Progetto Scheduling CPU

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

Operating Systems Project - Matteo Franchini

This project was developed as an optional assignment for the Operating Systems course. It involves the implementation of six short-term scheduling algorithms.

2.1 Implemented Algorithms

The following algorithms have been implemented:

- **FCFS** (First-Come, First-Served)
- **SJF** (Shortest Job First)
- **Priority Scheduling**
- **Round Robin (RR)**

In addition to the above algorithms, two additional algorithms were developed:

- **SRTF** (Shortest Remaining Time First)
- **Priority Scheduling with Round Robin**

2.2 Configuration Files

For all algorithms, except for "Priority Scheduling with Round Robin," a configuration file named "config.txt" is used for input.

However, for the "Priority Scheduling with Round Robin" algorithm, a separate configuration file named "config_↩ RR_priority.txt" was created, as there were no processes with equal priorities in the original "config.txt" file.

Please note that the project and its documentation are primarily written in Italian. This translation is provided for convenience, and it is recommended to refer to the original code and comments for precise details and instructions.

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

5.2 Riferimenti per la struct Processo_log

```
#include <struct.h>
```

Attributi pubblici

- string nome
- int time

5.2.1 Descrizione dettagliata

Struct creata per salvare il nome e il tempo di esecuzione del processo durante l'esecuzione dell'algoritmo SRTF. Verrà usata nel calcolo del tempo medio di esecuzione

Definizione alla linea 27 del file struct.h.

5.2.2 Documentazione dei membri dato

5.2.2.1 nome

```
string Processo_log::nome
```

Definizione alla linea 28 del file struct.h.

5.2.2.2 time

```
int Processo_log::time
```

Definizione alla linea 29 del file struct.h.

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

- struct.h

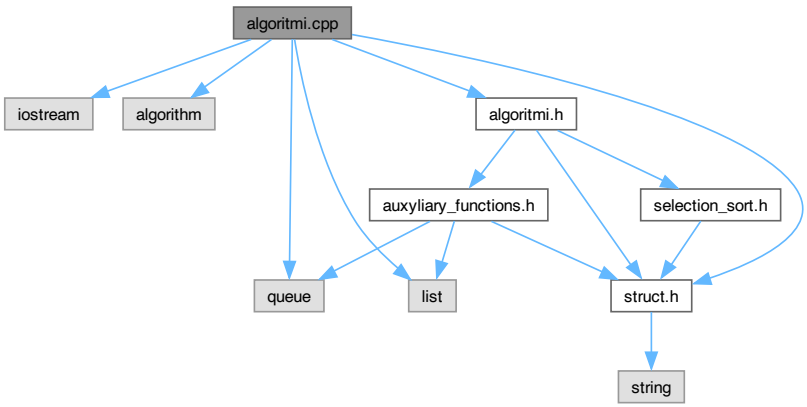
Capitolo 6

Documentazione dei file

6.1 Riferimenti per il file algoritmi.cpp

```
#include <iostream>
#include <algorithm>
#include <queue>
#include <list>
#include "algoritmi.h"
#include "struct.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)
- void algoritmo_priorita_RR (Processo *p, int num_processi, int quanto)

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 )
```

