Deadlocks-Algorithm-Simulation

Generated by Doxygen 1.9.7

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

Chapter 6

Class Documentation

6.1 BankiersAlgorithm Class Reference

Class represents the Bankier Algorithm to prevent Deadlocks.

`#include <bankiersalgorithm.h>`

Inheritance diagram for BankiersAlgorithm:

Collaboration diagram for BankiersAlgorithm:

Public Member Functions

- BankiersAlgorithm ()

  BankiersAlgorithm standard constructor.
- ~BankiersAlgorithm ()
- QList< int > findNextResource (SystemProcess process) override

  findNextResource function that will find the next resource,
- bool avoidance_algorithm ()

  avoidance_algorithm algorithm which checks for deadlocks
- BankiersAlgorithm ()
- QList< int > findNextResource (SystemProcess process, int stillNeededResources_RCopy[3][4], int assignedResources_CCopy[3][4], int differenceResources_ACopy[4], int availableResources_ECopy[4])

  findNextResource function that will find the next resource
- bool avoidance_algorithm (int stillNeededResources_R[3][4], int assignedResources_C[3][4], int differenceResources_A[4], int availableResources_E[4])

  avoidance_algorithm avoids a deadlock with the algorithm specifice limitations

Public Member Functions inherited from deadlock_avoidance_api

- deadlock_avoidance_api ()
- virtual ~deadlock_avoidance_api ()=0
- virtual QList< int > findNextResource (SystemProcess process)

  findNextResource virtual function that will find the next resource,
- virtual void aquireConditionMet (int processId)
- virtual bool checkAquireCondition (int processId)

6.1.1 Detailed Description

Class represents the Bankier Algorithm to prevent Deadlocks.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 BankiersAlgorithm() [1/2]

```
BankiersAlgorithm::BankiersAlgorithm ( )
```

BankiersAlgorithm standard constructor.

6.1.2.2 ~BankiersAlgorithm()

```
BankiersAlgorithm::~BankiersAlgorithm ( )
```

6.1.2.3 BankiersAlgorithm() [2/2]

```
BankiersAlgorithm::BankiersAlgorithm ( )
```

6.1.3 Member Function Documentation

6.1.3.1 avoidance_algorithm() [1/2]

```
bool BankiersAlgorithm::avoidance_algorithm ( )
```

avoidance_algorithm algorithm which checks for deadlocks

Parameters

stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation
assignedResources_C	matrix containing which and how many resources a process is occupying
differenceResources_A	is an array with the current available resources
availableResources_E	is an array with the over all available resources

Returns

returns false if the current state is a deadlock or true if not

6.1.3.2 avoidance_algorithm() [2/2]

```
bool BankiersAlgorithm::avoidance_algorithm (
            int stillNeededResources_R[3][4],
```

```
            int assignedResources_C[3][4],
            int differenceResources_A[4],
            int availableResources_E[4] )
```

avoidance_algorithm avoids a deadlock with the algorithm specifice limitations

Parameters

stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation
assignedResources_C	matrix containing which and how many resources a process is occupying
differenceResources_A	is an array with the current available resources
availableResources_E	is an array with the over all available resources

Returns

6.1.3.3 findNextResource() [1/2]

```
QList< int > BankiersAlgorithm::findNextResource (
            SystemProcess process )  [override], [virtual]
```

findNextResource function that will find the next resource,

Parameters

process	is the process of the thread
availableResources_E	is an array with the over all available resources
differenceResources_A	is an array with the current available resources
assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation
result	from type @List<int>: first int is nextResource, second int is nextResourceCount, third int is indexResourceList

Reimplemented from deadlock_avoidance_api.

6.1.3.4 findNextResource() [2/2]

```
QList< int > BankiersAlgorithm::findNextResource (
            SystemProcess process,
            int stillNeededResources_RCopy[3][4],
            int assignedResources_CCopy[3][4],
            int differenceResources_ACopy[4],
            int availableResources_ECopy[4] )
```

findNextResource function that will find the next resource

Parameters

process	is the process of the thread
availableResources_E	is an array with the over all available resources
differenceResources_A	is an array with the current available resources
assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation
result	from type @List<int>: first int is nextResource, second int is nextResourceCount, third int is indexResourceList

Returns

The documentation for this class was generated from the following files:

- Algorithms/bankiersalgorithm.h
- bankiersalgorithm.h
- Algorithms/bankiersalgorithm.cpp

6.2 deadlock_avoidance_api Class Reference

The virtual class represents an API the algorithms use.

`#include <deadlock_avoidance_api.h>`

Inheritance diagram for deadlock_avoidance_api:

Public Member Functions

- deadlock_avoidance_api ()
- virtual ~deadlock_avoidance_api ()=0
- virtual QList< int > findNextResource (SystemProcess process)

  findNextResource virtual function that will find the next resource,
- virtual void aquireConditionMet (int processId)
- virtual bool checkAquireCondition (int processId)

6.2.1 Detailed Description

The virtual class represents an API the algorithms use.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 deadlock_avoidance_api()

`deadlock_avoidance_api::deadlock_avoidance_api ( )`

6.2.2.2 ~deadlock_avoidance_api()

```
deadlock_avoidance_api::~deadlock_avoidance_api ( )  [pure virtual]
```

6.2.3 Member Function Documentation

6.2.3.1 aquireConditionMet()

```
virtual void deadlock_avoidance_api::aquireConditionMet (
            int processId )  [inline], [virtual]
```

Reimplemented in NoPreemption.

6.2.3.2 checkAquireCondition()

```
virtual bool deadlock_avoidance_api::checkAquireCondition (
            int processId )  [inline], [virtual]
```

Reimplemented in NoPreemption.

6.2.3.3 findNextResource()

```
QList< int > deadlock_avoidance_api::findNextResource (
            SystemProcess process )  [virtual]
```

findNextResource virtual function that will find the next resource,

Parameters

process	is the process of the thread
availableResources_E	is an array with the over all available resources
differenceResources_A	is an array with the current available resources
assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation

Returns

result from type @List<int>: first int is nextResource, second int is nextResourceCount, third int is indexResourceList

Reimplemented in BankiersAlgorithm, EliminateCircularWait, EliminateHoldAndWait, and NoPreemption.

The documentation for this class was generated from the following files:

- Algorithms/deadlock_avoidance_api.h
- Algorithms/deadlock_avoidance_api.cpp

6.3 EliminateCircularWait Class Reference

Class represents the algorithm to eliminate CircularWait.

`#include <eliminatecircularwait.h>`

Inheritance diagram for EliminateCircularWait:

Collaboration diagram for EliminateCircularWait:

Public Member Functions

- EliminateCircularWait ()
  EliminateCircularWait standard constructor.
- ~EliminateCircularWait ()
- QList< int > findNextResource (SystemProcess process) override
  findNextResource function that will find the next resource,
- QList< SystemResource > avoidance_algorithm (QList< SystemResource > neededResources)
  avoidance_algorithm algorithm which sorts the neededResources to avoid a deadlock

Public Member Functions inherited from deadlock_avoidance_api

- deadlock_avoidance_api ()
- virtual ~deadlock_avoidance_api ()=0
- virtual QList< int > findNextResource (SystemProcess process)
  findNextResource virtual function that will find the next resource,
- virtual void aquireConditionMet (int processId)
- virtual bool checkAquireCondition (int processId)

6.3.1 Detailed Description

Class represents the algorithm to eliminate CircularWait.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 EliminateCircularWait()

```
EliminateCircularWait::EliminateCircularWait ( )
```

EliminateCircularWait standard constructor.

6.3.2.2 ~EliminateCircularWait()

```
EliminateCircularWait::~EliminateCircularWait ( )
```

6.3.3 Member Function Documentation

6.3.3.1 avoidance_algorithm()

```
QList< SystemResource > EliminateCircularWait::avoidance_algorithm (
            QList< SystemResource > neededResources )
```

avoidance_algorithm algorithm which sorts the neededResources to avoid a deadlock

Parameters

neededResources	List of the SystemResources the process needs to work currectly

Returns

returns the sorted list of resources

6.3.3.2 findNextResource()

```
QList< int > EliminateCircularWait::findNextResource (
            SystemProcess process )  [override], [virtual]
```

findNextResource function that will find the next resource,

Parameters

process	is the process of the thread
availableResources_E	is an array with the over all available resources
differenceResources_A	is an array with the current available resources
assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation

Returns

result from type @List<int>: first int is nextResource, second int is nextResourceCount, third int is indexResourceList

Reimplemented from deadlock_avoidance_api.

