Gestor de Memoria - Práctica 3 OS

Generado por Doxygen 1.15.0

Capítulo 1

Índice de estructuras de datos

1.1. Estructuras de datos

Lista de estructuras con breves descripciones:

Capítulo 2

Índice de archivos

2.1. Lista de archivos

Lista de todos los archivos documentados y con breves descripciones:

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. tamano

```
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

ruta	Ruta del archivo de entrada
procesos	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

ruta	Ruta del archivo de salida
m	Puntero a la estructura de memoria a inicializar
instante	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

ruta	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

size	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

m	Puntero a la estructura de memoria
p	Proceso a asignar
tipo_algo	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

m	Puntero a la estructura de memoria
procesos	Array de procesos a gestionar
num_procesos	Número de procesos en el array
reloj_actual	Puntero al reloj actual de la simulación
algo	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

m	Puntero a la estructura de memoria
mem_requerida	Memoria requerida por el proceso
tipo_algo	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

m	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

m	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

m	Puntero a la estructura de memoria
nombre_proceso	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

m	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

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

p	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