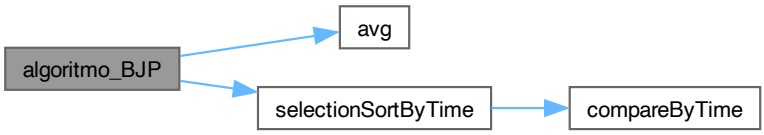
Definizione dell'algoritmo BJF

Parametri

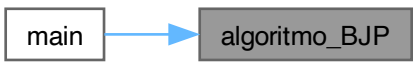
Processo	p
int	num_processi

Definizione alla linea 60 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

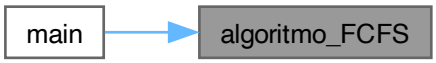
Parametri

Processo	p
int	num_processi

Definizione alla linea 23 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 )
```

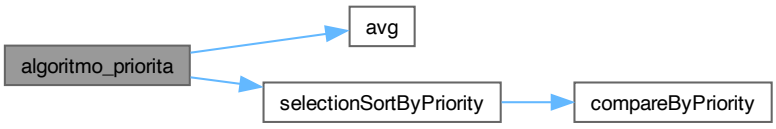
Definizione dell'algoritmo "Priorità"

Parametri

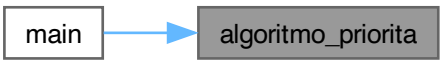
Processo	p
int	num_processi

Definizione alla linea 41 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_priorita_RR()

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

Algoritmo Round Robin con priorità

Parametri

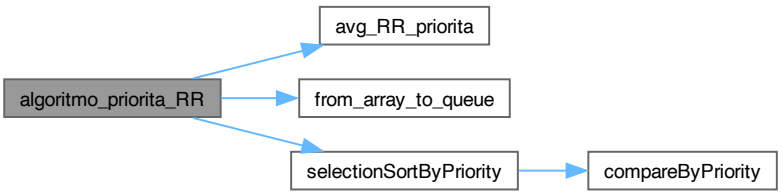
p	
num_processi	
quanto	

Ordinamento dei processi per priorità

Creazione della coda di processi a partire da un array ordinato per priorità

Definizione alla linea 198 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_RR()

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

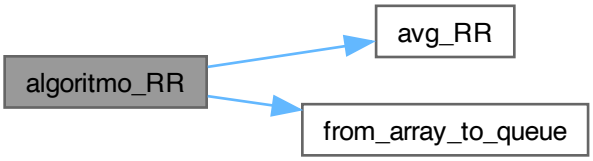
Definizione dell'algoritmo RR

Parametri

Processo	p
int	num_processi
int	quanto

Definizione alla linea 81 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.1.1.6 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

Inserimenti del processo all'intenro del log

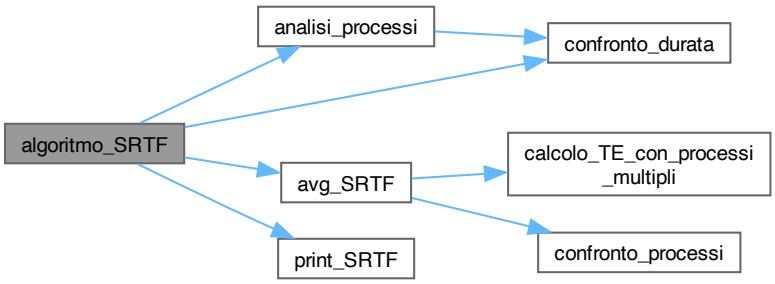
Quando la durata di un processo arriva a 0 lo andiamo a togliere dalla lista

L'algoritmo termina quando la lista è vuota

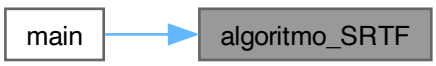
Stampa dei risultati dell'algoritmo

Definizione alla linea 121 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 15 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 #include "struct.h"
00011
00012
00013 using namespace std;
00014
00015 const int CONST = 100;
00016
00017
00023 void algoritmo_FCFS (Processo *p, int num_processi) {
00024     int n = CONST;
00025     int *array_durata = new int[n];
00026     cout « "FCFS ";
00027     for (int i = 0; i < num_processi; i++) {
00028         cout « "->" « p[i].nome;
00029         array_durata[i] = p[i].durata;
00030     }
00031     cout « endl « "TEMPO MEDIO " « avg(array_durata, num_processi) « endl;
00032     delete [] array_durata;
00033 }
00034
00035
00041 void algoritmo_priorita (Processo *p, int num_processi) {
00042     int n = CONST;
00043     int *array_durata = new int [n];
00044     selectionSortByPriority(p, num_processi);
00045     cout « "PRIORITÀ ";
00046     for (int i = 0; i < num_processi; i++) {
00047         cout « "->" « p[i].nome;
00048         array_durata[i] = p[i].durata;
00049     }
00050     cout « endl « "TEMPO MEDIO " « avg(array_durata, num_processi) « endl;
00051     delete [] array_durata;
00052 }
00053
00054
00060 void algoritmo_BJP (Processo *p, int num_processi) {
00061     int n = CONST;
00062     int *array_durata = new int [n];
00063     selectionSortByTime(p, num_processi);
00064     cout « "BJP ";
00065     for (int i = 0; i < num_processi; i++) {
00066         cout « "->" « p[i].nome;
00067         array_durata[i] = p[i].durata;
00068     }
00069     cout « endl « "TEMPO MEDIO " « avg(array_durata, num_processi) « endl;
00070     delete [] array_durata;
00071 }
00072
00073
00081 void algoritmo_RR (Processo *p, int num_processi, int quanto) {
00082     int array_counter = 0;
00083     int n = CONST;
00084     cout « "RR ";
00085     Processo *array_durata = new Processo [n];
00086     queue<Processo> processi = from_array_to_queue(p, num_processi);
00087     while (not processi.empty()) {
00088         if (processi.front().durata <= quanto) {
00089             cout « "->" « processi.front().nome;
00090             array_durata[array_counter] = processi.front();
00091             processi.pop();
00092             array_counter++;
00093         }
00094         else if (processi.front().durata >= quanto){
00095             Processo temp;
00096             temp.nome = processi.front().nome;
00097             temp.durata = processi.front().durata - quanto;
00098             temp.priorita = processi.front().priorita;
00099             cout « "->" « temp.nome;
00100             Processo durata;
00101             durata.nome = processi.front().nome;
00102             durata.durata = quanto;
00103             durata.priorita = processi.front().priorita;
00104             array_durata[array_counter] = durata;
```

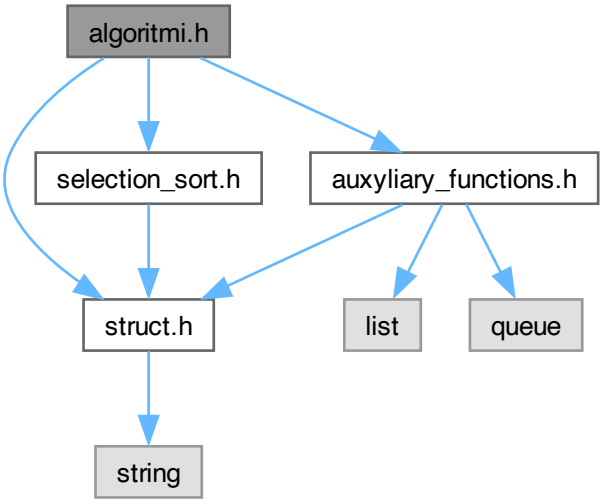
```
00105             processi.push(temp);
00106             processi.pop();
00107             array_counter++;
00108         }
00109     }
00110     cout « endl « "TEMPO MEDIO " « avg_RR(array_durata, array_counter, num_processi) « endl;
00111     delete [] array_durata;
00112 }
00113
00114
00121 void algoritmo_SRTF (Processo *p, int num_processi) {
00122     int counter = 0; int time = -1; Processo temp;
00123     list<Processo_log> log;
00124     list<Processo> lista;
00125     list<Processo> processi_analizzati;
00126     bool flag = false;
00127     cout « "SRTF ";
00128     while (flag == false) {
00129         time++;
00130
00138         processi_analizzati = analisi_processi(p, num_processi, time);
00139
00145         while (not processi_analizzati.empty()) {
00146             lista.push_front(processi_analizzati.front());
00147             processi_analizzati.pop_front();
00148         }
00149
00156         lista.sort(confronto_durata);
00157
00159
00160         lista.front().durata--;
00161
00162
00164
00165         Processo_log temp;
00166         temp.nome = lista.front().nome;
00167         temp.time = time;
00168         log.push_front(temp);
00169
00175         if (lista.front().durata == 0) {
00176             lista.pop_front();
00177         }
00178
00180
00181         if (lista.empty()) { flag = true; }
00182     }
00183
00185
00186     print_SRTF(log);
00187
00188     cout « "\nTEMPO MEDIO: " « avg_SRTF(log, num_processi) « endl;
00189 }
00190
00198 void algoritmo_priorita_RR (Processo *p, int num_processi, int quanto) {
00199     Processo_log* arr_log = new Processo_log[CONST];
00200     int counter = 0;
00201     int i = 0;
00202
00204     selectionSortByPriority(p, num_processi);
00205
00207     queue<Processo> coda_processi;
00208     coda_processi = from_array_to_queue(p, num_processi);
00209
00210     while(not coda_processi.empty()) {
00211         if (coda_processi.front().durata <= quanto) {
00212             Processo_log temp_log;
00213             arr_log[i].nome = coda_processi.front().nome;;
00214             arr_log[i].time = counter;
00215
00216             cout « "->" « coda_processi.front().nome;
00217             counter = counter + coda_processi.front().durata;
00218
00219             coda_processi.pop();
00220         }
00221         else if (coda_processi.front().durata > quanto) {
00222             Processo_log temp_log;
00223             temp_log.nome = coda_processi.front().nome;
00224             temp_log.time = counter;
00225             arr_log[i] = temp_log;
00226
00227             Processo temp;
00228             temp.nome = coda_processi.front().nome;
00229             temp.durata = coda_processi.front().durata - quanto;
00230             temp.priorita = coda_processi.front().priorita;
00231             temp.istante_arrivo = coda_processi.front().istante_arrivo;
00232             cout « "->" « temp.nome;
00233
```

```
00234             counter = counter + quanto;
00235
00236             coda_processi.pop();
00237             coda_processi.push(temp);
00238         }
00239         counter++;
00240         i++;
00241     }
00242
00243     cout « "\nTEMPO MEDIO: " « avg_RR_priorita(arr_log, num_processi, i) « endl;
00244     delete [] arr_log;
00245 }
00246
```

