Assegnamento Sistemi Operativi

Generato da Doxygen 1.9.6

Capitolo 1

LICENSE

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "{}" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright {yyyy} {name of copyright owner}

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

`http://www.apache.org/licenses/LICENSE-2.0`

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Capitolo 2

Progetto_Sistemi_Operativi

This project aims to simulate CPU scheduling, starting from a file containing processes there are several algorithms with different degrees of optimization that virtually execute the processes in different order depending on the chosen algorithm

Capitolo 3

Indice dei tipi composti

3.1 Elenco dei tipi composti

Queste sono le classi, le struct, le union e le interfacce con una loro breve descrizione:

Capitolo 4

Indice dei file

4.1 Elenco dei file

Questo è un elenco di tutti i file con una loro breve descrizione:

Capitolo 5

Documentazione delle classi

5.1 Riferimenti per la struct Processo

Struct creata per salvare il nome, la durata e la priorità del processo.

`#include <struct.h>`

Attributi pubblici

- string nome
- int istante_arrivo
- int durata
- int priorita

5.1.1 Descrizione dettagliata

Struct creata per salvare il nome, la durata e la priorità del processo.

Definizione alla linea 14 del file struct.h.

5.1.2 Documentazione dei membri dato

5.1.2.1 durata

`int Processo::durata`

Definizione alla linea 17 del file struct.h.

5.1.2.2 istante_arrivo

```
int Processo::istante_arrivo
```

Definizione alla linea 16 del file struct.h.

5.1.2.3 nome

```
string Processo::nome
```

Definizione alla linea 15 del file struct.h.

5.1.2.4 priorita

```
int Processo::priorita
```

Definizione alla linea 18 del file struct.h.

La documentazione per questa struct è stata generata a partire dal seguente file:

- struct.h

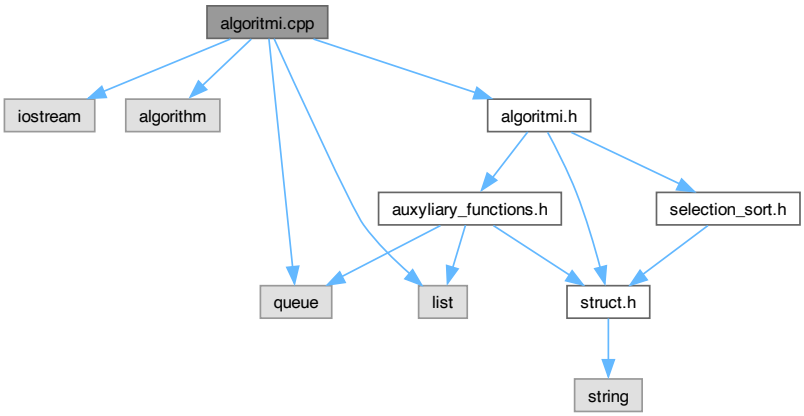
Capitolo 6

Documentazione dei file

6.1 Riferimenti per il file algoritmi.cpp

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <list>
#include "algoritmi.h"
```

Grafo delle dipendenze di inclusione per algoritmi.cpp:



Funzioni

- void algoritmo_FCFS (Processo *p, int num_processi)
- void algoritmo_priorita (Processo *p, int num_processi)
- void algoritmo_BJP (Processo *p, int num_processi)
- void algoritmo_RR (Processo *p, int num_processi, int quanto)
- void algoritmo_SRTF (Processo *p, int num_processi)

Variabili

- const int CONST = 100

6.1.1 Documentazione delle funzioni

6.1.1.1 algoritmo_BJP()

```
void algoritmo_BJP (
            Processo * p,
            int num_processi )
```

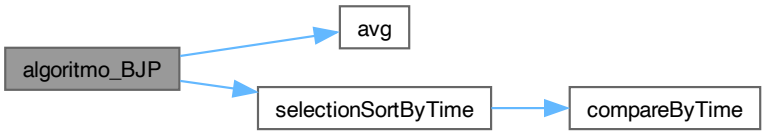
Definizione dell'algoritmo BJF

Parametri

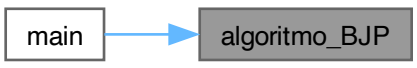
Processo	p
int	num_processi

Definizione alla linea 59 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



6.1.1.2 algoritmo_FCFS()

```
void algoritmo_FCFS (
            Processo * p,
            int num_processi )
```

Definizione dell'algoritmo FCFS

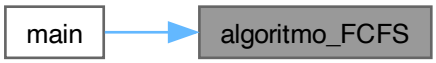
Parametri

Processo	p
int	num_processi

Definizione alla linea 22 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:



6.1.1.3 algoritmo_priorita()

```
void algoritmo_priorita (
            Processo * p,
            int num_processi )
```

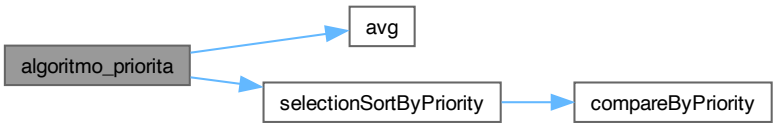
Definizione dell'algoritmo "Priorità"

Parametri

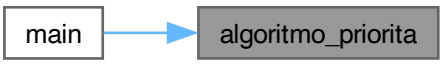
Processo	p
int	num_processi

Definizione alla linea 40 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



6.1.1.4 algoritmo_RR()

```
void algoritmo_RR (
            Processo * p,
            int num_processi,
            int quanto )
```

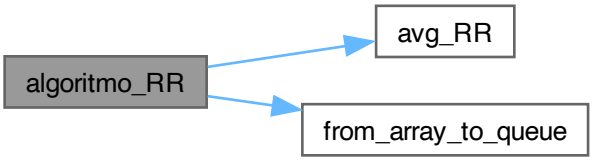
Definizione dell'algoritmo RR

Parametri

Processo	p
int	num_processi
int	quanto

Definizione alla linea 80 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.1.1.5 algoritmo_SRTF()