The documentation for this class was generated from the following files:

- Algorithms/eliminatecircularwait.h
- Algorithms/eliminatecircularwait.cpp

6.4 EliminateHoldAndWait Class Reference

Class represents the algorithm to eliminate HoldAndWait.

```
#include <eliminateholdandwait.h>
```

Inheritance diagram for EliminateHoldAndWait:

Collaboration diagram for EliminateHoldAndWait:

Public Member Functions

- EliminateHoldAndWait ()

  EliminateHoldAndWait standard constuctor.
- ~EliminateHoldAndWait ()
- QList< int > findNextResource (SystemProcess process) override

  findNextResource function that will find the next resource,
- bool avoidance_algorithm (SystemProcess process, int differenceResources_A[4])

  avoidance_algorithm algorithm which checks for deadlocks

Public Member Functions inherited from deadlock_avoidance_api

- deadlock_avoidance_api ()
- virtual ~deadlock_avoidance_api ()=0
- virtual QList< int > findNextResource (SystemProcess process)

  findNextResource virtual function that will find the next resource,
- virtual void aquireConditionMet (int processId)
- virtual bool checkAquireCondition (int processId)

6.4.1 Detailed Description

Class represents the algorithm to eliminate HoldAndWait.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 EliminateHoldAndWait()

```
EliminateHoldAndWait::EliminateHoldAndWait ( )
```

EliminateHoldAndWait standard constuctor.

6.4.2.2 ~EliminateHoldAndWait()

```
EliminateHoldAndWait::~EliminateHoldAndWait ( )
```

6.4.3 Member Function Documentation

6.4.3.1 avoidance_algorithm()

```
bool EliminateHoldAndWait::avoidance_algorithm (
            SystemProcess process,
            int differenceResources_A[4] )
```

avoidance_algorithm algorithm which checks for deadlocks

Parameters

process	process which runs in the current thread
differenceResources_A	is an array with the current available resources

Returns

returns false if the current state is a deadlock or true if not

6.4.3.2 findNextResource()

```
QList< int > EliminateHoldAndWait::findNextResource (
            SystemProcess process )  [override], [virtual]
```

findNextResource function that will find the next resource,

Parameters

process	is the process of the thread
availableResources_E	is an array with the over all available resources
differenceResources_A	is an array with the current available resources
assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation

Returns

result from type @List<int>: first int is nextResource, second int is nextResourceCount, third int is indexResourceList

Reimplemented from deadlock_avoidance_api.

The documentation for this class was generated from the following files:

- Algorithms/eliminateholdandwait.h
- Algorithms/eliminateholdandwait.cpp

6.5 EndDialog Class Reference

```
#include <enddialog.h>
```

Inheritance diagram for EndDialog:

Collaboration diagram for EndDialog:

Public Slots

- void getEndResults (QString textFromRuntime, int numOfResources, int maxResourceTimeA, int max↩ ResourceTimeB, int maxResourceTimeC)

Public Member Functions

- EndDialog (QWidget *parent=nullptr)

  EndDialog constructor.
- ~EndDialog ()

  destructor

6.5.1 Constructor & Destructor Documentation

6.5.1.1 EndDialog()

```
EndDialog::EndDialog (
            QWidget * parent = nullptr )  [explicit]
```

EndDialog constructor.

Parameters

parent	is the mainwindow

6.5.1.2 ~EndDialog()

```
EndDialog::~EndDialog ( )
```

destructor

6.5.2 Member Function Documentation

6.5.2.1 getEndResults

```
void EndDialog::getEndResults (
            QString textFromRuntime,
            int numOfResources,
            int maxResourceTimeA,
            int maxResourceTimeB,
            int maxResourceTimeC )  [slot]
```

The documentation for this class was generated from the following files:

- Dialogs/enddialog.h
- Dialogs/enddialog.cpp

6.6 MainWindow Class Reference

The MainWindow class represents the main window of the simulation application.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:

Collaboration diagram for MainWindow:

Public Member Functions

- MainWindow (QWidget *parent=nullptr)
  Constructs a MainWindow object.
- ~MainWindow ()
  Destructor for the MainWindow.
- void setUpResourcesAndProcesses (int countPrinters, int countCD, int countPlotters, int countTapeDrive)
  Sets up the initial resources.
- void loadTextFileIntoPlainTextEdit (const QString &filePath)

6.6.1 Detailed Description

The MainWindow class represents the main window of the simulation application.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr )
```

Constructs a MainWindow object.

Parameters

parent	The parent widget (default is nullptr)

6.6.2.2 ~MainWindow()

```
MainWindow::~MainWindow ( )
```

Destructor for the MainWindow.

6.6.3 Member Function Documentation

6.6.3.1 loadTextFileIntoPlainTextEdit()

```
void MainWindow::loadTextFileIntoPlainTextEdit (
            const QString & filePath )
```

6.6.3.2 setUpResourcesAndProcesses()

```
void MainWindow::setUpResourcesAndProcesses (
            int countPrinters,
            int countCD,
            int countPlotters,
            int countTapeDrive )
```

Sets up the initial resources.

Parameters

countPrinters	The count of printers
countCD	The count of CD-ROMs
countPlotters	The count of plotters
countTapeDrive	The count of tape drives

The documentation for this class was generated from the following files:

- Main/mainwindow.h
- Main/mainwindow.cpp

6.7 NoAvoidanceSimulation Class Reference

Class represents the standard algorithm to sort and use the resources, normally leads to a deadlock.

`#include <noavoidancesimulation.h>`

Inheritance diagram for NoAvoidanceSimulation:

Collaboration diagram for NoAvoidanceSimulation:

Public Member Functions

- NoAvoidanceSimulation ()

  NoAvoidanceSimulation standard constructor.
- ~NoAvoidanceSimulation ()

Public Member Functions inherited from deadlock_avoidance_api

- deadlock_avoidance_api ()
- virtual ~deadlock_avoidance_api ()=0
- virtual QList< int > findNextResource (SystemProcess process)

  findNextResource virtual function that will find the next resource,
- virtual void aquireConditionMet (int processId)
- virtual bool checkAquireCondition (int processId)

6.7.1 Detailed Description

Class represents the standard algorithm to sort and use the resources, normally leads to a deadlock.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 NoAvoidanceSimulation()

```
NoAvoidanceSimulation::NoAvoidanceSimulation ( )
```

NoAvoidanceSimulation standard constructor.

6.7.2.2 ~NoAvoidanceSimulation()

```
NoAvoidanceSimulation::~NoAvoidanceSimulation ( )
```

The documentation for this class was generated from the following files:

- Algorithms/noavoidancesimulation.h
- Algorithms/noavoidancesimulation.cpp

6.8 NoPreemption Class Reference

Class represents the algorithm to eliminate NoPreemption.

```
#include <nopreemption.h>
```

Inheritance diagram for NoPreemption:

Collaboration diagram for NoPreemption:

Public Member Functions

- NoPreemption ()

  NoPreemption standard constuctor.
- ~NoPreemption ()
- QList< int > findNextResource (SystemProcess process) override

  findNextResource function that will find the next resource,
- void aquireConditionMet (int processId) override
- bool checkAquireCondition (int processId) override

Public Member Functions inherited from deadlock_avoidance_api

- deadlock_avoidance_api ()
- virtual ~deadlock_avoidance_api ()=0
- virtual QList< int > findNextResource (SystemProcess process)

  findNextResource virtual function that will find the next resource,
- virtual void aquireConditionMet (int processId)
- virtual bool checkAquireCondition (int processId)

Static Public Attributes

- static bool slotPrinterLocked = false
- static bool slotCDLocked = false
- static bool slotPlotterLocked = false
- static bool slotTapeDriveLocked = false
- static bool lastRevokedProcessA = false
- static bool lastRevokedProcessB = false
- static bool lastRevokedProcessC = false
- static QMutex * mutex = new QMutex()

6.8.1 Detailed Description

Class represents the algorithm to eliminate NoPreemption.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 NoPreemption()

```
NoPreemption::NoPreemption ( )
```

NoPreemption standard constuctor.

6.8.2.2 ~NoPreemption()

```
NoPreemption::~NoPreemption ( )
```

6.8.3 Member Function Documentation

6.8.3.1 aquireConditionMet()

```
void NoPreemption::aquireConditionMet (
            int processId )  [override], [virtual]
```

Reimplemented from deadlock_avoidance_api.

6.8.3.2 checkAquireCondition()