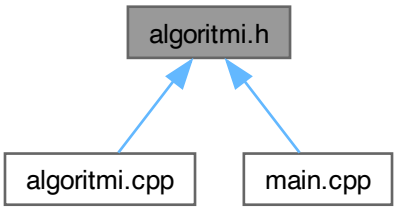
6.3 Riferimenti per il file algoritmi.h

```
#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)
- void algoritmo_priorita_RR (Processo *p, int num_processi, int quanto)

6.3.1 Documentazione delle funzioni

6.3.1.1 algoritmo_BJP()

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

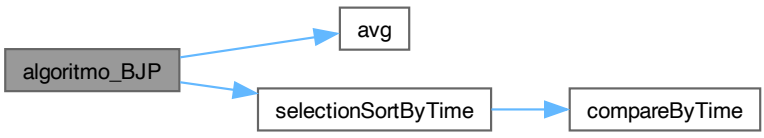
Definizione dell'algoritmo BJF

Parametri

Processo	p
int	num_processi

Definizione alla linea 60 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 23 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 )
```

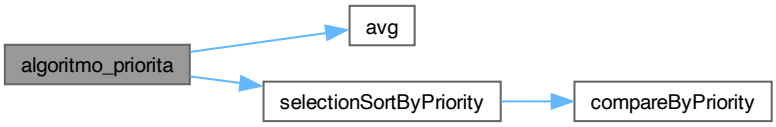
Definizione dell'algoritmo "Priorità"

Parametri

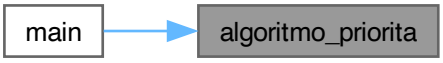
Processo	p
int	num_processi

Definizione alla linea 41 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_priorita_RR()

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

Algoritmo Round Robin con priorità

Parametri

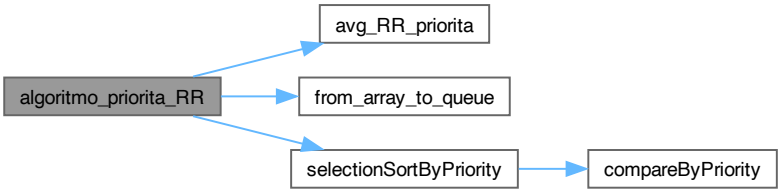
p	
num_processi	
quanto	

Ordinamento dei processi per priorità

Creazione della coda di processi a partire da un array ordinato per priorità

Definizione alla linea 198 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_RR()

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

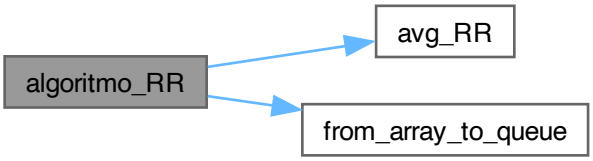
Definizione dell'algoritmo RR

Parametri

Processo	p
int	num_processi
int	quanto

Definizione alla linea 81 del file algoritmi.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:

6.3.1.6 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

Inserimenti del processo all'intenro del log

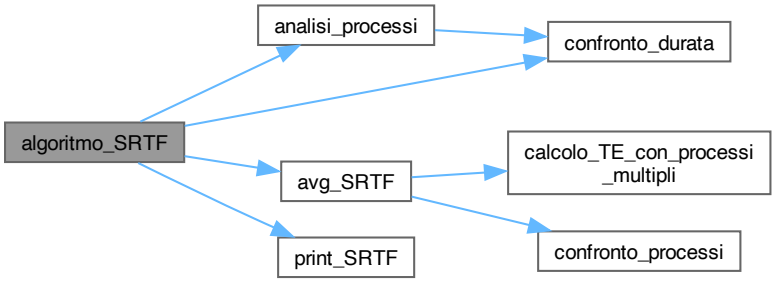
Quando la durata di un processo arriva a 0 lo andiamo a togliere dalla lista

L'algoritmo termina quando la lista è vuota

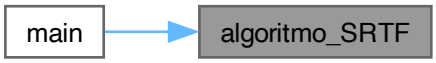
Stampa dei risultati dell'algoritmo

Definizione alla linea 121 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

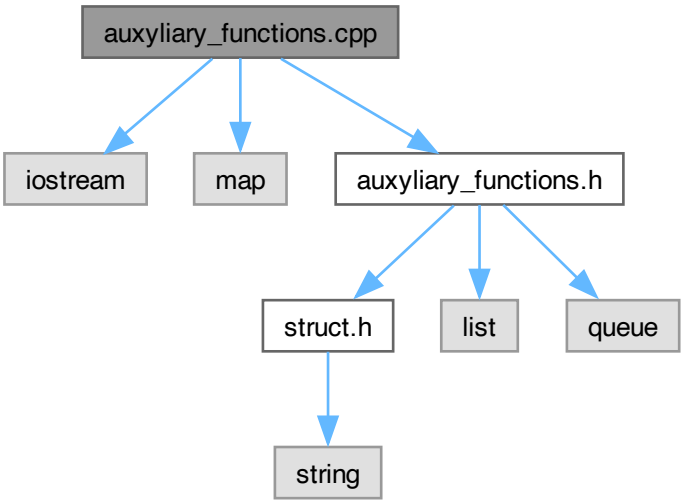
Vai alla documentazione di questo file.

```
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 void algoritmo_priorita_RR (Processo *p, int num_processi, int quanto);
00018
00019 #endif //PROGETTO_SISTEMI_OPERATIVI_ALGORITMI_H
```