```
void algoritmo_SRTF (
            Processo * p,
            int num_processi )
```

Definizione dell'algoritmo SRTF

Parametri

Processo	p
int	num_processi

in caso ci siano più processi in arrivo nello stesso istante viene creata una lista di processi ordinata per durata, quindi il primo elemento della lista sarà quello con durata minore

Andiamo ad inserire nella lista tutti i processi che entrano in giorco in quell'isante di tempo

Una volta che tutti i processi sono stati inseriti andiamo ad ordinare la lista in modo tale da avere come primo elemento quello con durata minore

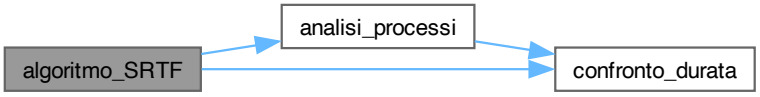
Ad ogni iterazione viene decrementata la durata del processo "front" della lista

Quando la durata di un processo arriva a 0 lo andiamo a togliere dalla lista

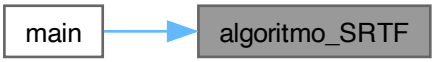
L'algoritmo termina quando la lista è vuota

Definizione alla linea 119 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



6.1.2 Documentazione delle variabili

6.1.2.1 CONST

```
const int CONST = 100
```

Definizione alla linea 14 del file algoritmi.cpp.

6.2 algoritmi.cpp

Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 04/04/23.
00003 //
00004
00005 #include <iostream>
00006 #include <algorithm>
00007 #include <queue>
00008 #include <list>
00009 #include "algoritmi.h"
00010
00011
```

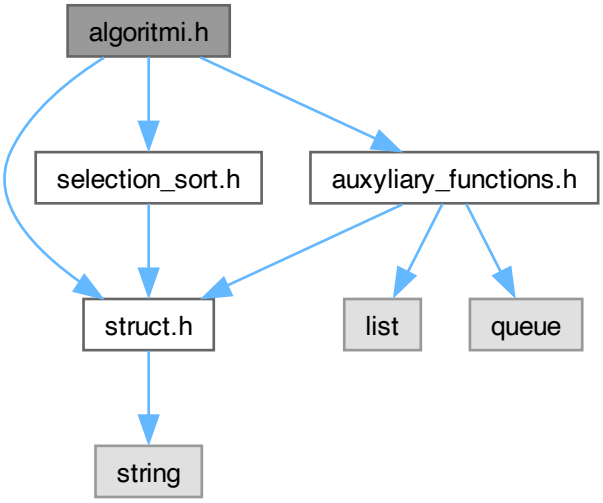
```
00012 using namespace std;
00013
00014 const int CONST = 100;
00015
00016
00022 void algoritmo_FCFS (Processo *p, int num_processi) {
00023     int n = CONST;
00024     int *array_durata = new int[n];
00025     cout « "FCFS ";
00026     for (int i = 0; i < num_processi; i++) {
00027         cout « "->" « p[i].nome;
00028         array_durata[i] = p[i].durata;
00029     }
00030     cout « endl « "TEMPO MEDIO " « avg(array_durata, num_processi) « endl;
00031     delete [] array_durata;
00032 }
00033
00034
00040 void algoritmo_priorita (Processo *p, int num_processi) {
00041     int n = CONST;
00042     int *array_durata = new int [n];
00043     selectionSortByPriority(p, num_processi);
00044     cout « "PRIORITÀ ";
00045     for (int i = 0; i < num_processi; i++) {
00046         cout « "->" « p[i].nome;
00047         array_durata[i] = p[i].durata;
00048     }
00049     cout « endl « "TEMPO MEDIO " « avg(array_durata, num_processi) « endl;
00050     delete [] array_durata;
00051 }
00052
00053
00059 void algoritmo_BJP (Processo *p, int num_processi) {
00060     int n = CONST;
00061     int *array_durata = new int [n];
00062     selectionSortByTime(p, num_processi);
00063     cout « "BJP ";
00064     for (int i = 0; i < num_processi; i++) {
00065         cout « "->" « p[i].nome;
00066         array_durata[i] = p[i].durata;
00067     }
00068     cout « endl « "TEMPO MEDIO " « avg(array_durata, num_processi) « endl;
00069     delete [] array_durata;
00070 }
00071
00072
00080 void algoritmo_RR (Processo *p, int num_processi, int quanto) {
00081     int array_counter = 0;
00082     int n = CONST;
00083     cout « "RR ";
00084     Processo *array_durata = new Processo [n];
00085     queue<Processo> processi = from_array_to_queue(p, num_processi);
00086     while (not processi.empty()) {
00087         if (processi.front().durata <= quanto) {
00088             cout « "->" « processi.front().nome;
00089             array_durata[array_counter] = processi.front();
00090             processi.pop();
00091             array_counter++;
00092         }
00093         else if (processi.front().durata >= quanto){
00094             Processo temp;
00095             temp.nome = processi.front().nome;
00096             temp.durata = processi.front().durata - quanto;
00097             temp.priorita = processi.front().priorita;
00098             cout « "->" « temp.nome;
00099             Processo durata;
00100             durata.nome = processi.front().nome;
00101             durata.durata = quanto;
00102             durata.priorita = processi.front().priorita;
00103             array_durata[array_counter] = durata;
00104             processi.push(temp);
00105             processi.pop();
00106             array_counter++;
00107         }
00108     }
00109     cout « endl « "TEMPO MEDIO " « avg_RR(array_durata, array_counter, num_processi) « endl;
00110     delete [] array_durata;
00111 }
00112
00113
00119 void algoritmo_SRTF (Processo *p, int num_processi) {
00120     int counter = 0; int time = -1; Processo temp;
00121     list<Processo> lista;
00122     list<Processo> processi_analizzati;
00123     bool flag = false;
00124     cout « "SRTF ";
00125     while (flag == false) {
```

```
00126         time++;
00127
00135         processi_analizzati = analisi_processi(p, num_processi, time);
00136
00142         while (not processi_analizzati.empty()) {
00143             lista.push_front(processi_analizzati.front());
00144             processi_analizzati.pop_front();
00145         }
00146
00153         lista.sort(confronto_durata);
00154
00156
00157         lista.front().durata--;
00158         //cout « "TIME " « time « endl;
00159         //cout « "Primo processo della lista " « lista.front().nome « " DURATA " «
      lista.front().durata « endl;
00160
00166         if (lista.front().durata == 0) {
00167             cout « "->" « lista.front().nome;
00168             lista.pop_front();
00169             //cout « "NUOVO FRONT " « lista.front().nome « endl;
00170         }
00171
00173
00174         if (lista.empty()) { flag = true; }
00175     }
00176 }
```