```
bool NoPreemption::checkAquireCondition (
            int processId )  [override], [virtual]
```

Reimplemented from deadlock_avoidance_api.

6.8.3.3 findNextResource()

```
QList< int > NoPreemption::findNextResource (
            SystemProcess process )  [override], [virtual]
```

findNextResource function that will find the next resource,

Parameters

process	is the process of the thread
availableResources_E	is an array with the over all available resources
differenceResources_A	is an array with the current available resources
assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation

Returns

result from type @List<int>: first int is nextResource, second int is nextResourceCount, third int is indexResourceList

Reimplemented from deadlock_avoidance_api.

6.8.4 Member Data Documentation

6.8.4.1 lastRevokedProcessA

```
bool NoPreemption::lastRevokedProcessA = false  [static]
```

6.8.4.2 lastRevokedProcessB

```
bool NoPreemption::lastRevokedProcessB = false  [static]
```

6.8.4.3 lastRevokedProcessC

```
bool NoPreemption::lastRevokedProcessC = false  [static]
```

6.8.4.4 mutex

```
QMutex * NoPreemption::mutex = new QMutex()  [static]
```

6.8.4.5 slotCDLocked

```
bool NoPreemption::slotCDLocked = false  [static]
```

6.8.4.6 slotPlotterLocked

```
bool NoPreemption::slotPlotterLocked = false  [static]
```

6.8.4.7 slotPrinterLocked

```
bool NoPreemption::slotPrinterLocked = false  [static]
```

6.8.4.8 slotTapeDriveLocked

```
bool NoPreemption::slotTapeDriveLocked = false  [static]
```

The documentation for this class was generated from the following files:

- Algorithms/nopreemption.h
- Algorithms/nopreemption.cpp

6.9 PreemptionWorker Class Reference

The PreemptionWorker class is for the NoPreemption algorithm to have a seperate Thread which can take resources out of other threads.

```
#include <preemptionworker.h>
```

Inheritance diagram for PreemptionWorker:

Collaboration diagram for PreemptionWorker:

Public Slots

- void reservationStarted (int processId, int nextResource, int nextCount)
  reservationStarted Slot when the reservation starts
- void reservationFinished (int processId, int nextResource, int nextCount, bool notProcessedYet)
  reservationFinished Slot when the reservation is finished
- void initTimers ()
- void revokePrinter ()
  revokePrinter When the Printer surpassed the timeslot it will be revoked
- void revokeCD ()
- void revokePlotter ()
- void revokeTapeDrive ()
- bool getTimerPrinterStatus ()
- bool getTimerCDStatus ()
- bool getTimerPlotterStatus ()
- bool getTimerTapeDriveStatus ()

Signals

- void resourceReleased (int processID, int resource, int count, bool notProcessedYet)

Public Member Functions

- PreemptionWorker (QObject *parent=0)

6.9.1 Detailed Description

The PreemptionWorker class is for the NoPreemption algorithm to have a seperate Thread which can take resources out of other threads.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 PreemptionWorker()

```
PreemptionWorker::PreemptionWorker (
            QObject * parent = 0 )  [explicit]
```

6.9.3 Member Function Documentation

6.9.3.1 getTimerCDStatus

```
bool PreemptionWorker::getTimerCDStatus ( )  [inline], [slot]
```

6.9.3.2 getTimerPlotterStatus

```
bool PreemptionWorker::getTimerPlotterStatus ( )  [inline], [slot]
```

6.9.3.3 getTimerPrinterStatus

```
bool PreemptionWorker::getTimerPrinterStatus ( )  [inline], [slot]
```

6.9.3.4 getTimerTapeDriveStatus

```
bool PreemptionWorker::getTimerTapeDriveStatus ( )  [inline], [slot]
```

6.9.3.5 initTimers

```
void PreemptionWorker::initTimers ( )  [slot]
```

6.9.3.6 reservationFinished

```
void PreemptionWorker::reservationFinished (
            int processId,
            int nextResource,
            int nextCount,
            bool notProcessedYet )  [slot]
```

reservationFinished Slot when the reservation is finished

Parameters

processId	ID from the called process
nextResource	is the next needed resource
nextCount	count of the next needed resource
notProcessedYet	true if some resources are not proccessed yet

6.9.3.7 reservationStarted

```
void PreemptionWorker::reservationStarted (
            int processId,
            int nextResource,
            int nextCount )  [slot]
```

reservationStarted Slot when the reservation starts

Parameters

processId	ID from the called process
nextResource	is the next needed resource
nextCount	count of the next needed resource

6.9.3.8 resourceReleased

```
void PreemptionWorker::resourceReleased (
            int processID,
            int resource,
            int count,
            bool notProcessedYet )  [signal]
```

6.9.3.9 revokeCD

```
void PreemptionWorker::revokeCD ( )  [slot]
```

6.9.3.10 revokePlotter

```
void PreemptionWorker::revokePlotter ( )  [slot]
```

6.9.3.11 revokePrinter

```
void PreemptionWorker::revokePrinter ( )  [slot]
```

revokePrinter When the Printer surpassed the timeslot it will be revoked

revokeCD When the CD surpassed the timeslot it will be revoked

revokePlotter When the Plotter surpassed the timeslot it will be revoked

revokeTapeDrive When the TapeDrive surpassed the timeslot it will be revoked

6.9.3.12 revokeTapeDrive

```
void PreemptionWorker::revokeTapeDrive ( )  [slot]
```

The documentation for this class was generated from the following files:

- Algorithms/preemptionworker.h
- Algorithms/preemptionworker.cpp

6.10 ProcessWorker Class Reference

Class represents the process worker which is responsible for the whole simulation process.

```
#include <processworker.h>
```

Inheritance diagram for ProcessWorker:

Collaboration diagram for ProcessWorker:

Public Slots

- void requestResource ()

  requestResource a slot that will request the resources a process needs until all resources were processed. It will notify the main thread throughout the process about changes in the resource occupation

Signals

- void startedAcquire (int processId, int nextResource, int nextCount)
- void resourceReserved (int processID, int resource, int count)

  resourceReserved notifies the main thread, that the given resource has been reserved and can not be used anymore
- void resourceReleased (int processID, int resource, int count, bool notProcessedYet)

  resourceReleased notifies the main thread, that the given resource has been released
- void finishedResourceProcessing (int processID)

  finishedResourceProcessing indicates that a process has released all resources it requires and is finished

Public Member Functions

- ProcessWorker (SystemProcess processes, int selectedAlgorithm, QObject ∗parent=0)

  creates a Process Worker with the given parameters and also initializes the semaphores
- void updateProcess (int nextResource, int countResource)

  updateProcess updates the neededResources List in the member Process
- void setUpOccupations (int assignedResources_C[3][4], int stillNeededResources_R[3][4])

  setUpOccupations used to copy the assignedResources_C and stillNeededResources_R from mainwindow to the threads
- void setAlgorithm (int algorithm)

  setAlgorithm sets the algorithm used for the simulation
- void printStillNeeded ()

Static Public Attributes

- static QSemaphore * semaphorePrinter
  to keep track of available Printers
- static QSemaphore * semaphoreCD
  to keep track of available CDs
- static QSemaphore * semaphorePlotter
  to keep track of available Plotters
- static QSemaphore * semaphoreTapeDrive
  to keep track of available TapeDrives
- static int differenceResources_A [4]
  is an array with the current available resources
- static int availableResources_E [4]
  is an array with the over all available resources
- static int assignedResources_C [3][4]
  containing which and how many resources a process is occupying
- static int stillNeededResources_R [3][4]
  matrix containing which and how many resources a process will still need to occupie throughout the simulation

6.10.1 Detailed Description

Class represents the process worker which is responsible for the whole simulation process.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ProcessWorker()

```
ProcessWorker::ProcessWorker (
            SystemProcess processes,
            int selectedAlgorithm,
            QObject * parent = 0 )  [explicit]
```

creates a Process Worker with the given parameters and also initializes the semaphores

Parameters

processes	is the process belonging to the worker object
availableResources_E	is an int array with 4 ints. Each int corresponds to a resource and how many are available at the start
differenceResources_A	is an int array with 4 ints. Each int corresponds to a resource that is available right now

6.10.3 Member Function Documentation

6.10.3.1 finishedResourceProcessing

```
void ProcessWorker::finishedResourceProcessing (
            int processID )  [signal]
```

finishedResourceProcessing indicates that a process has released all resources it requires and is finished

Parameters

information	unused

6.10.3.2 printStillNeeded()

```
void ProcessWorker::printStillNeeded ( )  [inline]
```

6.10.3.3 requestResource

