Operating Systems - Project 2 - Banker's Algorithm

Generated by Doxygen 1.9.1

Chapter 1

Programming Assignment 2 - Banker's Algorithm

1.0.1 Author: Elliott Hager

1.0.2 Fall 2023 Semester

1.0.3 CS-33211: Operating Systems

1.0.4 Dr. Qiang Guan

1.0.5 Kent State University

1.0.5.1 Description (Provided from the assignment instructions)

Considering a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances, with the snapshot provided of the system at time t0.

Use the Banker's Algorithm to determine if the system is in a safe state. If the system is in a safe state, provide the path that is safe.

1.0.6 Implementation

Each process is a structure, which stores it's max and currently used resources. All of the processes are stored in an array. The Banker's algorithm will go through and check for a safe path. This is where all processes get the maximum resources allocated to them, while finding a sequence that does not cause deadlock. If no safe path is found, then it is reported back that with the current resource table given to the algorithm, it is not possible for safe allocation and continuation of the processes.

If there is a safe way to allocate all the resources to each process, the specific path is found, recorded, and returned as the safe route, given the current resource table given to the algorithm.

1.0.7 Resource Table Formatting

The resource table is formatted in the following way:

```
{Process number (0-based)}; {Allocated resource numbers, separated with a comma, until the last (i.e.
       1,2,3)}; {Max resource numbers, separated with a comma, until the last (i.e. 1,2,3)}; {available
       resources,separated with a comma, until the last (i.e. 1,2,3)};
```

1.0.8 Compilation

The algorithm implementation is separate from reading in the resource table from a provided text (.txt) file

1.0.8.1 Banker:

The following make command will compile the banker's algorithm file for execution

```
make banker
```

1.0.9 Util

The following make command will compile only the utility functions of the application

```
make util
```

1.0.10 Run:

The following make command will compile the banker's algorithm file and execute the program with the `./assets/resource_table.txt file as input

```
make run
```

1.0.11 Clean:

The following make command will remove the compiled and executable program files
```
make clean
```

1.0.12 Run Instructions:

To compile and run the Banker's algorithm, there is a couple ways of doing so.

1.0.12.1 1.) Manually/Command

```
./out/banker ./assets/resource_table.txt
```

1.0.13 2.) Using Make (Preferred)

Using the `make run` command will compile and run the algorithm This is the preferred method of compilation and running the project.

1.0.14 Documentation:

Please see the `Doc` folder for a PDF manual/documentation packet.

1.0.15 Libraries & Tech Stack

- C++
- Make
- G++

Chapter 2

File Index

2.1 File List

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

Chapter 3

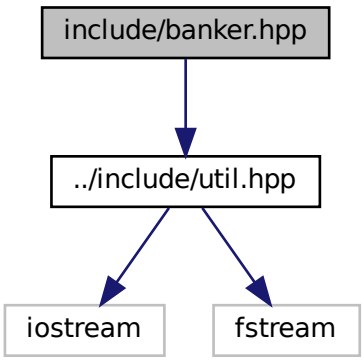
File Documentation

3.1 include/banker.hpp File Reference

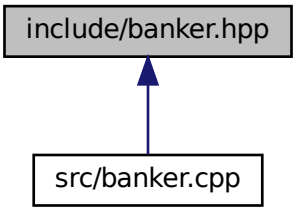
The header for the Banker's Algorithm implementation C++ file. Also sets global variables for the number of resources given and stated in the problem.

```
#include "../include/util.hpp"
```

Include dependency graph for banker.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- int main (int argc, char ∗argv[ ])

  The main function for the Banker's algorithm to start execution.

3.1.1 Detailed Description

The header for the Banker's Algorithm implementation C++ file. Also sets global variables for the number of resources given and stated in the problem.

Author

Elliott Hager

Date

2023-11-28

3.1.2 Function Documentation

3.1.2.1 main()

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

The main function for the Banker's algorithm to start execution.

Parameters

argc	(int) - The number of command line arguments fed into the program
argv	(char[]) - The command line arguments fed into the program

Returns

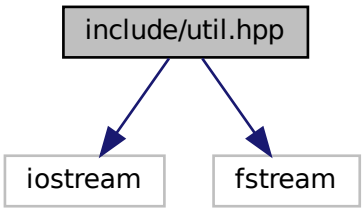
int - The status code of the code (0: successful execution, other - error)

3.2 include/util.hpp File Reference

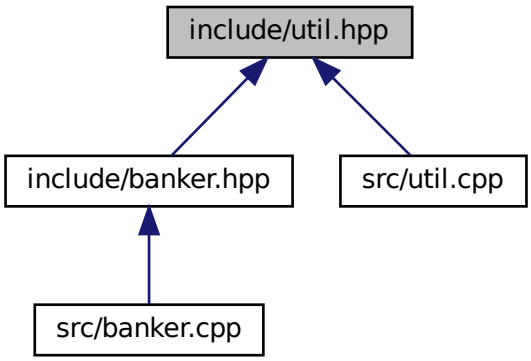
Header file to define functions and global constants necessary for File I/O and debug output.

```
#include <iostream>
#include <fstream>
```

Include dependency graph for util.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define RESOURCE_CATEGORIES 3

  The number of different system resources.
- #define PROCESSES 5

The number of processes for the system.

- #define LINE_DELIMITER ';'

  The delimeter to indicate the end of the list of one numerical type in the resource table (i.e. allocated, maximum, etc.)

- #define LIST_DELIMITER ','

  The delimiter to indicate the end of one numerical value for the resource table (i.e. 1,2,3 => 1 of Resource A, 2 of Resource B, etc.)

