5103 Project 2

Generated by Doxygen 1.7.6.1

Wed Mar 28 2012 13:26:03

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

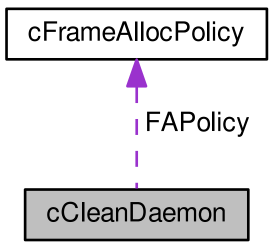
Class Documentation

4.1 cCleanDaemon Class Reference

Cleaning Daemon for system.

`#include <cleaningDaemon.h>`

Collaboration diagram for cCleanDaemon:



Public Member Functions

- **cCleanDaemon** (cFrameAllocPolicy &_FA)
- uint32_t checkClean ()

    Check if pages need to be cleaned.

Private Attributes

- cFrameAllocPolicy & **FAPolicy**
- uint32_t **min_thresh**
- uint32_t **clean_amnt**

4.1.1 Member Function Documentation

4.1.1.1 uint32_t cCleanDaemon::checkClean ( )

The VMM core calls this and the daemon checks with the FA module to see how many frames are available. If it is below its threshold it returns how many should be cleaned. This value is passed to the PR module which makes the policy decisions on which to replace.

Referenced by cVMM::start().

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

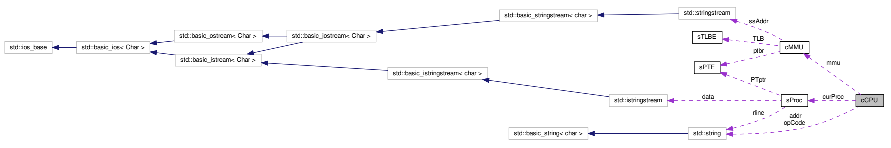
- include/Policy/cleaningDaemon.h
- src/Policy/cleaningDaemon.cpp

4.2 cCPU Class Reference

Very Simple CPU.

`#include <cpu.h>`

Collaboration diagram for cCPU:


std::ios_base
std::basic_ios< Char >
std::basic_ostream< Char >
std::basic_istream< Char >
std::basic_iostream< Char >
std::basic_stringstream< char >
std::basic_istringstream< char >
std::stringstream
std::istringstream
std::basic_string< char >
std::string
ssAddr
sTLBE
TLB
ptbr
cMMU
sPTE
mmu
PTptr
data
sProc
curProc
cCPU
rline
addr
opCode

Public Member Functions

- void addVC (int ∗VC)

  Give the cpu a reference to the core's VC.
- void switchProc (sProc ∗)

  Context switch for this process.
- uint32_t getFaultPage ()

  Get the page number that faulted.
- uint8_t run ()

  Start executing the current process.

Private Member Functions

- void incVC (int amnt)

  Increment the VC and print its value to trace file.

Private Attributes

- sProc ∗ **curProc**
- cMMU **mmu**
- string **opCode**
- string **addr**
- int **instr_time**
- int **cs_time**
- int **quanta**
- int ∗ **VC**

4.2.1 Detailed Description

This cpu is only used to abstract the actual reading/execution of the programs and the use of the MMU. This cpu is not meant to mimic the functionality of the cpu from the first project.

4.2.2 Member Function Documentation

4.2.2.1 cCPU::addVC (int ∗ *VC*)

The VMM core calls this to provide a reference to the VC so the cpu can update it appropriately. This makes it easier than having some back and forth time management between the two systems.

Referenced by cVMM::cVMM().

4.2.2.2 cCPU::getFaultPage ( )

On a page fault the VMM core will queury here to find out which page needs to be brough in. This function simply calls the mmu for this information.

Referenced by cVMM::start().

4.2.2.3 cCPU::run ( )

Once a process has been switched onto the cpu this is called to start execution. The process executes for the quanta specified in the .ini file/s. If the process page faults before this quanta it returns with a special staus code.

Referenced by cVMM::start().

4.2.2.4 cCPU::switchProc (sProc ∗ *newProc*)

There are a couple scenarios to consider here:

- parameter = NULL - Do Nothing
- curProc=NULL Switches process Flushes TLB (Not really necessary but make sure) Set PTBR in MMU
- curProc!=NULL Increment CS count for curProc Flush TLB Set PTBR for new proc Increment the VC by the cs_switch time (default 5)

Referenced by cVMM::start().

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

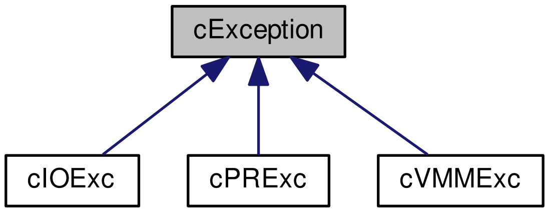
- include/cpu.h
- src/cpu.cpp

4.3 cException Class Reference

Generic exception class.