```
void ProcessWorker::requestResource ( )  [slot]
```

requestResource a slot that will request the resources a process needs until all resources were processed. It will notify the main thread throughout the process about changes in the resource occupation

6.10.3.4 resourceReleased

```
void ProcessWorker::resourceReleased (
            int processID,
            int resource,
            int count,
            bool notProcessedYet )  [signal]
```

resourceReleased notifies the main thread, that the given resource has been released

Parameters

processID	is the ID of the process member to determin the process in the main thread
resource	is the resource ID of the resource that has been released
count	is the count of released resources

6.10.3.5 resourceReserved

```
void ProcessWorker::resourceReserved (
            int processID,
            int resource,
            int count )  [signal]
```

resourceReserved notifies the main thread, that the given resource has been reserved and can not be used anymore

Parameters

processID	is the ID of the process member to determin the process in the main thread
resource	is the resource ID of the resource that has been reserved
count	is the count of reserved resources

6.10.3.6 setAlgorithm()

```
void ProcessWorker::setAlgorithm (
            int algorithm )  [inline]
```

setAlgorithm sets the algorithm used for the simulation

Parameters

algorithm	number to choose the algorithm

6.10.3.7 setUpOccupations()

```
void ProcessWorker::setUpOccupations (
            int assignedResources_C[3][4],
            int stillNeededResources_R[3][4] )  [inline]
```

setUpOccupations used to copy the assignedResources_C and stillNeededResources_R from mainwindow to the threads

Parameters

assignedResources_C	matrix containing which and how many resources a process is occupying
stillNeededResources_R	matix containing which and how many resources a process will still need to occupie throughout the simulation

6.10.3.8 startedAcquire

```
void ProcessWorker::startedAcquire (
            int processId,
            int nextResource,
            int nextCount )  [signal]
```

6.10.3.9 updateProcess()

```
void ProcessWorker::updateProcess (
            int nextResource,
            int countResource )
```

updateProcess updates the neededResources List in the member Process

Parameters

nextResource	is the ID of the resource that has been reserved
countResource	is the count of the reserce

6.10.4 Member Data Documentation

6.10.4.1 assignedResources_C

```
int ProcessWorker::assignedResources_C  [static]
```

containing which and how many resources a process is occupying

6.10.4.2 availableResources_E

```
int ProcessWorker::availableResources_E  [static]
```

is an array with the over all available resources

6.10.4.3 differenceResources_A

```
int ProcessWorker::differenceResources_A  [static]
```

is an array with the current available resources

6.10.4.4 semaphoreCD

```
QSemaphore * ProcessWorker::semaphoreCD  [static]
```

to keep track of available CDs

6.10.4.5 semaphorePlotter

```
QSemaphore * ProcessWorker::semaphorePlotter  [static]
```

to keep track of available Plotters

6.10.4.6 semaphorePrinter

```
QSemaphore * ProcessWorker::semaphorePrinter  [static]
```

to keep track of available Printers

semaphorePrinter regulates how many printers can be used by threads

semaphoreCD regulates how many cds can be used by threads

semaphorePlotter regulates how many plotters can be used by threads

semaphoreTapeDrive regulates how many tape drives can be used by threads

6.10.4.7 semaphoreTapeDrive

```
QSemaphore * ProcessWorker::semaphoreTapeDrive  [static]
```

to keep track of available TapeDrives

6.10.4.8 stillNeededResources_R

```
int ProcessWorker::stillNeededResources_R  [static]
```

matrix containing which and how many resources a process will still need to occupie throughout the simulation

The documentation for this class was generated from the following files:

- Objects/processworker.h
- Objects/processworker.cpp

6.11 StartDialog Class Reference

The StartDialog class represents the initial menu for selecting the algorithm and the number of resources the user wishes to use for the simulation.

```
#include <startdialog.h>
```

Inheritance diagram for StartDialog:

Collaboration diagram for StartDialog:

Public Slots

- void getResourceCount ()

  Slot to retrieve the chosen resource count from the user.
- void getAlgorithm ()

  Slot to retrieve the chosen algorithm from the user.

Signals

- void countsFinished (int *resourcesCounts)

  Signal emitted when resource counts are finished being collected.
- void algorithmsFinished (int algorithm)

  Signal emitted when the algorithm is finished.

Public Member Functions

- StartDialog (QWidget *parent=nullptr)

  Constructs a StartDialog object.
- ~StartDialog ()

  Destructor for the StartDialog.

6.11.1 Detailed Description

The StartDialog class represents the initial menu for selecting the algorithm and the number of resources the user wishes to use for the simulation.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 StartDialog()

```
StartDialog::StartDialog (
            QWidget * parent = nullptr )  [explicit]
```

Constructs a StartDialog object.

Parameters

parent	The parent widget (default is nullptr)

6.11.2.2 ~StartDialog()

```
StartDialog::~StartDialog ( )
```

Destructor for the StartDialog.

6.11.3 Member Function Documentation

6.11.3.1 algorithmsFinished

```
void StartDialog::algorithmsFinished (
            int algorithm )  [signal]
```

Signal emitted when the algorithm is finished.

Parameters

algorithm	The chosen algorithm identifier

6.11.3.2 countsFinished

```
void StartDialog::countsFinished (
            int * resourcesCounts )  [signal]
```

Signal emitted when resource counts are finished being collected.

Parameters

resourcesCounts	An array of resource counts

6.11.3.3 getAlgorithm

```
void StartDialog::getAlgorithm ( )  [slot]
```

Slot to retrieve the chosen algorithm from the user.

6.11.3.4 getResourceCount

```
void StartDialog::getResourceCount ( )  [slot]
```

Slot to retrieve the chosen resource count from the user.

The documentation for this class was generated from the following files:

- Dialogs/startdialog.h
- Dialogs/startdialog.cpp

6.12 SystemProcess Class Reference

Class represents the processes which use the resources.

```
#include <systemprocess.h>
```

Public Member Functions

- SystemProcess ()

  SystemProcess standard constuctor with no parameter.
- SystemProcess (QString name, int processId)

  SystemProcess constructor setting name and processID.
- SystemProcess (QString name, int processId, int maxPrinters, int maxCDs, int maxPlotters, int maxTape↩ Drives)

  SystemProcess constructor setting name and processId and filling neededResources with a resource of each type with a count between min and max.
- int getProcessId () const

  getProcessId getter for processId
- void setProcessId (int processId)

  setProcessId setter for processId
- QString getName () const

  getName getter for name
- void setName (QString name)

  setName setter for name
- int getRevokedResourceId () const
- void setRevokedResourceId (int revokedResourceId)
- const QList< SystemResource > getNeededResources () const

  getNeededResources list of needed resources
- void setNeededResources (QList< SystemResource > neededResources)

  setNeededResources setting the neededResources list
- void moveNeededResourceToBack (int index)

  moveNeededResourceToBack the needed resource will be placed at the end of the list
- void shuffleNeededResources ()
- void printNeededResources ()

6.12.1 Detailed Description

Class represents the processes which use the resources.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 SystemProcess() [1/3]

```
SystemProcess::SystemProcess ( )  [inline]
```

SystemProcess standard constuctor with no parameter.

6.12.2.2 SystemProcess() [2/3]

```
SystemProcess::SystemProcess (
            QString name,
            int processId )
```

SystemProcess constructor setting name and processID.

Parameters

name	of a process
processId	(corresponding to name: A = id 1...)

6.12.2.3 SystemProcess() [3/3]

```
SystemProcess::SystemProcess (
            QString name,
            int processId,
            int maxPrinters,
            int maxCDs,
            int maxPlotters,
            int maxTapeDrives )
```

SystemProcess constructor setting name and processId and filling neededResources with a resource of each type with a count between min and max.

Parameters

name	of a process
processId	(corresponding to name: A = id 1...)
min	the minimum count a resource in neededResources can have
max	the maximum count a resource in neededResources can have

6.12.3 Member Function Documentation

6.12.3.1 getName()

```
QString SystemProcess::getName ( ) const  [inline]
```

getName getter for name

Returns

name

6.12.3.2 getNeededResources()

```
const QList< SystemResource > SystemProcess::getNeededResources ( ) const  [inline]
```

getNeededResources list of needed resources

Returns

neededResources

6.12.3.3 getProcessId()

```
int SystemProcess::getProcessId ( ) const  [inline]
```

getProcessId getter for processId

Returns

processId

6.12.3.4 getRevokedResourceId()

```
int SystemProcess::getRevokedResourceId ( ) const  [inline]
```

6.12.3.5 moveNeededResourceToBack()

```
void SystemProcess::moveNeededResourceToBack (
            int index )  [inline]
```

moveNeededResourceToBack the needed resource will be placed at the end of the list

Parameters

index	placed at index

6.12.3.6 printNeededResources()

```
void SystemProcess::printNeededResources ( )  [inline]
```

6.12.3.7 setName()

```
void SystemProcess::setName (
            QString name )  [inline]
```

setName setter for name

Parameters

name	

6.12.3.8 setNeededResources()

```
void SystemProcess::setNeededResources (
            QList< SystemResource > neededResources )  [inline]
```

setNeededResources setting the neededResources list

Parameters

neededResources	

6.12.3.9 setProcessId()

```
void SystemProcess::setProcessId (
            int processId )  [inline]
```

setProcessId setter for processId

Parameters

processId	

6.12.3.10 setRevokedResourceId()

```
void SystemProcess::setRevokedResourceId (
            int revokedResourceId )  [inline]
```

6.12.3.11 shuffleNeededResources()

```
void SystemProcess::shuffleNeededResources ( )  [inline]
```

The documentation for this class was generated from the following files:

- Objects/systemprocess.h
- Objects/systemprocess.cpp

6.13 SystemResource Class Reference

Class which represents the resources used by processes.