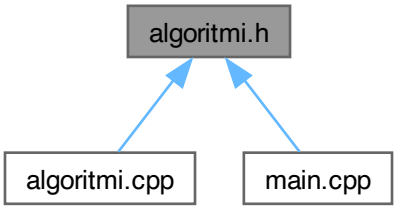
6.3 Riferimenti per il file algoritmi.h

```
#include "struct.h"
#include "selection_sort.h"
#include "auxyliary_functions.h"
```

Grafo delle dipendenze di inclusione per algoritmi.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Funzioni

- void algoritmo_FCFS (Processo *p, int num_processi)
- void algoritmo_priorita (Processo *p, int num_processi)
- void algoritmo_BJP (Processo *p, int num_processi)
- void algoritmo_RR (Processo *p, int num_processi, int quanto)
- void algoritmo_SRTF (Processo *p, int num_processi)

6.3.1 Documentazione delle funzioni

6.3.1.1 algoritmo_BJP()

```
void algoritmo_BJP (
            Processo * p,
            int num_processi )
```

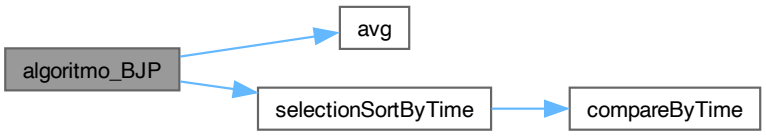
Definizione dell'algoritmo BJF

Parametri

Processo	p
int	num_processi

Definizione alla linea 59 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.3.1.2 algoritmo_FCFS()

```
void algoritmo_FCFS (
            Processo * p,
            int num_processi )
```

Definizione dell'algoritmo FCFS

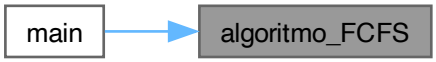
Parametri

Processo	p
int	num_processi

Definizione alla linea 22 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:



6.3.1.3 algoritmo_priorita()

```
void algoritmo_priorita (
            Processo * p,
            int num_processi )
```

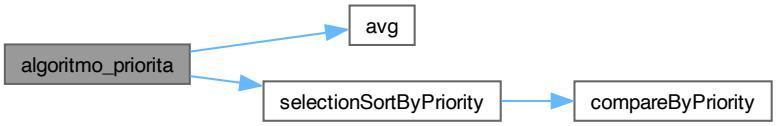
Definizione dell'algoritmo "Priorità"

Parametri

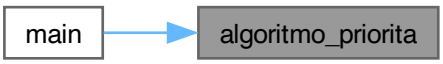
Processo	p
int	num_processi

Definizione alla linea 40 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



6.3.1.4 algoritmo_RR()

```
void algoritmo_RR (
            Processo * p,
            int num_processi,
            int quanto )
```

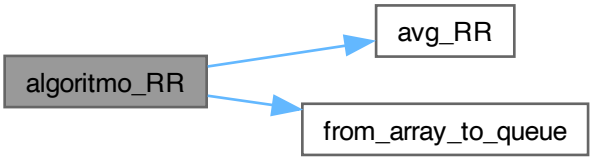
Definizione dell'algoritmo RR

Parametri

Processo	p
int	num_processi
int	quanto

Definizione alla linea 80 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.3.1.5 algoritmo_SRTF()

```
void algoritmo_SRTF (
            Processo * p,
            int num_processi )
```

Definizione dell'algoritmo SRTF

Parametri

Processo	p
int	num_processi

in caso ci siano più processi in arrivo nello stesso istante viene creata una lista di processi ordinata per durata, quindi il primo elemento della lista sarà quello con durata minore

Andiamo ad inserire nella lista tutti i processi che entrano in giorco in quell'isante di tempo

Una volta che tutti i processi sono stati inseriti andiamo ad ordinare la lista in modo tale da avere come primo elemento quello con durata minore

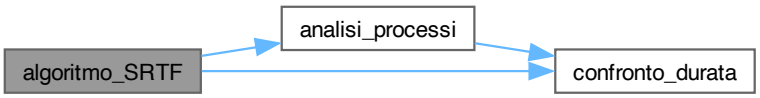
Ad ogni iterazione viene decrementata la durata del processo "front" della lista

Quando la durata di un processo arriva a 0 lo andiamo a togliere dalla lista

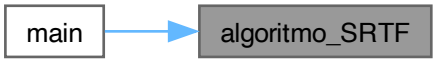
L'algoritmo termina quando la lista è vuota