`#include <exceptions.h>`

Inheritance diagram for cException:



Public Member Functions

- void **setErrorStr** (const string &msg)
- void setFatality (bool f)

    Is this a fatal error?
- string & **getErrorStr** ()
- bool **isFatal** ()

Private Attributes

- string **errMsg**
- bool **fatal**

4.3.1 Detailed Description

These exceptions are caught in main and their error messages are printed out.

4.3.2 Member Function Documentation

4.3.2.1 void cException::setFatality (bool *f*) `[inline]`

After this is set you must press up, up, down, down, left, right, left, right, B, A. Hopefully then the program will fix itself. No really, you should end the program.

Referenced by cPRFifo::clearPages(), cPRLruApprox::clearPages(), and cPRLru::clear-Pages().

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

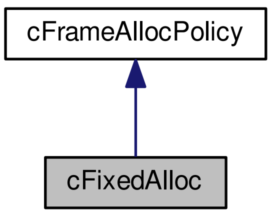
- include/exceptions.h

4.4 cFixedAlloc Class Reference

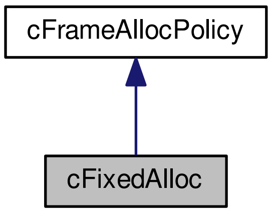
A class implementing a simple fcfs style frame allocation.

`#include <frameAlloc.h>`

Inheritance diagram for cFixedAlloc:



Collaboration diagram for cFixedAlloc:



Public Member Functions

- void regProcs (vector< sProc ∗ > &procs)

  Give the policy module information about processes.
- bool **checkAvailable** (uint32_t frame, bool pinnedTaken=true)
- pair< bool, uint32_t > **getFrame** ()
- pair< bool, uint32_t > **getFrame** (sProc ∗)
- bool **getFrame** (uint32_t frame)
- void returnFrame (uint32_t frame)

  System is returning a page.
- uint32_t **checkOpen** (bool)
- bool pin (uint32_t frame)

  Pin a frame.
- bool **unpin** (uint32_t frame)

Private Member Functions

- uint32_t **findFirstOf** (bool check, dynamic_bitset<> &bits)
- void **printFrames** ()

Private Attributes

- int **allocSize**
- dynamic_bitset **frames**
- dynamic_bitset **pinned**
- uint32_t **numFrames**
- uint32_t **openFrames**
- bool **_printF**
- bool **_printF_on_pin**
- FILE ∗ **printLoc**

4.4.1 Detailed Description

The namae "fixed alloc" is a bit of a misnomer because originally we though that each process would only be given a fixed number of frames. With the global demand paging policy this changed to simply being a class for managing which frames were open and whenever someone requests a page they are given one if it is available, regardless of how many they have gotten before.

4.4.2 Member Function Documentation

4.4.2.1 **cFixedAlloc::pin (** uint32_t *frame* **)** `[virtual]`

A pinned frame cannot be given to anyone else. Usually this does not matter because the page is reserved first and therefore the frame allocator won't give it out again. There are scenarios where on a page fault they PR module may decide to replace a page which is still coming into memory. In this case the PR module would notice from this class that the frame is pinned so it can't spill it.

This scenario happens mostly in low memory situations.

Implements cFrameAllocPolicy.

4.4.2.2 void **cFixedAlloc::regProcs (** vector< **sProc** * > & *procs* **)** `[virtual]`

Some frame allocation policies may want information regarding particular processes. This is called after construction and it gives the policy a chance to build any necessary datastructures.

Implements cFrameAllocPolicy.

4.4.2.3 void **cFixedAlloc::returnFrame (** uint32_t *frame* **)** `[virtual]`

This could happen on process termination or invocation of the page cleaning daemon.

Implements cFrameAllocPolicy.

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

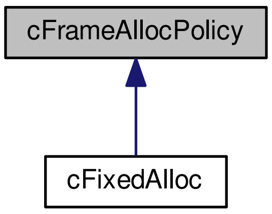
- include/Policy/frameAlloc.h
- src/Policy/frameAlloc.cpp

4.5 cFrameAllocPolicy Class Reference

An abstract class for allocating frames to processes.

```
#include <frameAlloc.h>
```

Inheritance diagram for cFrameAllocPolicy:



Public Member Functions

- void setNumFrames (int frames)

  How many frames does the whole system have.
- virtual void regProcs (vector< sProc ∗ > &procs)=0

  Give the policy module information about processes.
- virtual pair< bool, uint32_t > **getFrame** (sProc ∗)=0
- virtual void returnFrame (uint32_t frame)=0

  System is returning a page.
- virtual uint32_t **checkOpen** (bool)=0
- virtual bool **pin** (uint32_t frame)=0
- virtual bool **unpin** (uint32_t frame)=0

Private Member Functions

- virtual void **printFrames** ()=0

Private Attributes

- uint32_t **numFrames**

4.5.1 Detailed Description

The VMM core uses derived classes of this type to decide how many frames to give a process when it is created.