```
#include <systemresource.h>
```

Public Member Functions

- SystemResource (QString name, int resourceId, int count)
  SystemResource constructor.
- int getResourceId () const
  getResourceId getter for resourceId
- void setResourceId (int resourceId)
  setResourceId setter for resourceId
- const QString & getName () const
  getName getter for name
- void setName (const QString &name)
  setName setter for name
- int getCount () const
  getCount getter for count
- void setCount (int count)
  setCount setter for count
- void decreaseCount (int count)
  decreaseCount decreases the count
- bool operator== (const SystemResource &otherResource) const

6.13.1 Detailed Description

Class which represents the resources used by processes.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 SystemResource()

```
SystemResource::SystemResource (
            QString name,
            int resourceId,
            int count )
```

SystemResource constructor.

Parameters

name	of a process
resource↩ Id	corresponds to a resource: (0 = Printer, 1 = CD-ROM, 2 = Plotter, 3 = TapeDrive)
count	of the resource (how many are physically available)

6.13.3 Member Function Documentation

6.13.3.1 decreaseCount()

```
void SystemResource::decreaseCount (
            int count )  [inline]
```

decreaseCount decreases the count

Parameters

count	

6.13.3.2 getCount()

```
int SystemResource::getCount ( ) const  [inline]
```

getCount getter for count

Returns

count

6.13.3.3 getName()

```
const QString & SystemResource::getName ( ) const  [inline]
```

getName getter for name

Returns

name

6.13.3.4 getResourceId()

```
int SystemResource::getResourceId ( ) const  [inline]
```

getResourceId getter for resourceId

Returns

resourceId

6.13.3.5 operator==()

```
bool SystemResource::operator== (
            const SystemResource & otherResource ) const  [inline]
```

6.13.3.6 setCount()

```
void SystemResource::setCount (
            int count )  [inline]
```

setCount setter for count

Parameters

count	

6.13.3.7 setName()

```
void SystemResource::setName (
            const QString & name )  [inline]
```

setName setter for name

Parameters

name	

6.13.3.8 setResourceId()

```
void SystemResource::setResourceId (
            int resourceId )
```

setResourceId setter for resourceId

Parameters

resourceId	

The documentation for this class was generated from the following files:

- Objects/systemresource.h
- Objects/systemresource.cpp

Chapter 7

File Documentation

7.1 Algorithms/bankiersalgorithm.cpp File Reference

```
#include ¨bankiersalgorithm.h¨
#include ¨QDebug¨
#include ¨qmutex.h¨
#include <QThread>
#include <Objects/ProcessWorker.h>
```

Include dependency graph for bankiersalgorithm.cpp:

Variables

- QMutex ∗ mutexOne = new QMutex()

7.1.1 Variable Documentation

7.1.1.1 mutexOne

```
QMutex* mutexOne = new QMutex()
```

7.2 Algorithms/bankiersalgorithm.h File Reference

```
#include <Objects/systemprocess.h>
#include <Algorithms/deadlock_avoidance_api.h>
```

Include dependency graph for bankiersalgorithm.h:

7.3 bankiersalgorithm.h

Go to the documentation of this file.

```
00001 #ifndef BANKIERSALGORITHM_H
00002 #define BANKIERSALGORITHM_H
00003
00004 #include <Objects/systemprocess.h>
00005 #include <Algorithms/deadlock_avoidance_api.h>
00006
00007 using namespace std;
00008
00012 class BankiersAlgorithm : public deadlock_avoidance_api
00013 {
00014 public:
00018     BankiersAlgorithm();
00019     ~BankiersAlgorithm();
00020
00021
00031     QList<int> findNextResource(SystemProcess process)  override;
00032
00041     bool avoidance_algorithm();
00042 };
00043
00044 #endif // BANKIERSALGORITHM_H
```

7.4 bankiersalgorithm.h File Reference

```
#include <Objects/systemprocess.h>
#include <Algorithms/deadlock_avoidance_api.h>
```

Include dependency graph for bankiersalgorithm.h:

Classes

- class BankiersAlgorithm

  Class represents the Bankier Algorithm to prevent Deadlocks.

7.5 bankiersalgorithm.h

Go to the documentation of this file.

```
00001 #ifndef BANKIERSALGORITHM_H
00002 #define BANKIERSALGORITHM_H
00003
00004 #include <Objects/systemprocess.h>
00005 #include <Algorithms/deadlock_avoidance_api.h>
00006
00007 using namespace std;
00008
00009 class BankiersAlgorithm : public deadlock_avoidance_api
00010 {
00011 public:
00012
00013     BankiersAlgorithm();
00014
00015
00026     QList<int> findNextResource(SystemProcess process, int stillNeededResources_RCopy[3][4], int
      assignedResources_CCopy[3][4], int differenceResources_ACopy[4], int availableResources_ECopy[4]);
00027
00036     bool avoidance_algorithm(int stillNeededResources_R[3][4], int assignedResources_C[3][4], int
      differenceResources_A[4], int availableResources_E[4]);
00037 };
00038
00039 #endif // BANKIERSALGORITHM_H
```

7.6 Algorithms/deadlock_avoidance_api.cpp File Reference

```
#include ¨deadlock_avoidance_api.h¨
#include <Objects/ProcessWorker.h>
```

Include dependency graph for deadlock_avoidance_api.cpp:

7.7 Algorithms/deadlock_avoidance_api.h File Reference

```
#include <Objects/SystemProcess.h>
#include <QSemaphore>
#include <QDebug>
```

Include dependency graph for deadlock_avoidance_api.h: This graph shows which files directly or indirectly include this file:

Classes

- class deadlock_avoidance_api

  The virtual class represents an API the algorithms use.

7.8 deadlock_avoidance_api.h

Go to the documentation of this file.

```
00001 #ifndef DEADLOCK_AVOIDANCE_API_H
00002 #define DEADLOCK_AVOIDANCE_API_H
00003
00004 #include <Objects/SystemProcess.h>
00005 #include <QSemaphore>
00006 #include <QDebug>
00007
00011 class deadlock_avoidance_api
00012 {
00013 public:
00014     deadlock_avoidance_api();
00015
00016     virtual ~deadlock_avoidance_api() = 0;
00026     virtual QList<int> findNextResource(SystemProcess process);
00027
00028     virtual void aquireConditionMet(int processId){
00029     }
00030
00031     virtual bool checkAquireCondition(int processId){
00032         return false;
00033     }
00034 };
00035
00036 #endif // DEADLOCK_AVOIDANCE_API_H
```

7.9 Algorithms/eliminatecircularwait.cpp File Reference

```
#include ¨eliminatecircularwait.h¨
#include <Objects/ProcessWorker.h>
```

Include dependency graph for eliminatecircularwait.cpp:

7.10 Algorithms/eliminatecircularwait.h File Reference

```
#include <Algorithms/deadlock_avoidance_api.h>
```

Include dependency graph for eliminatecircularwait.h: This graph shows which files directly or indirectly include this file:

Classes

- class EliminateCircularWait

  Class represents the algorithm to eliminate CircularWait.

7.11 eliminatecircularwait.h

Go to the documentation of this file.