Definizione alla linea 119 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



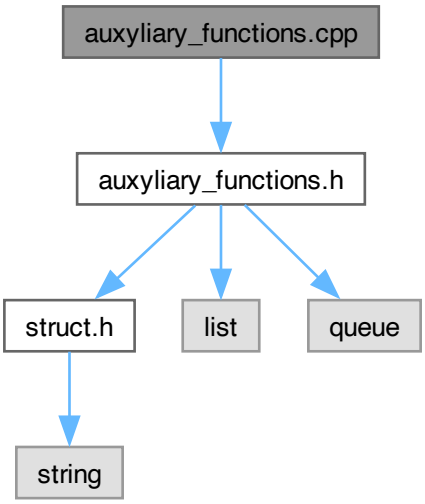
6.4 algoritmi.h

Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 04/04/23.
00003 //
00004
00005 #ifndef PROGETTO_SISTEMI_OPERATIVI_ALGORITMI_H
00006 #define PROGETTO_SISTEMI_OPERATIVI_ALGORITMI_H
00007
00008 #include "struct.h"
00009 #include "selection_sort.h"
00010 #include "auxyliary_functions.h"
00011
00012 void algoritmo_FCFS (Processo *p, int num_processi);
00013 void algoritmo_priorita (Processo *p, int num_processi);
00014 void algoritmo_BJP (Processo *p, int num_processi);
00015 void algoritmo_RR (Processo *p, int num_processi, int quanto);
00016 void algoritmo_SRTF (Processo *p, int num_processi);
00017
00018 #endif //PROGETTO_SISTEMI_OPERATIVI_ALGORITMI_H
```

6.5 Riferimenti per il file auxyliary_functions.cpp

```
#include "auxyliary_functions.h"
```
Grafo delle dipendenze di inclusione per auxyliary_functions.cpp:



Funzioni

- list< Processo > analisi_processi (Processo *p, int num_processi, int time)
- bool confronto_durata (const Processo &a, const Processo &b)
- queue< Processo > from_array_to_queue (Processo *p, int num_processi)
- float avg (int *durata, int size)
- float avg_RR (Processo *durata, int size, int num_processi)

6.5.1 Documentazione delle funzioni

6.5.1.1 analisi_processi()

```
list< Processo > analisi_processi (
            Processo * p,
            int num_processi,
            int time )
```

Questa funzione viene chiamata nell'esecuzione dell'algoritmo SRTF e permette di analizzare quali processi vengono chiamati in un particolare istante di tempo e li ordina in base alla durata

Parametri

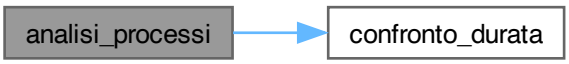
Processo	p
int	num_processi
int	time

Restituisce

list<Processo> processi_in_arrivo

Definizione alla linea 17 del file auxyliary_functions.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.5.1.2 avg()

```
float avg (
            int * durata,
            int size )
```

Funzione di calcolo del tempo medio

Parametri

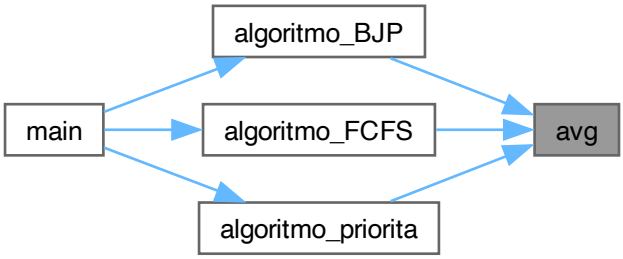
int	durata
int	size

Restituisce

float avg

Definizione alla linea 65 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.3 avg_RR()

```
float avg_RR (
            Processo * durata,
            int size,
            int num_processi )
```

Funzione di calcolo del tempo medio per l'algoritmo Round Robin

Parametri

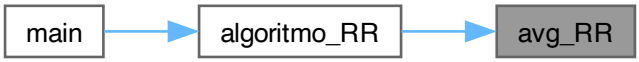
Processo	durata
int	size
int	num_processi

Restituisce

float avg

Definizione alla linea 90 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.4 confronto_durata()

```
bool confronto_durata (
            const Processo & a,
            const Processo & b )
```

Insieme alla funzione "analisi_processi" permette l'ordinamento dei processi in base alla durata

Parametri

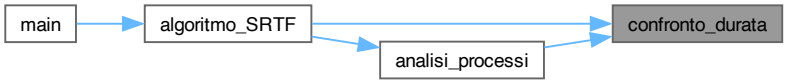
Processo	a
Processo	b

Restituisce

bool

Definizione alla linea 36 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.5 from_array_to_queue()