6.5 Riferimenti per il file auxyliary_functions.cpp

```
#include <iostream>
#include <map>
#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)
- void reset_array (Processo *arr, Processo *arr_copia, int num_processi)
- void print_SRTF (list< Processo_log > &log)
- bool confronto_processi (const Processo_log &a, const Processo_log &b)
- int calcolo_TE_con_processi_multipli (Processo_log *arr, int count)
- float avg_SRTF (list< Processo_log > &log, int num_processi)
- float avg_RR_priorita (Processo_log *arr, int num_processi, int size)

Variabili

- const int CONST_NUM = 100000

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

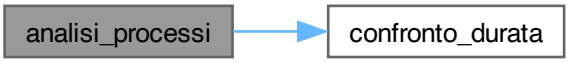
Processo	p
int	num_processi
int	time

Restituisce

list<Processo> processi_in_arrivo

Definizione alla linea 20 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

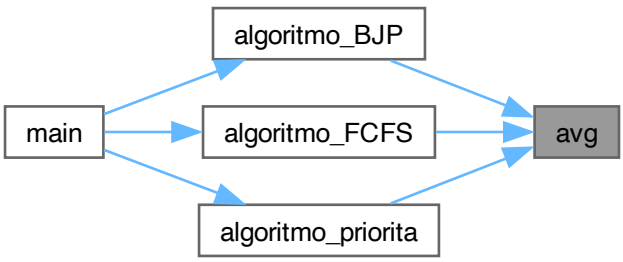
Restituisce

float avg

Conversione da int a float per garantire che la funzione restituisca anche numeri decimali

Definizione alla linea 72 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

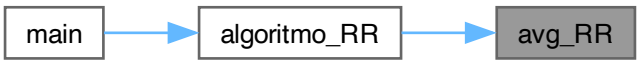
Restituisce

float avg

Conversione da int a float per garantire che la funzione restituisca anche numeri decimali

Definizione alla linea 106 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.4 avg_RR_priorita()

```
float avg_RR_priorita (
            Processo_log * arr,
            int num_processi,
            int size )
```

Calcolo del tempo medio per l'algoritmo RR con priorità

Parametri

arr	
num_processi	
size	

Restituisce

float avg

Creazione di un dizionario per salvare i processi che sono stati eseguiti per ultiimi

Creazione di un dizionario per salvare i tempi di esecuzione dei processi eseguiti prima dell'ultimo, questo valore andrà sottratto a quello massimo in modo da calcolare il tempo medio

Popolazione dei dizionari

Definizione alla linea 314 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.5.1.5 avg_SRTF()

```
float avg_SRTF (
            list< Processo_log > & log,
            int num_processi )
```

Questa funzione permette il calcolo del tempo di attesa per l'algoritmo SRTF

Parametri

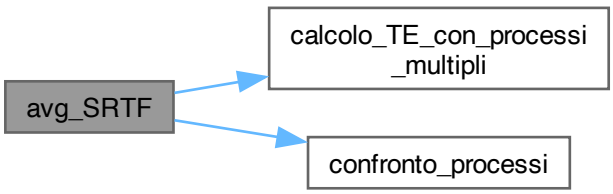
log	
num_processi	

Restituisce

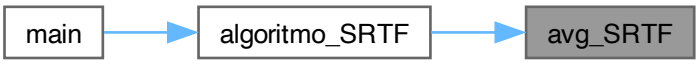
float avg

Definizione alla linea 222 del file auxyliary_functions.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



6.5.1.6 calcolo_TE_con_processi_multipli()

```
int calcolo_TE_con_processi_multipli (
            Processo_log * arr,
            int count )
```

Questa funzione permette il calcolo del tempo di attesa per l'algoritmo SRTF in presenza di più di un processo ripetuto

Parametri

arr	
count	

Restituisce

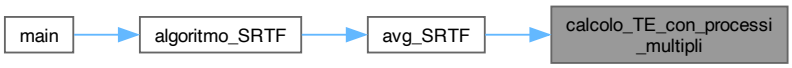
int sum

Come prima cosa andiamo a prendere il tempo del processo che sarà in posizione count-2 in quanto l'ultimo processo, in posizione count-1, non ci interessa perché stiamo calcolando il tempo di attesa e non quello di esecuzione, però non basta calcolare il tempo di esecuzione prendendo questo tempo, ma sarà necessario andare a sottrarre a questo tempo quello delle precedenti esecuzioni del processo

Una volta individuato qual è il processo che ha tempo di attesa più lungo per andare a togliere il tempo delle precedenti esecuzioni non facciamo altro che andare a sottrarre il tempo che abbiamo nelle posizioni dispari, questo perché ogni processo ha un valore iniziale e uno finale (il valore iniziale sarà in posizione dispari mentre quello finale in posizione pari), ma per quanto detto prima a noi interessa il valore iniziale e quindi prendiamo solo quelli in posizione dispari

Definizione alla linea 186 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.7 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

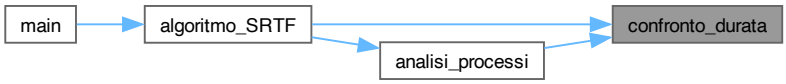
Processo	a
Processo	b

Restituisce

bool

Definizione alla linea 39 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.8 confronto_processi()

```
bool confronto_processi (
            const Processo_log & a,
            const Processo_log & b )
```

Funzione che permette di ordinare una lista di processi in base al nome

Parametri

log	
p	
num_processi	

Restituisce

bool

Definizione alla linea 174 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.5.1.9 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

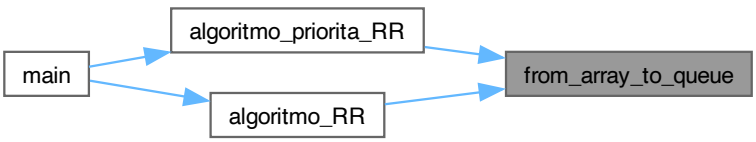
Processo	p
int	num_processi

Restituisce

queue<Processo> coda

Definizione alla linea 50 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.1.10 print_SRTF()

```
void print_SRTF (
            list< Processo_log > & log )
```

Funzione che permette di stampare i risultati dell'algoritmo SRTF

Parametri

log	
p	
num_processi	

Definizione alla linea 152 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.5.1.11 reset_array()

```
void reset_array (
            Processo * arr,
            Processo * arr_copia,
            int num_processi )
```

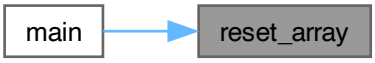
Funzione che permette di resettare l'array dei processi

Parametri

Processo	arr
Processo	arr_copia
int	num_processi

Definizione alla linea 139 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.5.2 Documentazione delle variabili

6.5.2.1 CONST_NUM

```
const int CONST_NUM = 100000
```

Definizione alla linea 8 del file auxyliary_functions.cpp.

6.6 auxyliary_functions.cpp

Vai alla documentazione di questo file.

