de asignación de memoria.

#### Valores de enumeraciones

ALGO_PRIMER_HUECO	Algoritmo de Primer Hueco
ALGO_SIGUIENTE_HUECO	Algoritmo de Siguiente Hueco

### 4.3.3. Documentación de funciones

#### 4.3.3.1. `alinear_size()`

```
int alinear_size (
    int size)
```

Alinea el tamaño solicitado a múltiplos de UNIDAD\_MINIMA (100).

#### Parámetros

---



<i>size</i>	Tamaño solicitado
-------------	-------------------

Devuelve

Tamaño alineado

#### 4.3.3.2. `asignar_proceso()`

```
bool asignar_proceso (  
    Memoria * m,  
    Proceso p,  
    TipoAlgo tipo_algo)
```

Asigna un proceso a la memoria según el algoritmo especificado.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
<i>p</i>	Proceso a asignar
<i>tipo_algo</i>	Algoritmo de búsqueda (primer hueco o siguiente hueco)

Devuelve

true si se asignó correctamente, false en caso de error

#### 4.3.3.3. `avanzar_tiempo()`

```
void avanzar_tiempo (  
    Memoria * m,  
    Proceso procesos[],  
    int num_procesos,  
    int * reloj_actual,  
    TipoAlgo algo,  
    const char * ruta_log)
```

Avanza el tiempo de la simulación, gestionando procesos y memoria.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
<i>procesos</i>	Array de procesos a gestionar
<i>num_procesos</i>	Número de procesos en el array
<i>reloj_actual</i>	Puntero al reloj actual de la simulación
<i>algo</i>	Algoritmo de asignación de memoria a utilizar

Devuelve

nada

#### 4.3.3.4. buscar\_hueco()

```
int buscar_hueco (  
    Memoria * m,  
    int mem_requerida,  
    TipoAlgo tipo_algo)
```

Busca un hueco adecuado según el algoritmo especificado.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
<i>mem_requerida</i>	Memoria requerida por el proceso
<i>tipo_algo</i>	Algoritmo de búsqueda (primer hueco o siguiente hueco)

##### Devuelve

Índice del hueco encontrado, o -1 si no se encontró ninguno

#### 4.3.3.5. compactar()

```
void compactar (  
    Memoria * m)
```

Compacta la memoria uniendo huecos adyacentes.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
----------	------------------------------------

##### Devuelve

nada

#### 4.3.3.6. inicializar\_memoria()

```
void inicializar_memoria (  
    Memoria * m)
```

Inicializa la estructura de memoria con un solo proceso de tipo hueco.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria a inicializar
----------	--

##### Devuelve

nada

#### 4.3.3.7. liberar\_proceso()

```
bool liberar_proceso (
    Memoria * m,
    char * nombre_proceso)
```

Libera un proceso de la memoria y compacta si es necesario.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
<i>nombre_proceso</i>	Nombre del proceso a liberar

##### Devuelve

true si se liberó correctamente, false si no se encontró el proceso

#### 4.3.3.8. mostrar\_estado()

```
void mostrar_estado (
    Memoria * m)
```

Muestra el estado actual de la memoria en consola.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
----------	------------------------------------

##### Devuelve

nada

#### 4.3.3.9. ocupar\_memoria()

```
int ocupar_memoria (
    Memoria * m,
    int indice_hueco,
    Proceso p)
```

Intenta ocupar un hueco de memoria con un proceso.

##### Parámetros

<i>m</i>	Puntero a la estructura de memoria
<i>indice_hueco</i>	Índice del hueco donde se intentará asignar el proceso