```
queue< Processo > from_array_to_queue (
            Processo * p,
            int num_processi )
```

Creazione di una funzione che trasforma un array in una coda

Parametri

Processo	p
int	num_processi

Restituisce

queue<Processo> coda

Definizione alla linea 46 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.6 auxyliary_functions.cpp

Vai alla documentazione di questo file.

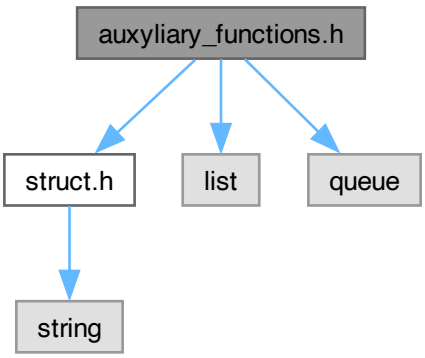
```
00001 //
00002 // Created by Matteo Franchini on 25/04/23.
00003 //
00004
00005 #include "auxyliary_functions.h"
00006
00017 list<Processo> analisi_processi (Processo *p, int num_processi, int time) {
00018     list<Processo> processi_in_arrivo;
00019     for (int i = 0; i < num_processi; i++) {
00020         if (p[i].istante_arrivo == time) {
00021             processi_in_arrivo.push_back(p[i]);
00022             processi_in_arrivo.sort(confronto_durata);
00023         }
00024     }
00025     return processi_in_arrivo;
00026 }
00027
00036 bool confronto_durata (const Processo& a, const Processo& b) {
00037     return a.durata < b.durata;
00038 }
00039
00046 queue<Processo> from_array_to_queue(Processo *p, int num_processi) {
00047     queue<Processo> var;
00048     for (int i = 0; i < num_processi; i++) {
00049         Processo temp;
00050         temp.nome = p[i].nome;
00051         temp.durata = p[i].durata;
00052         temp.priorita = p[i].priorita;
00053         var.push(temp);
00054     }
00055     return var;
00056 }
00057
00058
00065 float avg (int *durata, int size) {
00066     // Tempo di attesa del singolo processo
00067
00068     int sum_int = 0;
00069
00070     // Tempo di attesa complessivo
00071
00072     int sum = 0;
```

```
00073
00074     for (int j = 0; j < size-1; j++) {
00075         sum_int += durata[j];
00076         sum += sum_int;
00077     }
00078     return sum/size;
00079 }
00080
00081
00090 float avg_RR (Processo *durata, int size, int num_processi) {
00091     bool flag = false;
00092     int sum = 0;
00093     for (int i = 0; i < num_processi; i++) {
00094         for (int j = size; j >= 0; j--) {
00095             if (durata[i].nome == durata[j].nome) {
00096                 flag = true;
00097             }
00098             if (flag == true && durata[i].nome != durata[j].nome) {
00099                 sum += durata[j].durata;
00100             }
00101         }
00102         flag = false;
00103     }
00104     return sum/num_processi;
00105 }
```

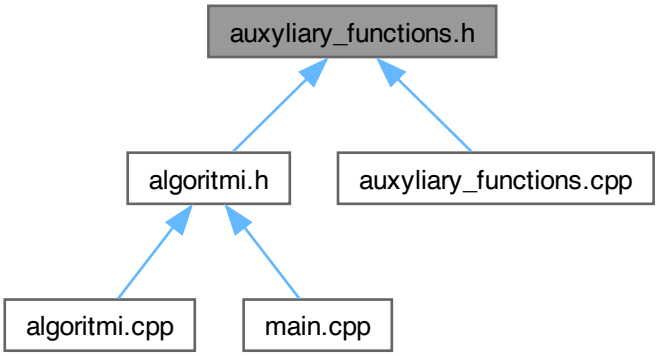
6.7 Riferimenti per il file auxyliary_functions.h

```
#include "struct.h"
#include <list>
#include <queue>
```

Grafo delle dipendenze di inclusione per auxyliary_functions.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Funzioni

- list< Processo > analisi_processi (Processo *p, int num_processi, int time)
- bool confronto_durata (const Processo &a, const Processo &b)
- queue< Processo > from_array_to_queue (Processo *p, int num_processi)
- float avg (int *durata, int size)
- float avg_RR (Processo *durata, int size, int num_processi)

6.7.1 Documentazione delle funzioni

6.7.1.1 analisi_processi()

```
list< Processo > analisi_processi (
            Processo * p,
            int num_processi,
            int time )
```

Questa funzione viene chiamata nell'esecuzione dell'algoritmo SRTF e permette di analizzare quali processi vengono chiamati in un particolare istante di tempo e li ordina in base alla durata

Parametri

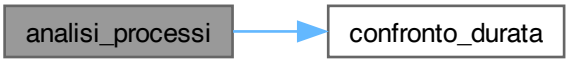
Processo	p
int	num_processi
int	time

Restituisce

list<Processo> processi_in_arrivo

Definizione alla linea 17 del file auxyliary_functions.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.7.1.2 avg()

```
float avg (
            int * durata,
            int size )
```

Funzione di calcolo del tempo medio

Parametri

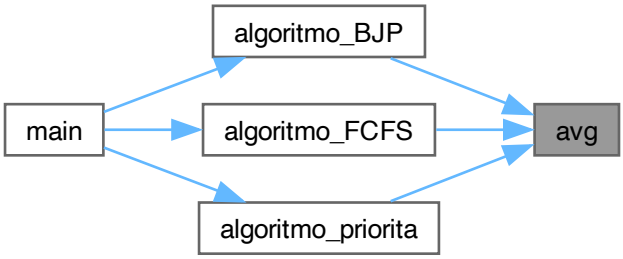
int	durata
int	size

Restituisce

float avg

Definizione alla linea 65 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.3 avg_RR()

```
float avg_RR (
            Processo * durata,
            int size,
            int num_processi )
```

Funzione di calcolo del tempo medio per l'algoritmo Round Robin

Parametri

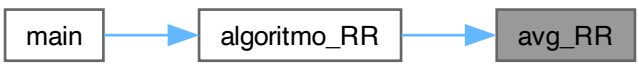
Processo	durata
int	size
int	num_processi

Restituisce

float avg

Definizione alla linea 90 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.4 confronto_durata()

```
bool confronto_durata (
            const Processo & a,
            const Processo & b )
```

Insieme alla funzione "analisi_processi" permette l'ordinamento dei processi in base alla durata

Parametri

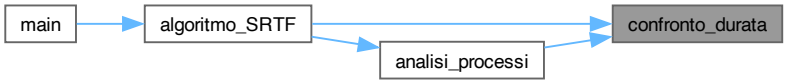
Processo	a
Processo	b

Restituisce

bool

Definizione alla linea 36 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.5 from_array_to_queue()