```
00001 //
00002 // Created by Matteo Franchini on 25/04/23.
00003 //
00004 #include <iostream>
00005 #include <map>
00006 #include "auxyliary_functions.h"
00007
00008 const int CONST_NUM = 100000;
00009
00020 list<Processo> analisi_processi (Processo *p, int num_processi, int time) {
00021     list<Processo> processi_in_arrivo;
00022     for (int i = 0; i < num_processi; i++) {
00023         if (p[i].istante_arrivo == time) {
00024             processi_in_arrivo.push_back(p[i]);
00025             processi_in_arrivo.sort(confronto_durata);
00026         }
00027     }
00028     return processi_in_arrivo;
00029 }
00030
00039 bool confronto_durata (const Processo& a, const Processo& b) {
00040     return a.durata < b.durata;
00041 }
00042
00050 queue<Processo> from_array_to_queue(Processo *p, int num_processi) {
00051     queue<Processo> var;
00052     for (int i = 0; i < num_processi; i++) {
00053         if (p[i].nome == "") {
00054             continue;
00055         }
00056         Processo temp;
00057         temp.nome = p[i].nome;
00058         temp.durata = p[i].durata;
00059         temp.priorita = p[i].priorita;
00060         var.push(temp);
00061     }
00062     return var;
00063 }
00064
00065
00072 float avg (int *durata, int size) {
00073     // Tempo di attesa del singolo processo
00074
00075     int sum_int = 0;
00076
00077     // Tempo di attesa complessivo
00078
00079     int sum = 0;
00080
00081     for (int j = 0; j < size-1; j++) {
00082         sum_int += durata[j];
00083         sum += sum_int;
00084     }
00085
00091     float sum_dec = static_cast<float>(sum);
00092     float size_dec = static_cast<float>(size);
00093
00094     return sum_dec/size_dec;
00095 }
00096
00097
00106 float avg_RR (Processo *durata, int size, int num_processi) {
00107     bool flag = false;
00108     int sum = 0;
00109     for (int i = 0; i < num_processi; i++) {
00110         for (int j = size; j >= 0; j--) {
00111             if (durata[i].nome == durata[j].nome) {
00112                 flag = true;
00113             }
```

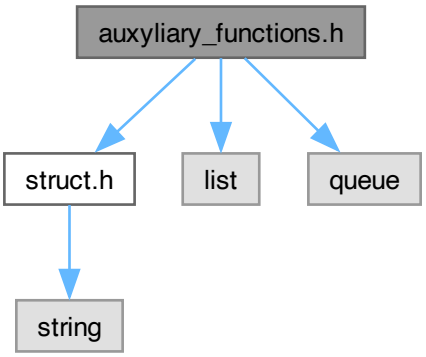
```
00114             if (flag && durata[i].nome != durata[j].nome) {
00115                 sum += durata[j].durata;
00116             }
00117         }
00118         flag = false;
00119     }
00120
00126     float sum_dec = static_cast<float>(sum);
00127     float processi_dec = static_cast<float>(num_processi);
00128
00129     return sum_dec/processi_dec;
00130 }
00131
00139 void reset_array (Processo *arr, Processo *arr_copia, int num_processi) {
00140     for (int i = 0; i < num_processi; i++) {
00141         arr_copia[i] = arr[i];
00142     }
00143 }
00144
00152 void print_SRTF (list<Processo_log> &log) {
00153     list<Processo_log>::iterator it = log.end();
00154     list<Processo_log>::iterator temp = prev(it);
00155     list<Processo_log> cambio_processo;
00156
00157     while (it != log.begin()) {
00158         if (temp->nome != it->nome) {
00159             cout « "->" « temp->nome;
00160         }
00161         --it;
00162         --temp;
00163     }
00164 }
00165
00174 bool confronto_processi (const Processo_log& a, const Processo_log& b) {
00175     return a.nome < b.nome;
00176 }
00177
00186 int calcolo_TE_con_processi_multipli (Processo_log *arr, int count) {
00187     int sum = 0;
00188
00197     sum = arr[count -2].time;
00198
00207     for (int i = count - 3; i >= 0; i--) {
00208         if (count % 2 != 0) {
00209             sum = sum - arr[i].time;
00210         }
00211     }
00212     return sum;
00213 };
00214
00222 float avg_SRTF (list<Processo_log> &log, int num_processi) {
00223     list<Processo_log>::iterator it = log.end();
00224     list<Processo_log>::iterator temp = prev(it);
00225     list<Processo_log> cambio_processo;
00226
00227     while (it != log.begin()) {
00228         if (temp->nome != it->nome) {
00229             if (it->nome != "") {
00230                 Processo_log temp_proc_prec;
00231                 temp_proc_prec.nome = it->nome;
00232                 temp_proc_prec.time = it->time;
00233                 cambio_processo.push_back(temp_proc_prec);
00234             }
00235
00236             Processo_log temp_proc_succ;
00237             temp_proc_succ.nome = temp->nome;
00238             temp_proc_succ.time = temp->time;
00239             cambio_processo.push_back(temp_proc_succ);
00240
00241         }
00242         --it;
00243         --temp;
00244
00245         if (it == log.begin()) {
00246             Processo_log temp_proc_prec;
00247             temp_proc_prec.nome = it->nome;
00248             temp_proc_prec.time = it->time;
00249             cambio_processo.push_back(temp_proc_prec);
00250         }
00251     }
00252
00253     cambio_processo.sort(confronto_processi);
00254
00255     it = cambio_processo.begin();
00256     map<string, int> counter;
00257
00258     while (it != cambio_processo.end()) {
```

```
00259         temp = cambio_processo.begin();
00260         while (temp != cambio_processo.end()) {
00261             if (it->nome == temp->nome && it->time == temp->time) {
00262                 counter[it->nome]++;
00263             }
00264             ++temp;
00265         }
00266         ++it;
00267     }
00268 
00269     it = cambio_processo.begin();
00270     temp = next(it);
00271 
00272     Processo_log *arr_ausiliario = new Processo_log[CONST_NUM];
00273     float sum = 0;
00274 
00275     int count = 1;
00276     bool flag = false;
00277 
00278     while (it != cambio_processo.end()) {
00279         if (it->nome == temp->nome) {
00280             if (counter[it->nome] == 2) {
00281                 if (it->time < temp->time) { sum += it->time; }
00282                 else if (it->time > temp->time) { sum += temp->time; }
00283             }
00284 
00285             else if (counter[it->nome] > 2) {
00286                 arr_ausiliario[count] = *it;
00287                 count++;
00288                 if (count+1 == counter[it->nome]) { flag = true; }
00289             }
00290 
00291             if (flag) {
00292                 sum += calcolo_TE_con_processi_multipli(arr_ausiliario, counter[it->nome]);
00293                 flag = false;
00294             }
00295         }
00296         ++it;
00297         ++temp;
00298     }
00299     delete[] arr_ausiliario;
00300 
00301     float size_dec = static_cast<float>(num_processi);
00302 
00303     return sum/size_dec;
00304 }
00305 
00314 float avg_RR_priorita (Processo_log *arr, int num_processi, int size) {
00315     float sum = 0;
00316 
00318 
00319     map<string, int> map_max;
00320 
00327     map<string, int> map_sum;
00328 
00330 
00331     for (int i = 0; i < size; i++) {
00332         for (int j = 0; j < size; j++) {
00333             if (arr[i].nome == arr[j].nome && arr[i].time <= arr[j].time) {
00334                 map_max[arr[j].nome] = arr[j].time;
00335             }
00336         }
00337     }
00338 
00339     for (int i = 0; i < size; i++) {
00340         if (arr[i].time < map_max[arr[i].nome]) {
00341             map_sum[arr[i].nome] += arr[i+1].time - arr[i].time - 1;
00342         }
00343     }
00344 
00345     for (int i = 0; i < num_processi; i++) {
00346         sum += map_max[arr[i].nome] - map_sum[arr[i].nome];
00347     }
00348 
00349     float size_dec = static_cast<float>(num_processi);
00350     return sum/size_dec;
00351 }
```

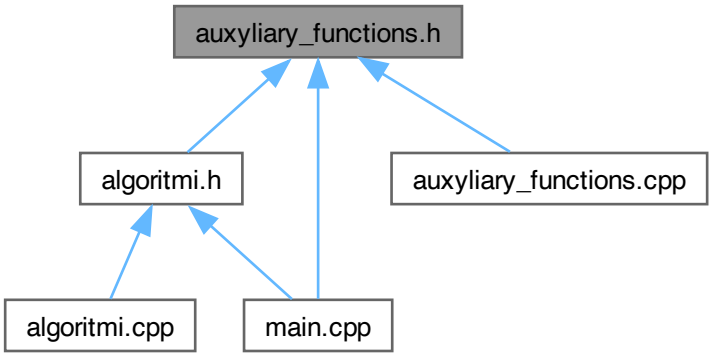
6.7 Riferimenti per il file auxyliary_functions.h