```
00001 #ifndef ELIMINATECIRCULARWAIT_H
00002 #define ELIMINATECIRCULARWAIT_H
00003
00004 #include <Algorithms/deadlock_avoidance_api.h>
00005
00006
00010 class EliminateCircularWait : public deadlock_avoidance_api
00011 {
00012 public:
00016     EliminateCircularWait();
00017     ~EliminateCircularWait();
00018
00028     QList<int> findNextResource(SystemProcess process) override;
00029
00035     QList<SystemResource> avoidance_algorithm(QList<SystemResource> neededResources);
00036 };
00037
00038 #endif // ELIMINATECIRCULARWAIT_H
```

7.12 Algorithms/eliminateholdandwait.cpp File Reference

```
#include ¨eliminateholdandwait.h¨
#include <QDebug>
#include <Objects/ProcessWorker.h>
```

Include dependency graph for eliminateholdandwait.cpp:

7.13 Algorithms/eliminateholdandwait.h File Reference

```
#include <Algorithms/deadlock_avoidance_api.h>
#include <QSemaphore>
```

Include dependency graph for eliminateholdandwait.h: This graph shows which files directly or indirectly include this file:

Classes

- class EliminateHoldAndWait

  Class represents the algorithm to eliminate HoldAndWait.

7.14 eliminateholdandwait.h

Go to the documentation of this file.

```
00001 #ifndef ELIMINATEHOLDANDWAIT_H
00002 #define ELIMINATEHOLDANDWAIT_H
00003
00004 #include <Algorithms/deadlock_avoidance_api.h>
00005 #include <QSemaphore>
00006
00007
00011 class EliminateHoldAndWait : public deadlock_avoidance_api
00012 {
00013 public:
00017     EliminateHoldAndWait();
00018     ~EliminateHoldAndWait();
00019
00029     QList<int> findNextResource(SystemProcess process) override;
00030
00037     bool avoidance_algorithm(SystemProcess process, int differenceResources_A[4]);
00038
00039 private:
00040     static QSemaphore *semaphore;
00041     static QList<int>currentProcess;
00042     static int copyDifference[4];
00043 };
00044
00045
00046 #endif // ELIMINATEHOLDANDWAIT_H
```

7.15 Algorithms/noavoidancesimulation.cpp File Reference

```
#include ¨noavoidancesimulation.h¨
#include ¨QDebug¨
```

Include dependency graph for noavoidancesimulation.cpp:

7.16 Algorithms/noavoidancesimulation.h File Reference

```
#include <Algorithms/deadlock_avoidance_api.h>
```

Include dependency graph for noavoidancesimulation.h: This graph shows which files directly or indirectly include this file:

Classes

- class NoAvoidanceSimulation

  Class represents the standard algorithm to sort and use the resources, normally leads to a deadlock.

7.17 noavoidancesimulation.h

Go to the documentation of this file.

```
00001 #ifndef NOAVOIDANCESIMULATION_H
00002 #define NOAVOIDANCESIMULATION_H
00003
00004 #include <Algorithms/deadlock_avoidance_api.h>
00005
00009 class NoAvoidanceSimulation : public deadlock_avoidance_api
00010 {
00011 public:
00015     NoAvoidanceSimulation();
00016     ~NoAvoidanceSimulation();
00017 };
00018
00019 #endif // NOAVOIDANCESIMULATION_H
```

7.18 Algorithms/nopreemption.cpp File Reference

```
#include ¨nopreemption.h¨
#include <Objects/ProcessWorker.h>
```
Include dependency graph for nopreemption.cpp:

7.19 Algorithms/nopreemption.h File Reference

```
#include <Algorithms/deadlock_avoidance_api.h>
```
Include dependency graph for nopreemption.h: This graph shows which files directly or indirectly include this file:

Classes

- class NoPreemption

  Class represents the algorithm to eliminate NoPreemption.

7.20 nopreemption.h

Go to the documentation of this file.

```
00001 #ifndef NOPREEMPTION_H
00002 #define NOPREEMPTION_H
00003
00004 #include <Algorithms/deadlock_avoidance_api.h>
00005
00006
00010 class NoPreemption : public deadlock_avoidance_api
00011 {
00012 public:
00016     NoPreemption();
00017     ~NoPreemption();
00018
00028     QList<int> findNextResource(SystemProcess process) override;
00029
00030     void aquireConditionMet(int processId) override;
00031     bool checkAquireCondition(int processId) override;
00032
00033
00034 public:
00035     static bool slotPrinterLocked;
00036     static bool slotCDLocked;
00037     static bool slotPlotterLocked;
00038     static bool slotTapeDriveLocked;
00039     static bool lastRevokedProcessA, lastRevokedProcessB, lastRevokedProcessC;
00040     static QMutex *mutex;
00041 };
00042
00043 #endif // NOPREEMPTION_H
```

7.21 Algorithms/preemptionworker.cpp File Reference

```
#include ¨preemptionworker.h¨
#include ¨Objects/ProcessWorker.h¨
```
Include dependency graph for preemptionworker.cpp:

Variables

- int maxWaitTime

7.21.1 Variable Documentation

7.21.1.1 maxWaitTime

```
int maxWaitTime
```

7.22 Algorithms/preemptionworker.h File Reference

```
#include ¨qtimer.h¨
#include <QObject>
#include <Objects/SystemProcess.h>
```

Include dependency graph for preemptionworker.h: This graph shows which files directly or indirectly include this file:

Classes

- class PreemptionWorker

  The PreemptionWorker class is for the NoPreemption algorithm to have a seperate Thread which can take resources out of other threads.

7.23 preemptionworker.h

Go to the documentation of this file.

```
00001 #ifndef PREEMPTIONWORKER_H
00002 #define PREEMPTIONWORKER_H
00003
00004 #include "qtimer.h"
00005 #include <QObject>
00006
00007 #include <Objects/SystemProcess.h>
00008
00012 class PreemptionWorker : public QObject
00013 {
00014     Q_OBJECT
00015 public:
00016     explicit PreemptionWorker(QObject *parent = 0);
00017
00018
00019 public slots:
00026     void reservationStarted(int processId, int nextResource, int nextCount);
00027
00035     void reservationFinished(int processId, int nextResource, int nextCount, bool notProcessedYet);
00036
00037     void initTimers();
00038
00045     void revokePrinter();
00046     void revokeCD();
00047     void revokePlotter();
00048     void revokeTapeDrive();
00049
00050     bool getTimerPrinterStatus(){
00051         return timerPrinter->isActive();
00052     }
00053
00054     bool getTimerCDStatus(){
00055         return timerCD->isActive();
00056     }
00057
00058     bool getTimerPlotterStatus(){
00059         return timerPlotter->isActive();
00060     }
00061
00062     bool getTimerTapeDriveStatus(){
00063         return timerTapeDrive->isActive();
```

```
00064     }
00065
00066 signals:
00067     void resourceReleased(int processID, int resource, int count, bool notProcessedYet);
00068
00069 private:
00070     QTimer *timerPrinter, *timerCD, *timerPlotter, *timerTapeDrive;
00071     int nextPrinterResource = -1, nextCDResource = -1,nextPlotterResource = -1, nextTapeDriveResource
    = -1;
00072     int nextPrinterCount = 0, nextCDCount = 0, nextPlotterCount = 0, nextTapeDriveCount = 0;
00073     int processPrinter, processCD, processPlotter, processTapeDrive;
00074 };
00075
00076 #endif // PREEMPTIONWORKER_H
```

7.24 Algorithms/roundrobinscheduling.cpp File Reference

```
#include ¨roundrobinscheduling.h¨
#include <QtConcurrent>
```

Include dependency graph for roundrobinscheduling.cpp:

7.25 Dialogs/enddialog.cpp File Reference

```
#include ¨enddialog.h¨
#include ¨ui_enddialog.h¨
#include <QGraphicsDropShadowEffect>
#include <QMainWindow>
#include <qprocess.h>
```

Include dependency graph for enddialog.cpp:

7.26 Dialogs/enddialog.h File Reference

```
#include <QDialog>
```

Include dependency graph for enddialog.h: This graph shows which files directly or indirectly include this file:

Classes

- class EndDialog

Namespaces

- namespace Ui

7.27 enddialog.h

Go to the documentation of this file.