```
queue< Processo > from_array_to_queue (
            Processo * p,
            int num_processi )
```

Creazione di una funzione che trasforma un array in una coda

Parametri

Processo	p
int	num_processi

Restituisce

queue<Processo> coda

Definizione alla linea 46 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.8 auxyliary_functions.h

Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 25/04/23.
00003 //
00004
00005 #ifndef PROGETTO_SISTEMI_OPERATIVI_AUXYLIARY_FUNCTIONS_H
00006 #define PROGETTO_SISTEMI_OPERATIVI_AUXYLIARY_FUNCTIONS_H
00007
00008 #include "struct.h"
00009 #include <list>
00010 #include <queue>
00011
00012 list<Processo> analisi_processi (Processo *p, int num_processi, int time);
00013 bool confronto_durata (const Processo& a, const Processo& b);
00014 queue<Processo> from_array_to_queue(Processo *p, int num_processi);
00015 float avg (int *durata, int size);
00016 float avg_RR (Processo *durata, int size, int num_processi);
00017
00018 #endif //PROGETTO_SISTEMI_OPERATIVI_AUXYLIARY_FUNCTIONS_H
```

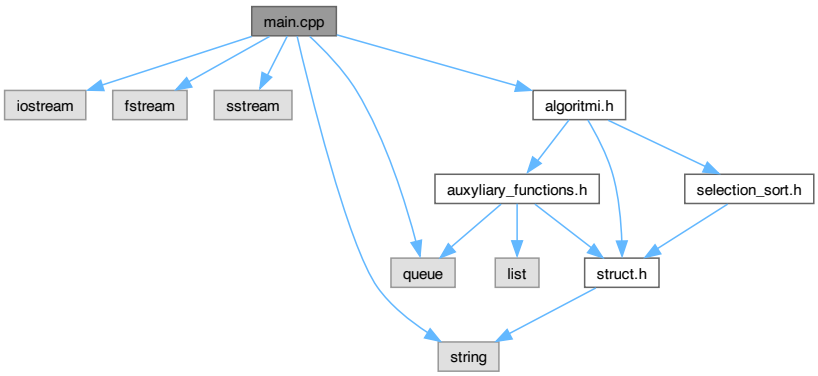
6.9 Riferimenti per il file LICENSE.md

6.10 Riferimenti per il file main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <queue>
```

```
#include "algoritmi.h"
```

Grafo delle dipendenze di inclusione per main.cpp:



Funzioni

- int main (int argc, char ∗argv[ ])

Variabili

- const int CONST = 100

6.10.1 Documentazione delle funzioni

6.10.1.1 main()

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

Controllo del numero di argomenti passati da riga di comando

Se il numero di argomenti è corretto, si procede con l'esecuzione del programma

Inizializzazione delle variabili

Creazione di un array di tipo Processo con allocazione dinamica

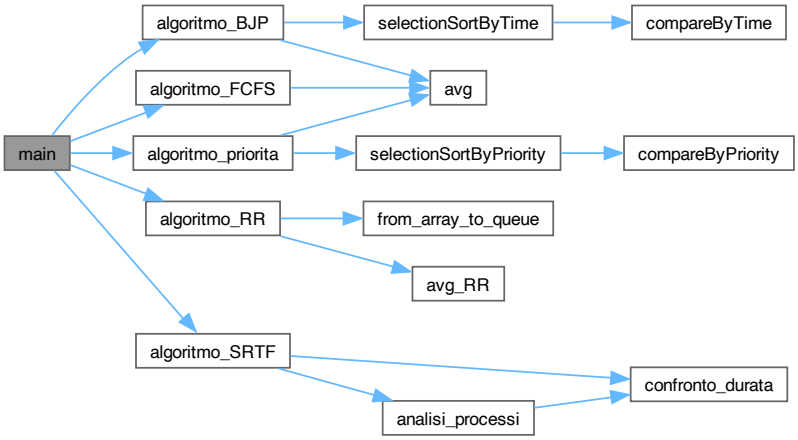
Lettura da file

Scelta dell'algoritmo

Cancellazione dell'array con allocazione dinamica

Definizione alla linea 16 del file main.cpp.

Questo è il grafo delle chiamate per questa funzione:



6.10.2 Documentazione delle variabili

6.10.2.1 CONST

```
const int CONST = 100
```

Definizione alla linea 14 del file main.cpp.

6.11 main.cpp

Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 03/04/23.
00003 //
00004
00005 #include <iostream>
00006 #include <fstream>
00007 #include <sstream>
00008 #include <string>
00009 #include <queue>
00010 #include "algoritmi.h"
00011
00012 using namespace std;
00013
00014 const int CONST = 100;
00015
00016 int main(int argc, char *argv[]) {
00017
00019     if (argc != 2) {
00020         cout « "Errore: inserire il nome del file di input" « endl;
00021         return 1;
00022     }
00023
00025     else {
```