```
#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)
- void reset_array (Processo *arr, Processo *arr_copia, int num_processi)
- void print_SRTF (list< Processo_log > &log)
- int calcolo_TE_con_processi_multipli (Processo_log *arr, int count)
- bool confronto_processi (const Processo_log &a, const Processo_log &b)
- float avg_SRTF (list< Processo_log > &log, int num_processi)
- float avg_RR_priorita (Processo_log *arr, int num_processi, int size)

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

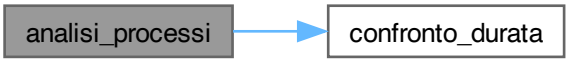
Processo	p
int	num_processi
int	time

Restituisce

list<Processo> processi_in_arrivo

Definizione alla linea 20 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

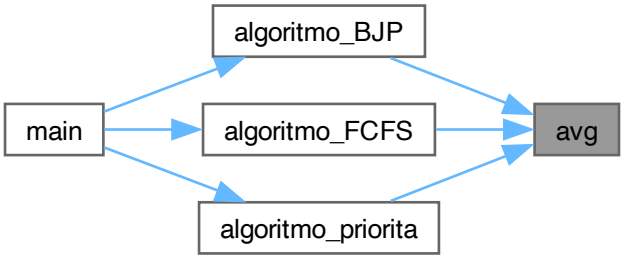
Restituisce

float avg

Conversione da int a float per garantire che la funzione restituisca anche numeri decimali

Definizione alla linea 72 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

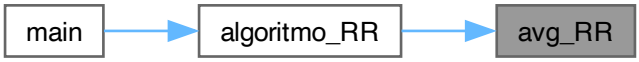
Restituisce

float avg

Conversione da int a float per garantire che la funzione restituisca anche numeri decimali

Definizione alla linea 106 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.4 avg_RR_priorita()

```
float avg_RR_priorita (
            Processo_log * arr,
            int num_processi,
            int size )
```

Calcolo del tempo medio per l'algoritmo RR con priorità

Parametri

arr	
num_processi	
size	

Restituisce

float avg

Creazione di un dizionario per salvare i processi che sono stati eseguiti per ultiimi

Creazione di un dizionario per salvare i tempi di esecuzione dei processi eseguiti prima dell'ultimo, questo valore andrà sottratto a quello massimo in modo da calcolare il tempo medio

Popolazione dei dizionari

Definizione alla linea 314 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.7.1.5 avg_SRTF()

```
float avg_SRTF (
            list< Processo_log > & log,
            int num_processi )
```

Questa funzione permette il calcolo del tempo di attesa per l'algoritmo SRTF

Parametri

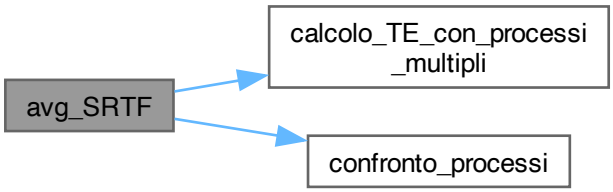
log	
num_processi	

Restituisce

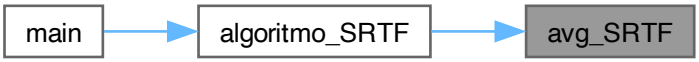
float avg

Definizione alla linea 222 del file auxyliary_functions.cpp.

Questo è il grafo delle chiamate per questa funzione:



Questo è il grafo dei chiamanti di questa funzione:



6.7.1.6 calcolo_TE_con_processi_multipli()

```
int calcolo_TE_con_processi_multipli (
            Processo_log * arr,
            int count )
```

Questa funzione permette il calcolo del tempo di attesa per l'algoritmo SRTF in presenza di più di un processo ripetuto

Parametri

arr	
count	

Restituisce

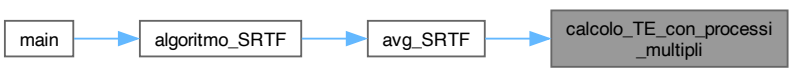
int sum

Come prima cosa andiamo a prendere il tempo del processo che sarà in posizione count-2 in quanto l'ultimo processo, in posizione count-1, non ci interessa perché stiamo calcolando il tempo di attesa e non quello di esecuzione, però non basta calcolare il tempo di esecuzione prendendo questo tempo, ma sarà necessario andare a sottrarre a questo tempo quello delle precedenti esecuzioni del processo

Una volta individuato qual è il processo che ha tempo di attesa più lungo per andare a togliere il tempo delle precedenti esecuzioni non facciamo altro che andare a sottrarre il tempo che abbiamo nelle posizioni dispari, questo perché ogni processo ha un valore iniziale e uno finale (il valore iniziale sarà in posizione dispari mentre quello finale in posizione pari), ma per quanto detto prima a noi interessa il valore iniziale e quindi prendiamo solo quelli in posizione dispari

Definizione alla linea 186 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.7 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

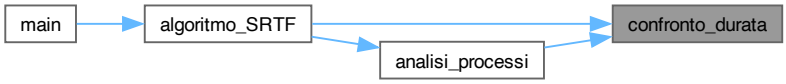
Processo	a
Processo	b

Restituisce

bool

Definizione alla linea 39 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.8 confronto_processi()

```
bool confronto_processi (
            const Processo_log & a,
            const Processo_log & b )
```