```
00001 #ifndef ENDDIALOG_H
00002 #define ENDDIALOG_H
00003
00004 #include <QDialog>
00005
00006 namespace Ui {
00007
00011 class EndDialog;
00012 }
00013
00014 class EndDialog : public QDialog
00015 {
00016     Q_OBJECT
00017
00018 public:
00023     explicit EndDialog(QWidget *parent = nullptr);
00027     ~EndDialog();
00028
00029 private slots:
00033     void on_end_pushButton_clicked();
00034
00035     void on_restart_pushButton_clicked();
00036
00037 public slots:
00038
00039     void getEndResults(QString textFromRuntime, int numOfResources, int maxResourceTimeA, int
      maxResourceTimeB, int maxResourceTimeC);
00040
00041 private:
00042     Ui::EndDialog *ui;
00043 };
00044
00045 #endif // ENDDIALOG_H
```

7.28 Dialogs/startdialog.cpp File Reference

```
#include ¨startdialog.h¨
#include ¨ui_startdialog.h¨
#include <QGraphicsDropShadowEffect>
#include <QDesktopServices>
```

Include dependency graph for startdialog.cpp:

7.29 Dialogs/startdialog.h File Reference

```
#include <QDialog>
```

Include dependency graph for startdialog.h: This graph shows which files directly or indirectly include this file:

Classes

- class StartDialog

  The StartDialog class represents the initial menu for selecting the algorithm and the number of resources the user wishes to use for the simulation.

Namespaces

- namespace Ui

7.30 startdialog.h

Go to the documentation of this file.

```
00001 #ifndef STARTDIALOG_H
00002 #define STARTDIALOG_H
00003
00004 #include <QDialog>
00005
00006 namespace Ui {
00007 class StartDialog;
00008 }
00009
00014 class StartDialog : public QDialog
00015 {
00016     Q_OBJECT
00017
00018 public:
00023     explicit StartDialog(QWidget *parent = nullptr);
00024
00028     ~StartDialog();
00029
00030 public slots:
00034     void getResourceCount();
00035
00039     void getAlgorithm();
00040
00041 signals:
00046     void countsFinished(int* resourcesCounts);
00047
00052     void algorithmsFinished(int algorithm);
00053
00054 private slots:
00055     void on_openGithubButton_clicked();
00056
00057 private:
00058     Ui::StartDialog *ui;
00059 };
00060
00061 #endif // STARTDIALOG_H
```

7.31 Main/main.cpp File Reference

```
#include ¨mainwindow.h¨
#include <QApplication>
```

Include dependency graph for main.cpp:

Functions

- int main (int argc, char ∗argv[ ])

7.31.1 Function Documentation

7.31.1.1 main()

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

7.32 Main/mainwindow.cpp File Reference

```
#include ¨mainwindow.h¨
#include ¨ui_mainwindow.h¨
#include <Objects/ProcessWorker.h>
#include <QProcess>
#include <QApplication>
#include <QThread>
#include <QTime>
#include <QTimer>
#include <Dialogs/StartDialog.h>
#include <Dialogs/EndDialog.h>
#include <QGraphicsDropShadowEffect>
#include <QFont>
#include <QFile>
```

Include dependency graph for mainwindow.cpp:

Variables

- const int system_resource_count = 4

  how many resources the system has (constant)
- const int system_process_count = 3

  how many processes the system has (constant)
- int existingResources [4]

  the count of how many of each resource type exist
- int selectedAlgorithmNumber = -1

  the selected algorithm
- int finished = 0

  count for completely finished processes
- int countAllResourcesUsed = 0

  count of all the resources used throughout the simulation, displayed at the end
- QTimer * timer
- QElapsedTimer * processATimer = new QElapsedTimer()
- QList< int > * processATimeList = new QList<int>()
- QElapsedTimer * processBTimer = new QElapsedTimer()
- QList< int > * processBTimeList = new QList<int>()
- QElapsedTimer * processCTimer = new QElapsedTimer()
- QList< int > * processCTimeList = new QList<int>()
- int assignedResources_C [3][4]
- int stillNeededResources_R [3][4] = {{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0}}
- int availableResources_E [4]
- int occupiedResources_P [4] = {0, 0, 0, 0}
- int differenceResources_A [4]
- QList< SystemProcess > processes

7.32.1 Variable Documentation

7.32.1.1 assignedResources_C

```
int assignedResources_C[3][4]
```

7.32.1.2 availableResources_E

```
int availableResources_E[4]
```

7.32.1.3 countAllResourcesUsed

```
int countAllResourcesUsed = 0
```

count of all the resources used throughout the simulation, displayed at the end

7.32.1.4 differenceResources_A

```
int differenceResources_A[4]
```

7.32.1.5 existingResources

```
int existingResources[4]
```

the count of how many of each resource type exist

7.32.1.6 finished

```
int finished = 0
```

count for completely finished processes

7.32.1.7 occupiedResources_P

```
int occupiedResources_P[4] = {0, 0, 0, 0}
```

7.32.1.8 processATimeList

```
QList<int>* processATimeList = new QList<int>()
```

7.32.1.9 processATimer

```
QElapsedTimer* processATimer = new QElapsedTimer()
```

7.32.1.10 processBTimeList

```
QList<int>* processBTimeList = new QList<int>()
```

7.32.1.11 processBTimer

```
QElapsedTimer* processBTimer = new QElapsedTimer()
```

7.32.1.12 processCTimeList

```
QList<int>* processCTimeList = new QList<int>()
```

7.32.1.13 processCTimer

```
QElapsedTimer* processCTimer = new QElapsedTimer()
```

7.32.1.14 processes

```
QList<SystemProcess> processes
```

7.32.1.15 selectedAlgorithmNumber

```
int selectedAlgorithmNumber = -1
```

the selected algorithm

7.32.1.16 stillNeededResources_R

```
int stillNeededResources_R[3][4] = {{0, 0, 0, 0},{0, 0, 0, 0},{0, 0, 0, 0}}
```

7.32.1.17 system_process_count

```
const int system_process_count = 3
```

how many processes the system has (constant)

7.32.1.18 system_resource_count

```
const int system_resource_count = 4
```

how many resources the system has (constant)

7.32.1.19 timer

```
QTimer* timer
```

7.33 Main/mainwindow.h File Reference

```
#include <QMainWindow>
#include <QList>
#include <Objects/SystemResource.h>
#include <Objects/SystemProcess.h>
#include <Objects/ProcessWorker.h>
#include <Algorithms/preemptionworker.h>
```

Include dependency graph for mainwindow.h: This graph shows which files directly or indirectly include this file:

Classes

- class MainWindow

  The MainWindow class represents the main window of the simulation application.

Namespaces

- namespace Ui

7.34 mainwindow.h

Go to the documentation of this file.

```
00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005 #include <QList>
00006 #include <Objects/SystemResource.h>
00007 #include <Objects/SystemProcess.h>
00008 #include <Objects/ProcessWorker.h>
00009 #include <Algorithms/preemptionworker.h>
00010
00011 QT_BEGIN_NAMESPACE
00012 namespace Ui { class MainWindow; }
00013 QT_END_NAMESPACE
00014
00018 class MainWindow : public QMainWindow
00019 {
00020     Q_OBJECT
00021
00022 public:
00027     MainWindow(QWidget *parent = nullptr);
00028
00032     ~MainWindow();
00033
00041     void setUpResourcesAndProcesses(int countPrinters, int countCD, int countPlotters, int
      countTapeDrive);
00042
00043     void loadTextFileIntoPlainTextEdit(const QString &filePath);
00044
00045 private slots:
00047     void update_occupation_matrix();
00048     void update_needed_matrix();
00049     void update_resource_occupation();
00050     void update_resource_occupation_list();
00051     void updateElapsedTime(const QTime &startTime);
00052
00053     void setShadows();
00054     void initResourceCount(int* resourcesCounts);
00055     void selectedAlgorithm(int algorithm);
00056
00058     void updateStillNeededRessources_R();
00059
00061     void reserveResources(int process, int resource, int count);
00062     void releaseResources(int process, int resource, int count, bool notProcessedYet);
00063     void processFinished(int processId);
00064
```

```
00066     void on_button_stop_simulation_clicked();
00067     void on_button_start_simulation_clicked();
00068     void on_button_restart_simulation_clicked();
00069
00070     void on_explanation_Button_explanation_clicked();
00071
00072     void on_explanation_Button_algorithm_clicked();
00073
00074 private:
00075     Ui::MainWindow *ui;
00076     QThread *threadProcessA, *threadProcessB, *threadProcessC, *threadPreemption;
00077     ProcessWorker *workerA, *workerB, *workerC;
00078     PreemptionWorker *preemptionWorker;
00079 };
00080
00081 #endif // MAINWINDOW_H
```

7.35 Objects/processworker.cpp File Reference

```
#include ¨processworker.h¨
```

Include dependency graph for processworker.cpp:

7.36 Objects/processworker.h File Reference

```
#include ¨qdebug.h¨
#include <QObject>
#include <Objects/SystemProcess.h>
#include <Objects/systemresource.h>
#include <QSemaphore>
#include <QThread>
#include <QDebug>
#include <Algorithms/BankiersAlgorithm.h>
#include <Algorithms/NoAvoidanceSimulation.h>
#include <Algorithms/EliminateCircularWait.h>
#include <Algorithms/EliminateHoldAndWait.h>
#include <Algorithms/NoPreemption.h>
```

Include dependency graph for processworker.h: This graph shows which files directly or indirectly include this file:

Classes

- class ProcessWorker

  Class represents the process worker which is responsible for the whole simulation process.

7.37 processworker.h

Go to the documentation of this file.