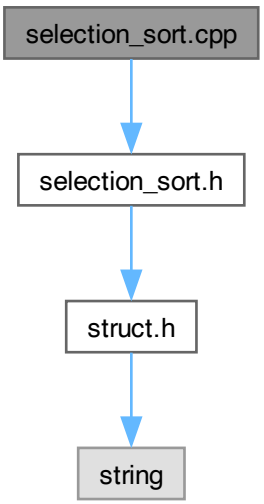
```
00026
00028     string nome; int durata; int priorita; int istante_arrivo;
00029     string str1; string str2; string str3; string str4; string str5;
00030     int num_processi;
00031     int quanto;
00032     int counter = -1;
00033     int scelta_algoritmo = 0;
00034
00035
00037     int n = CONST;
00038     Processo* arr = new Processo[n];
00039
00041     ifstream myfile (argv[1]);
00042
00043     if (myfile.is_open()) {
00044         while (getline(myfile, str5, ' ')) {
00045             if (counter == -1) {
00046                 getline(myfile, nome, ' ');
00047                 getline(myfile, str3, ' ');
00048                 istringstream tk3 (str3);
00049                 tk3 » num_processi;
00050                 getline(myfile, str4);
00051                 istringstream tk4 (str4);
00052                 tk4 » quanto;
00053                 counter++;
00054             }
00055             else {
00056                 istringstream tk5(str5);
00057                 tk5 » istante_arrivo;
00058                 getline(myfile, nome, ' ');
00059                 getline(myfile, str1, ' ');
00060                 istringstream tk1(str1);
00061                 tk1 » durata;
00062                 getline(myfile, str2);
00063                 istringstream tk2(str2);
00064                 tk2 » priorita;
00065                 arr[counter].nome = nome;
00066                 arr[counter].istante_arrivo = istante_arrivo;
00067                 arr[counter].durata = durata;
00068                 arr[counter].priorita = priorita;
00069                 counter++;
00070             }
00071         }
00072         myfile.close();
00073     } else cout « "Impossibile aprire il file";
00074
00076     cout « "Selezionare l'algoritmo che si intende utilizzare" « endl;
00077     cout « "Premere 1 per FCFS" « endl;
00078     cout « "Premere 2 per BJP" « endl;
00079     cout « "Premere 3 per Priorità" « endl;
00080     cout « "Premere 4 per RR" « endl;
00081     cout « "Premere 5 per SRTF" « endl;
00082     cin » scelta_algoritmo;
00083
00084     switch (scelta_algoritmo) {
00085         case 1:
00086             cout « "Hai scelto l'algoritmo FCFS" « endl;
00087             algoritmo_FCFS(arr, num_processi);
00088             break;
00089
00090         case 2:
00091             cout « "Hai scelto l'algoritmo BJP" « endl;
00092             algoritmo_BJP(arr, num_processi);
00093             break;
00094
00095         case 3:
00096             cout « "Hai scelto l'algoritmo Priorità" « endl;
00097             algoritmo_priorita(arr, num_processi);
00098             break;
00099
00100         case 4:
00101             cout « "Hai scelto l'algoritmo RR" « endl;
00102             algoritmo_RR(arr, num_processi, quanto);
00103             break;
00104
00105         case 5:
00106             cout « "Hai scelto l'algoritmo SRTF" « endl;
00107             algoritmo_SRTF(arr, num_processi);
00108             break;
00109     }
00110
00111
00113     delete[] arr;
00114
00115     return 0;
00116     }
00117 }
```

6.12 Riferimenti per il file README.md

6.13 Riferimenti per il file selection_sort.cpp

```
#include "selection_sort.h"
```

Grafo delle dipendenze di inclusione per selection_sort.cpp:



Funzioni

- bool compareByPriority (Processo a, Processo b)

  Funzioni per ordinare gli elementi in base alla priorità
- void selectionSortByPriority (Processo ∗arr, int size)

  Algoritmo selection sort che opera in base alla priorità secondo la funzione "compareByPriority".
- bool compareByTime (Processo a, Processo b)

  Funzione per confrontare i processi in base alla durata.
- void selectionSortByTime (Processo ∗arr, int size)

  Algoritmo selection sort che ordina in base alla durata secondo la funzione "compareByTime".

6.13.1 Documentazione delle funzioni

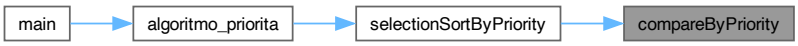
6.13.1.1 compareByPriority()

```
bool compareByPriority (
            Processo a,
            Processo b )
```

Funzioni per ordinare gli elementi in base alla priorità

Definizione alla linea 8 del file selection_sort.cpp.

Questo è il grafo dei chiamanti di questa funzione:



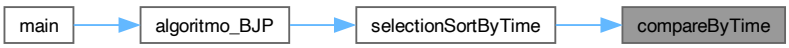
6.13.1.2 compareByTime()

```
bool compareByTime (
            Processo a,
            Processo b )
```

Funzione per confrontare i processi in base alla durata.

Definizione alla linea 27 del file selection_sort.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.13.1.3 selectionSortByPriority()

```
void selectionSortByPriority (
            Processo * arr,
            int size )
```

Algoritmo selection sort che opera in base alla priorità secondo la funzione "compareByPriority".

Definizione alla linea 13 del file selection_sort.cpp.

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:

6.13.1.4 selectionSortByTime()

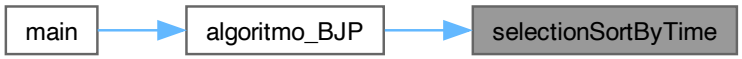
```
void selectionSortByTime (
            Processo * arr,
            int size )
```

Algoritmo selection sort che ordina in base alla durata secondo la funzione "compareByTime".

Definizione alla linea 32 del file selection_sort.cpp.