<i>p</i>	Proceso a asignar
----------	-------------------

Devuelve

0 si se asignó correctamente, 1 en caso de error

## 4.4. sim\_engine.h

[Ir a la documentación de este archivo.](#)

```

00001 #ifndef SIM_ENGINE_H
00002 #define SIM_ENGINE_H
00003
00004 #include <stdbool.h>
00005
00012
00014 #define MEMORIA_TOTAL 2000
00016 #define UNIDAD_MINIMA 100
00018 #define MAX_PARTICIONES 50
00020 #define MAX_PROCESOS 100
00021
00025 typedef struct {
00026     char nombre[10];
00027     int t_llegada;
00028     int mem_requerida;
00029     int t_ejecucion;
00030
00031     // Variables de control
00032     int t_final;
00033     int t_restante;
00034     bool en_memoria;
00035     bool finalizado;
00036 } Proceso;
00037
00041 typedef struct {
00042     int dir_inicio;
00043     int tamano;
00044     int estado;
00045     char nombre_proceso[10];
00046 } Particion;
00047
00051 typedef struct {
00052     Particion particiones[MAX_PARTICIONES];
00053     int cant_particiones;
00054     int ultimo_indice_asignado;
00055 } Memoria;
00056
00060 typedef enum {
00061     ALGO_PRIMER_HUECO,
00062     ALGO_SIGUIENTE_HUECO
00063 } TipoAlgo;
00064
00071 void inicializar_memoria(Memoria *m);
00072
00078 void mostrar_estado(Memoria *m);
00079
00087 int ocupar_memoria(Memoria *m, int indice_hueco, Proceso p);
00088
00094 void compactar(Memoria *m);
00095
00102 bool liberar_proceso(Memoria *m, char *nombre_proceso);
00103
00111 int buscar_hueco(Memoria *m, int mem_requerida, TipoAlgo tipo_algo);
00112
00118 int alinear_size(int size);
00119
00127 bool asignar_proceso(Memoria *m, Proceso p, TipoAlgo tipo_algo);
00128
00138 void avanzar_tiempo(Memoria *m, Proceso procesos[], int num_procesos, int *reloj_actual, TipoAlgo
    algo, const char* ruta_log);
00139
00140 #endif // ENGINE_H

```

# Índice alfabético

ALGO\_PRIMER\_HUECO  
    sim\_engine.h, 12

ALGO\_SIGUIENTE\_HUECO  
    sim\_engine.h, 12

alinear\_size  
    sim\_engine.h, 12

asignar\_proceso  
    sim\_engine.h, 13

avanzar\_tiempo  
    sim\_engine.h, 13

buscar\_hueco  
    sim\_engine.h, 13

cant\_particiones  
    Memoria, 5

cargar\_procesos  
    ficheros.h, 9

compactar  
    sim\_engine.h, 14

dir\_inicio  
    Particion, 6

en\_memoria  
    Proceso, 7

estado  
    Particion, 6

ficheros.h  
    cargar\_procesos, 9  
    guardar\_estado, 10  
    limpiar\_log, 10

finalizado  
    Proceso, 7

guardar\_estado  
    ficheros.h, 10

inicializar\_memoria  
    sim\_engine.h, 14

liberar\_proceso  
    sim\_engine.h, 14

limpiar\_log  
    ficheros.h, 10

mem\_requerida  
    Proceso, 7

Memoria, 5  
    cant\_particiones, 5

particiones, 5  
    ultimo\_indice\_asignado, 5

mostrar\_estado  
    sim\_engine.h, 15

nombre  
    Proceso, 8

nombre\_proceso  
    Particion, 6

ocupar\_memoria  
    sim\_engine.h, 15

Particion, 6  
    dir\_inicio, 6  
    estado, 6  
    nombre\_proceso, 6  
    tamano, 6

particiones  
    Memoria, 5

Proceso, 7  
    en\_memoria, 7  
    finalizado, 7  
    mem\_requerida, 7  
    nombre, 8  
    t\_ejecucion, 8  
    t\_final, 8  
    t\_llegada, 8  
    t\_restante, 8

sim\_engine.h  
    ALGO\_PRIMER\_HUECO, 12  
    ALGO\_SIGUIENTE\_HUECO, 12  
    alinear\_size, 12  
    asignar\_proceso, 13  
    avanzar\_tiempo, 13  
    buscar\_hueco, 13  
    compactar, 14  
    inicializar\_memoria, 14  
    liberar\_proceso, 14  
    mostrar\_estado, 15  
    ocupar\_memoria, 15  
    TipoAlgo, 12

src/ficheros.h, 9, 11

src/sim\_engine.h, 11, 16

t\_ejecucion  
    Proceso, 8

t\_final  
    Proceso, 8

t\_llegada  
    Proceso, [8](#)  
t\_restante  
    Proceso, [8](#)  
tamano  
    Particion, [6](#)  
TipoAlgo  
    sim\_engine.h, [12](#)  
  
ultimo\_indice\_asignado  
    Memoria, [5](#)