Funzione che permette di ordinare una lista di processi in base al nome

Parametri

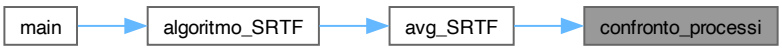
log	
p	
num_processi	

Restituisce

bool

Definizione alla linea 174 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.9 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

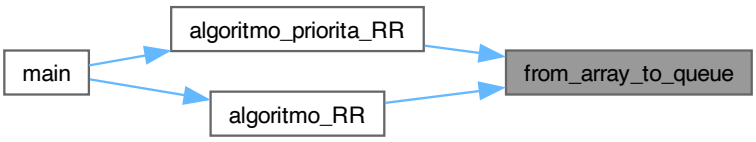
Processo	p
int	num_processi

Restituisce

queue<Processo> coda

Definizione alla linea 50 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:



6.7.1.10 print_SRTF()

```
void print_SRTF (
            list< Processo_log > & log )
```

Funzione che permette di stampare i risultati dell'algoritmo SRTF

Parametri

log	
p	
num_processi	

Definizione alla linea 152 del file auxyliary_functions.cpp.

Questo è il grafo dei chiamanti di questa funzione:

6.7.1.11 reset_array()

```
void reset_array (
            Processo * arr,
            Processo * arr_copia,
            int num_processi )
```

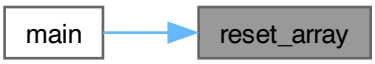
Funzione che permette di resettare l'array dei processi

Parametri

Processo	arr
Processo	arr_copia
int	num_processi

Definizione alla linea 139 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.

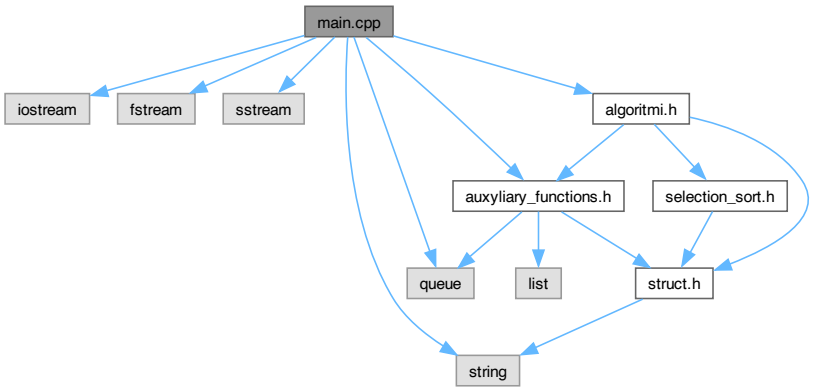
```
00001 //
00002 // Created by Matteo Franchini on 25/04/23.
00003 //
00004
00005 #ifndef PROGETTO_SISTEMI_OPERATIVI_AUXYLIARY_FUNCTIONS_H
00006 #define PROGETTO_SISTEMI_OPERATIVI_AUXYLIARY_FUNCTIONS_H
00007
00008 #include "struct.h"
00009 #include <list>
00010 #include <queue>
00011
00012 list<Processo> 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 void reset_array (Processo *arr, Processo *arr_copia, int num_processi);
00018 void print_SRTF (list<Processo_log> &log);
00019 int calcolo_TE_con_processi_multipli (Processo_log *arr, int count);
00020 bool confronto_processi (const Processo_log& a, const Processo_log& b);
00021 float avg_SRTF (list<Processo_log> &log, int num_processi);
00022 float avg_RR_priorita (Processo_log *arr, int num_processi, int size);
00023
00024 #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"
#include "auxyliary_functions.h"
```

Grafo delle dipendenze di inclusione per main.cpp:



Funzioni

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

Variabili

- const int CONST = 100
- const int NUM_ALGORITMI = 6

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

Creazione di un array di copia per far si che l'array originale venga mantenuto anche dopo l'esecuzione di un algoritmo

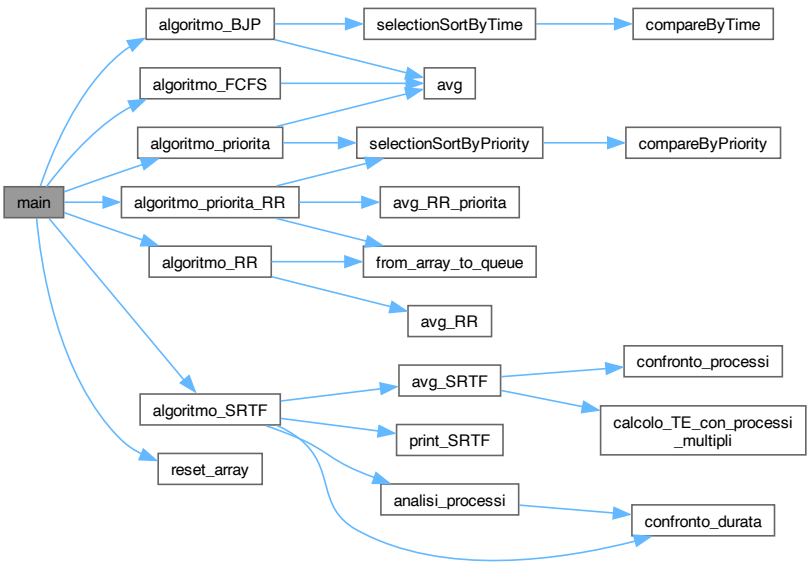
Esecuzione degli algoritmi

Reset dell'array di copia dopo ogni esecuzione di un algoritmo in modo che l'array di copia sia sempre uguale all'array originale

Cancellazione degli array creati con allocazione dinamica

Definizione alla linea 18 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 15 del file main.cpp.