```
00001 #ifndef PROCESSWORKER_H
00002 #define PROCESSWORKER_H
00003
00004 #include "qdebug.h"
00005 #include <QObject>
00006 #include <Objects/SystemProcess.h>
00007 #include <Objects/systemresource.h>
00008 #include <QSemaphore>
00009 #include <QThread>
00010 #include <QDebug>
00011 #include <Algorithms/BankiersAlgorithm.h>
00012 #include <Algorithms/NoAvoidanceSimulation.h>
```

```
00013 #include <Algorithms/EliminateCircularWait.h>
00014 #include <Algorithms/EliminateHoldAndWait.h>
00015 #include <Algorithms/NoPreemption.h>
00016
00017 //Q_DECLARE_METATYPE(QList<SystemProcess>)
00018
00022 class ProcessWorker : public QObject
00023 {
00024     Q_OBJECT
00025 public:
00032     explicit ProcessWorker(SystemProcess processes, int selectedAlgorithm, QObject *parent = 0);
00033
00039     void updateProcess(int nextResource, int countResource);
00040
00046     void setUpOccupations(int assignedResources_C[3][4], int stillNeededResources_R[3][4]){
00047         for(int i = 0; i < 3; i++){
00048             for(int j = 0; j < 4; j++){
00049                 this->assignedResources_C[i][j] = assignedResources_C[i][j];
00050                 this->stillNeededResources_R[i][j] = stillNeededResources_R[i][j];
00051             }
00052         }
00053     }
00054
00059     void setAlgorithm(int algorithm){
00060         this->selectedAlgorithm = algorithm;
00061     }
00062
00063     //temporary debug method to check variables
00064     void printStillNeeded(){
00065         QDebug dbg(QtDebugMsg);
00066         dbg << "R:" << "\n";
00067         for(int i = 0; i < 3; i++){
00068             for(int j = 0; j < 4; j++){
00069                 dbg << stillNeededResources_R[i][j];
00070             }
00071             dbg << "\n";
00072         }
00073         dbg << "A: " << differenceResources_A[0] << differenceResources_A[1] << differenceResources_A[2] <<
      differenceResources_A[3] << "\n";
00074     }
00075
00076 signals:
00077
00078     void startedAcquire(int processId, int nextResource, int nextCount);
00085     void resourceReserved(int processID, int resource, int count);
00086
00093     void resourceReleased(int processID, int resource, int count, bool notProcessedYet);
00094
00099     void finishedResourceProcessing(int processID);
00100
00101
00102 public slots:
00107     void requestResource();
00108
00109 public:
00110     static QSemaphore *semaphorePrinter;
00111     static QSemaphore *semaphoreCD;
00112     static QSemaphore *semaphorePlotter;
00113     static QSemaphore *semaphoreTapeDrive;
00114     static int differenceResources_A[4];
00115     static int availableResources_E[4];
00116     static int assignedResources_C[3][4];
00117     static int stillNeededResources_R[3][4];
00118
00119 private:
00120     SystemProcess process;
00121     int selectedAlgorithm;
00122
00123 };
00124
00125 #endif // PROCESSWORKER_H
```

7.38 Objects/systemprocess.cpp File Reference

```
#include ¨systemprocess.h¨
#include <algorithm>
#include <QDebug>
#include <QRandomGenerator>
```

Include dependency graph for systemprocess.cpp:

7.39 Objects/systemprocess.h File Reference

```
#include <QList>
#include ¨Objects/systemresource.h¨
#include <QDebug>
#include <random>
#include <QtGlobal>
#include <QVector>
#include <QRandomGenerator>
```

Include dependency graph for systemprocess.h: This graph shows which files directly or indirectly include this file:

Classes

- class SystemProcess

  Class represents the processes which use the resources.

7.40 systemprocess.h

Go to the documentation of this file.

```
00001 #ifndef SYSTEMPROCESS_H
00002 #define SYSTEMPROCESS_H
00003
00004 #include <QList>
00005 #include "Objects/systemresource.h"
00006 #include <QDebug>
00007 #include <random>
00008 #include <QtGlobal>
00009 #include <QVector>
00010 #include <QRandomGenerator>
00011
00012
00016 class SystemProcess
00017 {
00018
00019 private:
00025     QString name;
00026     int processId;
00027     QList<SystemResource> neededResources;
00028     int revokedResourceId;
00029
00030 public:
00034     SystemProcess(){
00035
00036     };
00037
00043     SystemProcess(QString name,int processId);
00044
00052     SystemProcess(QString name,int processId, int maxPrinters, int maxCDs, int maxPlotters, int
      maxTapeDrives);
00053
00058     int getProcessId() const{
00059         return processId;
00060     }
00061
00066     void setProcessId(int processId){
00067         this->processId = processId;
00068     }
00069
00070
00075     QString getName() const{
00076         return name;
00077     }
00078
00083     void setName(QString name){
00084         this->name = name;
00085     }
00086
00087     int getRevokedResourceId() const{
00088         return revokedResourceId;
00089     }
00090
```

```
00091     void setRevokedResourceId(int revokedResourceId){
00092         this->revokedResourceId = revokedResourceId;
00093     }
00094
00099     const QList<SystemResource> getNeededResources() const{
00100         return neededResources;
00101     }
00102
00107     void setNeededResources(QList<SystemResource> neededResources) {
00108         SystemProcess::neededResources = neededResources;
00109     }
00110
00115     void moveNeededResourceToBack(int index){
00116         neededResources.swapItemsAt(index, neededResources.count()-1);
00117     }
00118
00119     void shuffleNeededResources(){
00120         int n = neededResources.size();
00121         QRandomGenerator rng = QRandomGenerator::securelySeeded();
00122
00123         for (int i = n - 1; i > 0; --i) {
00124             int j = rng.bounded(i + 1); // Generate a random index in [0, i]
00125             std::swap(neededResources[i], neededResources[j]); // Swap the elements at indices i and j
00126         }
00127
00128     }
00129
00130     void printNeededResources(){
00131         QDebug dbg(QtDebugMsg);
00132         dbg « "Process " « name « ":" « "\n";
00133         for(int i = 0; i < neededResources.size(); i++){
00134             dbg « neededResources.at(i).getName() « " (" « neededResources.at(i).getCount() « "), ";
00135         }
00136     }
00137 };
00138
00139 #endif // SYSTEMPROCESS_H
```

7.41 Objects/systemresource.cpp File Reference

`#include ¨systemresource.h¨`
Include dependency graph for systemresource.cpp:

7.42 Objects/systemresource.h File Reference

`#include <QString>`
Include dependency graph for systemresource.h: This graph shows which files directly or indirectly include this file:

Classes

- class SystemResource

  Class which represents the resources used by processes.

7.43 systemresource.h

Go to the documentation of this file.

```
00001 #ifndef SYSTEMRESOURCE_H
00002 #define SYSTEMRESOURCE_H
00003
00004 #include <QString>
00005
00009 class SystemResource
00010 {
```

```
00011
00012 private:
00018     QString name;
00019     int resourceId;
00020     int count;
00021
00022 public:
00029     SystemResource(QString name, int resourceId, int count);
00030
00035     int getResourceId() const{
00036         return resourceId;
00037     }
00038
00043     void setResourceId(int resourceId);
00044
00049     const QString &getName() const {
00050         return name;
00051     }
00052
00057     void setName(const QString &name) {
00058         SystemResource::name = name;
00059     }
00060
00065     int getCount() const {
00066         return count;
00067     }
00068
00073     void setCount(int count) {
00074         this->count = count;
00075     }
00076
00081     void decreaseCount(int count){
00082         this->count -= count;
00083     }
00084
00085     bool operator==(const SystemResource& otherResource) const{
00086         return resourceId == otherResource.resourceId and name == otherResource.name;
00087     }
00088 };
00089
00090 #endif // SYSTEMRESOURCE_H
```

Index