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:



6.14 selection_sort.cpp

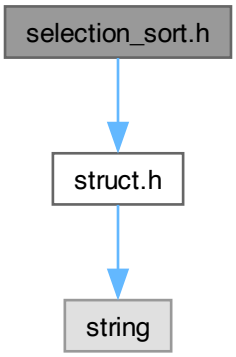
Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 06/04/23.
00003 //
00004
00005 #include "selection_sort.h"
00006
00008 bool compareByPriority(Processo a, Processo b) {
00009     return a.priorita < b.priorita;
00010 }
00011
00013 void selectionSortByPriority(Processo *arr, int size) {
00014     int i, j, min_idx;
00015     for (i = 0; i < size - 1; i ++) {
00016         min_idx = i;
00017         for (j = i + 1; j < size; j++) {
00018             if (compareByPriority(arr[j], arr[min_idx])) {
00019                 min_idx = j;
00020             }
00021         }
00022         swap(arr[min_idx], arr[i]);
00023     }
00024 }
00025
00027 bool compareByTime(Processo a, Processo b) {
00028     return a.durata < b.durata;
00029 }
00030
00032 void selectionSortByTime(Processo *arr, int size) {
00033     int i, j, min_idx;
00034     for (i = 0; i < size - 1; i ++) {
00035         min_idx = i;
00036         for (j = i + 1; j < size; j++) {
00037             if (compareByTime(arr[j], arr[min_idx])) {
00038                 min_idx = j;
00039             }
00040         }
00041         swap(arr[min_idx], arr[i]);
00042     }
00043 }
```

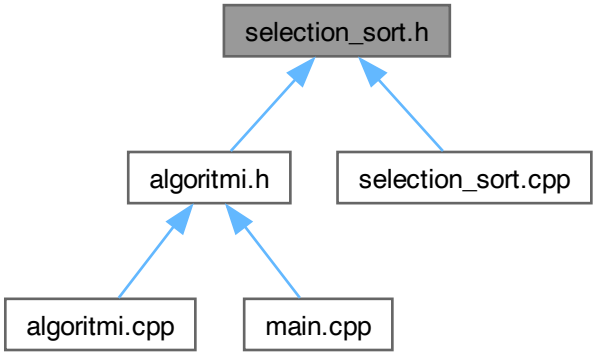
6.15 Riferimenti per il file selection_sort.h

```
#include "struct.h"
```

Grafo delle dipendenze di inclusione per selection_sort.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Funzioni

- void selectionSortByPriority (Processo *arr, int size)

  Algoritmo selection sort che opera in base alla priorità secondo la funzione "compareByPriority".
- void selectionSortByTime (Processo *arr, int size)

  Algoritmo selection sort che ordina in base alla durata secondo la funzione "compareByTime".

6.15.1 Documentazione delle funzioni

6.15.1.1 selectionSortByPriority()

```
void selectionSortByPriority (
            Processo * arr,
            int size )
```

Algoritmo selection sort che opera in base alla priorità secondo la funzione "compareByPriority".

Definizione alla linea 13 del file selection_sort.cpp.

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:

6.15.1.2 selectionSortByTime()

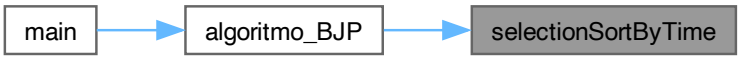
```
void selectionSortByTime (
            Processo * arr,
            int size )
```

Algoritmo selection sort che ordina in base alla durata secondo la funzione "compareByTime".

Definizione alla linea 32 del file selection_sort.cpp.

Questo è il grafo delle chiamate per questa funzione:

Questo è il grafo dei chiamanti di questa funzione:



6.16 selection_sort.h

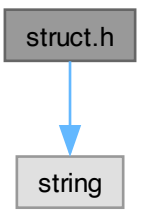
Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 06/04/23.
00003 //
00004
00005 #ifndef PROGETTO_SISTEMI_OPERATIVI_SELECTION_SORT_H
00006 #define PROGETTO_SISTEMI_OPERATIVI_SELECTION_SORT_H
00007
00008 #include "struct.h"
00009
00010 void selectionSortByPriority(Processo *arr, int size);
00011 void selectionSortByTime(Processo *arr, int size);
00012
00013 #endif //PROGETTO_SISTEMI_OPERATIVI_SELECTION_SORT_H
```

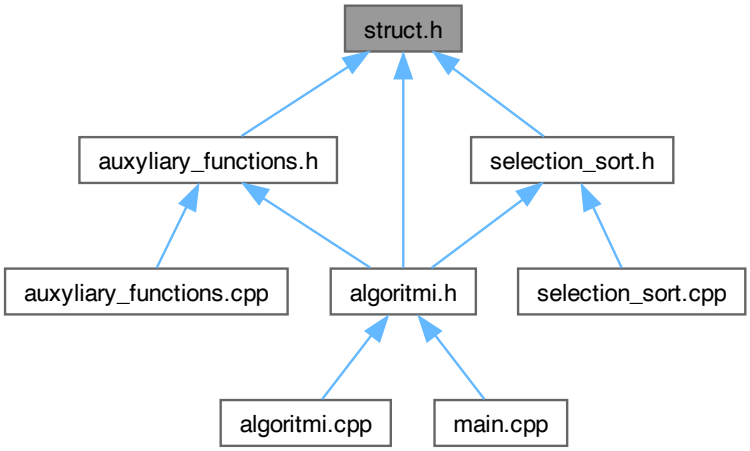
6.17 Riferimenti per il file struct.h

`#include <string>`

Grafo delle dipendenze di inclusione per struct.h:



Questo grafo mostra quali altri file includono direttamente o indirettamente questo file:



Composti

- struct Processo

  Struct creata per salvare il nome, la durata e la priorità del processo.

6.18 struct.h

Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 07/04/23.
00003 //
00004
00005 #ifndef PROGETTO_SISTEMI_OPERATIVI_STRUCT_H
00006 #define PROGETTO_SISTEMI_OPERATIVI_STRUCT_H
00007
00008 #include <string>
00009
00010
00011 using namespace std;
00012
00014 struct Processo{
00015     string nome;
00016     int istante_arrivo;
00017     int durata;
00018     int priorita;
00019 };
00020
00021 #endif //PROGETTO_SISTEMI_OPERATIVI_STRUCT_H
```

Indice analitico