6.10.2.2 NUM_ALGORITMI

```
const int NUM_ALGORITMI = 6
```

Definizione alla linea 16 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 #include "auxyliary_functions.h"
00012
00013 using namespace std;
00014
00015 const int CONST = 100;
00016 const int NUM_ALGORITMI = 6;
00017
00018 int main(int argc, char *argv[]) {
00019
00021     if (argc != 2) {
00022         cout « "Errore: inserire il nome del file di input" « endl;
00023         return 1;
00024     }
00025
00027     else {
00028
00030     string nome; int durata; int priorita; int istante_arrivo;
00031     string str1; string str2; string str3; string str4; string str5;
00032     int num_processi;
00033     int quanto;
00034     int counter = -1;
00035     int scelta_algoritmo = 0;
00036
00037
00039     int n = CONST;
00040     Processo* arr = new Processo[n];
00041
00043     ifstream myfile (argv[1]);
00044
00045     if (myfile.is_open()) {
00046         while (getline(myfile, str5, ' ')) {
00047             if (counter == -1) {
00048                 getline(myfile, str3, ' ');
00049                 istringstream tk3 (str3);
00050                 tk3 » num_processi;
00051                 getline(myfile, str4);
```

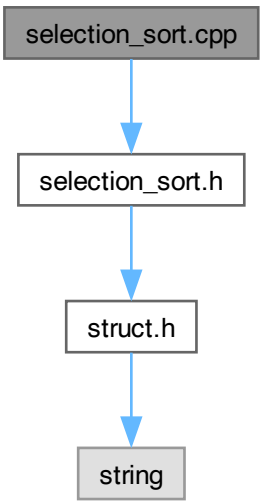
```
00052                 istringstream tk4 (str4);
00053                 tk4 » quanto;
00054                 counter++;
00055             }
00056             else {
00057                 istringstream tk5(str5);
00058                 tk5 » istante_arrivo;
00059                 getline(myfile, nome, ' ');
00060                 getline(myfile, str1, ' ');
00061                 istringstream tk1(str1);
00062                 tk1 » durata;
00063                 getline(myfile, str2);
00064                 istringstream tk2(str2);
00065                 tk2 » priorita;
00066                 arr[counter].nome = nome;
00067                 arr[counter].istante_arrivo = istante_arrivo;
00068                 arr[counter].durata = durata;
00069                 arr[counter].priorita = priorita;
00070                 counter++;
00071             }
00072         }
00073         myfile.close();
00074     } else cout « "Impossibile aprire il file";
00075
00082     Processo arr_copia[num_processi];
00083
00085
00086     for (int j = 0; j < NUM_ALGORITMI; j++) {
00087
00094         reset_array(arr, arr_copia, num_processi);
00095         switch (j) {
00096             case 0:
00097                 cout « "Esecuzione algoritmo FCFS" « endl;
00098                 algoritmo_FCFS(arr_copia, num_processi);
00099                 break;
00100             case 1:
00101                 cout « "<------------------------------------------>" « endl;
00102                 cout « "Esecuzione algoritmo BJP" « endl;
00103                 algoritmo_BJP(arr_copia, num_processi);
00104                 break;
00105             case 2:
00106                 cout « "<------------------------------------------>" « endl;
00107                 cout « "Esecuzione algoritmo Priorità" « endl;
00108                 algoritmo_priorita(arr_copia, num_processi);
00109                 break;
00110             case 3:
00111                 cout « "<------------------------------------------>" « endl;
00112                 cout « "Esecuzione algoritmo RR" « endl;
00113                 algoritmo_RR(arr_copia, num_processi, quanto);
00114                 break;
00115             case 4:
00116                 cout « "<------------------------------------------>" « endl;
00117                 cout « "Esecuzione algoritmo SRTF" « endl;
00118                 algoritmo_SRTF(arr_copia, num_processi);
00119                 break;
00120             case 5:
00121                 cout « "<------------------------------------------>" « endl;
00122                 cout « "Esecuzione algoritmo priorità RR" « endl;
00123                 algoritmo_priorita_RR(arr_copia, num_processi, quanto);
00124                 break;
00125         }
00126     }
00127
00128     cout « "<------------------------------------------>" « endl;
00130
00131     delete[] arr;
00132
00133     return 0;
00134     }
00135 }
```

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:

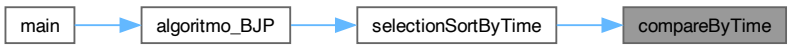
6.13.1.2 compareByTime()

```
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()

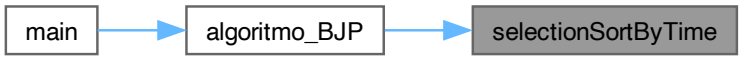
```
void selectionSortByTime (
            Processo * arr,
            int size )
```

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

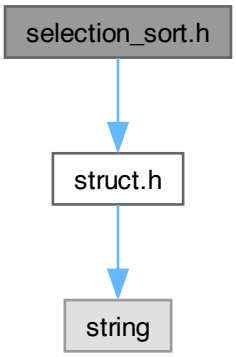
Vai alla documentazione di questo file.

```
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 }
```

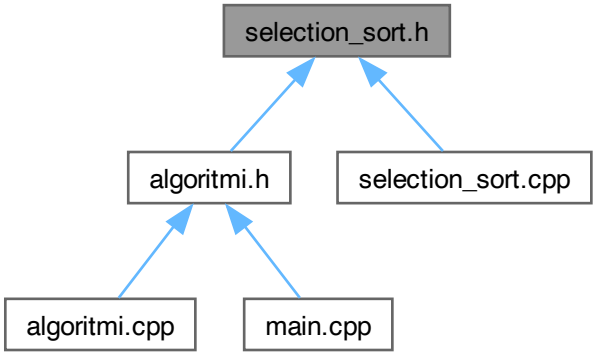
6.15 Riferimenti per il file selection_sort.h

```
#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()

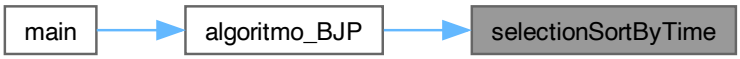
```
void selectionSortByTime (
            Processo * arr,
            int size )
```

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

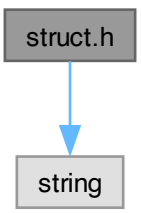
Vai alla documentazione di questo file.

```
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
```

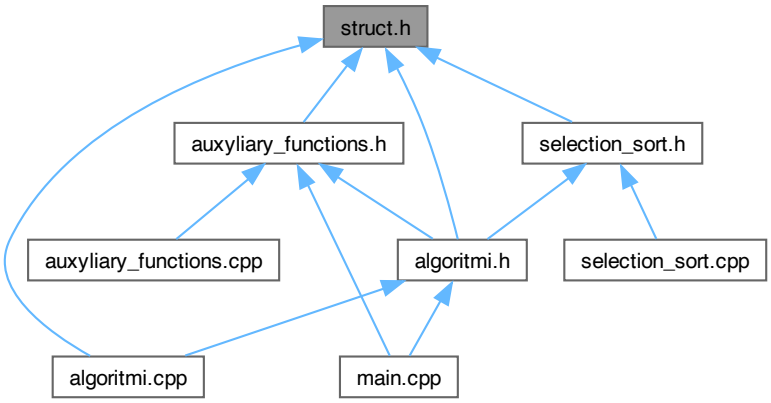
6.17 Riferimenti per il file struct.h

`#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.
- struct Processo_log

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
00027 struct Processo_log {
00028     string nome;
00029     int time;
00030 };
00031
00032 #endif //PROGETTO_SISTEMI_OPERATIVI_STRUCT_H
```

Indice analitico