Functions

- void printTables (int allocated[ ][RESOURCE_CATEGORIES], int max[ ][RESOURCE_CATEGORIES], int *available)

  Print out each array and its contents, with multiple dimensions indicated (where applicable) for debugging purposes.

- void processResourceTable (char *file_name, int allocated[ ][RESOURCE_CATEGORIES], int max[ ][RESOURCE_CATEGORIE int *available)

  Processes the provided file for a resource table and fills allocated, maximum, and available resource arrays for algorithm processing and operations.

3.2.1 Detailed Description

Header file to define functions and global constants necessary for File I/O and debug output.

Author

Elliott Hager

Date

2023-12-01

3.2.2 Function Documentation

3.2.2.1 printTables()

```
void printTables (
            int allocated[][RESOURCE_CATEGORIES],
            int max[][RESOURCE_CATEGORIES],
            int * available )
```

Print out each array and its contents, with multiple dimensions indicated (where applicable) for debugging purposes.

Parameters

allocated	The 2-D array that holds the allocated resource counts for each process
max	The 2-D array that holds the maximum resource counts needed for each process
available	The 1-D array that holds the available resource counts the system has currently available

3.2.2.2 processResourceTable()

```
void processResourceTable (
            char * file_name,
            int allocated[][RESOURCE_CATEGORIES],
            int max[][RESOURCE_CATEGORIES],
            int * available )
```

Processes the provided file for a resource table and fills allocated, maximum, and available resource arrays for algorithm processing and operations.

Parameters

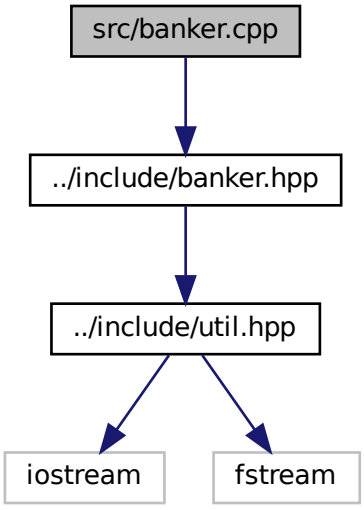
file_name	The file name of the resource table
allocated	The 2-D array that holds the allocated resource counts for each process
max	The 2-D array that holds the maximum resource counts needed for each process
available	The 1-D array that holds the available resource counts the system has currently available

3.3 src/banker.cpp File Reference

The implementation of the Deadlock avoidant, system resource allocation Banker's Algorithm.

```
#include "../include/banker.hpp"
```

Include dependency graph for banker.cpp:



Functions

- int main (int argc, char ∗argv[ ])

  The main function for the Banker's algorithm to start execution.

3.3.1 Detailed Description

The implementation of the Deadlock avoidant, system resource allocation Banker's Algorithm.

Author

Elliott Hager

Date

2023-12-01

3.3.2 Function Documentation

3.3.2.1 main()

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

The main function for the Banker's algorithm to start execution.

Parameters

argc	(int) - The number of command line arguments fed into the program
argv	(char[]) - The command line arguments fed into the program

Returns

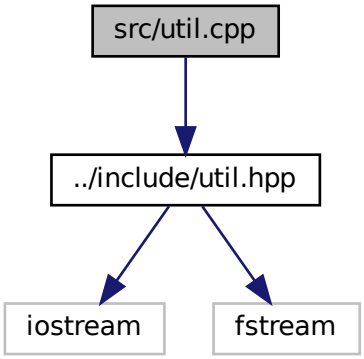
int - The status code of the code (0: successful execution, other - error)

3.4 src/util.cpp File Reference

The implementation of the File I/O and debugging functionality to read in the resource table for the Banker's Algorithm to analyze.

```
#include "../include/util.hpp"
```

Include dependency graph for util.cpp:



Functions

- void printTables (int allocated[ ][RESOURCE_CATEGORIES], int max[ ][RESOURCE_CATEGORIES], int ∗available)

  Print out each array and its contents, with multiple dimensions indicated (where applicable) for debugging purposes.
- void processResourceTable (char ∗file_name, int allocated[ ][RESOURCE_CATEGORIES], int max[ ][RESOURCE_CATEGORIE int ∗available)

  Processes the provided file for a resource table and fills allocated, maximum, and available resource arrays for algorithm processing and operations.

3.4.1 Detailed Description

The implementation of the File I/O and debugging functionality to read in the resource table for the Banker's Algorithm to analyze.

Author

Elliott Hager

Date

2023-12-01

3.4.2 Function Documentation

3.4.2.1 printTables()

```
void printTables (
            int allocated[][RESOURCE_CATEGORIES],
            int max[][RESOURCE_CATEGORIES],
            int * available )
```

Print out each array and its contents, with multiple dimensions indicated (where applicable) for debugging purposes.

Parameters

allocated	The 2-D array that holds the allocated resource counts for each process
max	The 2-D array that holds the maximum resource counts needed for each process
available	The 1-D array that holds the available resource counts the system has currently available

3.4.2.2 processResourceTable()

```
void processResourceTable (
            char * file_name,
            int allocated[][RESOURCE_CATEGORIES],
            int max[][RESOURCE_CATEGORIES],
            int * available )
```

Processes the provided file for a resource table and fills allocated, maximum, and available resource arrays for algorithm processing and operations.

Parameters

file_name	The file name of the resource table
allocated	The 2-D array that holds the allocated resource counts for each process
max	The 2-D array that holds the maximum resource counts needed for each process
available	The 1-D array that holds the available resource counts the system has currently available

Index