4.5.2 Member Function Documentation

4.5.2.1 virtual void **cFrameAllocPolicy::regProcs** (vector< **sProc** * > & *procs*) `[pure virtual]`

Some frame allocation policies may want information regarding particular processes. This is called after construction and it gives the policy a chance to build any necessary datastructures.

Implemented in cFixedAlloc.

4.5.2.2 virtual void **cFrameAllocPolicy::returnFrame** (uint32_t *frame*) `[pure virtual]`

This could happen on process termination or invocation of the page cleaning daemon.

Implemented in cFixedAlloc.

Referenced by cPRFifo::clearPages(), cPRLruApprox::clearPages(), cPRLru::clearPages(), cPRFifo::returnFrame(), cPRLruApprox::returnFrame(), and cPRLru::returnFrame().

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

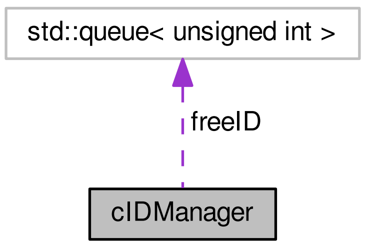
- include/Policy/frameAlloc.h

4.6 cIDManager Class Reference

A class for managing unique IDs.

`#include <id.h>`

Collaboration diagram for cIDManager:



Public Member Functions

- cIDManager (unsigned int startID=0)

    Creates a new ID Manager object.
- unsigned int getID ()

    Reserves a unique ID.
- unsigned int getLowID ()

    Get a low ID.
- void returnID (unsigned int id)

    Returns an ID to the manager.
- unsigned int nextLowID ()

    See what the next low ID would be.
- unsigned int reservedIDs ()

    How many IDs have been given out.

Private Attributes

- queue$<$ unsigned int $>$ freeID

    Queue of returned IDs.
- unsigned int baseID

    To prevent the IDs from dropping below when being returned.
- unsigned int currentID

    Next ID to be given out.
- bool consumeQueue

    This signals that IDs have reached their max and freeIDs should be used.

4.6.1 Constructor & Destructor Documentation

4.6.1.1 cIDManager::cIDManager (unsigned int *startID* = 0)

Default start ID is 0.

4.6.2 Member Function Documentation

4.6.2.1 unsigned int cIDManager::getID ( )

Unique is in the sense that no one else is currently using it but it may have been used previously. are distributed in increasing order until UINT_MAX is reached. After this is reached, IDs are given from the queue of returned IDs. If this queue is empty then an exception is thrown.

Referenced by getLowID().

4.6.2.2 unsigned int cIDManager::getLowID ( )

When generating process PID's we use the regular getID so that process IDs continue to grow. This choice was mainly to prevent confusion when process 1 terminated and the next one to start had pid = 1. For functions which use vectors and need an ID system, it is more efficient to maintain a smaller window to keep the array small. This method provides this by preferring to return IDs from the freeID queue. Therefore, if there is an ID availabe in freeID then the total range of IDs will not grow after this function call.

4.6.2.3 unsigned int cIDManager::nextLowID ( )

There is not longer any purpose for this functino but I left it here for the potential functionality. The intention was to improve performance in the process logger to determine if the next ID would be right after the previous low ID. That way, if we had variable length records we wouldn't have to search from the beginning.

4.6.2.4 unsigned int cIDManager::reservedIDs ( )

Returns the number of IDs which have been reserved

4.6.2.5 void cIDManager::returnID (unsigned int *id*)

If the ID is not equal to the one last given then it is added to a 'free queue'. If it is equal to the last one reserved then the ID counter is simply decremented If this last case happens, it causes cIDManager::getID to stop consuming from the queue and return this newly availabe ID.

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

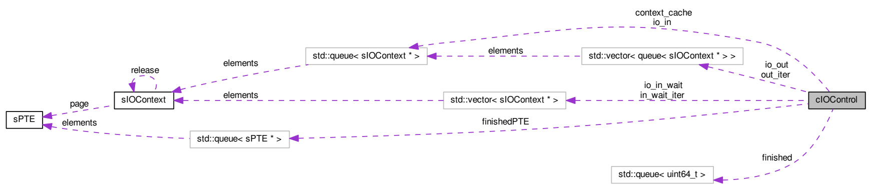
- include/utility/id.h
- src/utility/id.cpp

4.7 cIOControl Class Reference

Class representing an I/O controller.

`#include <io_control.h>`

Collaboration diagram for cIOControl:



Public Member Functions

- cIOControl (int numProcs, int ∗VC)

    Reference virtual counter in core.
- queue< uint64_t > & getFinishedQueue ()

    Get a reference to the finished queue.
- queue< sPTE ∗ > & getFinishedPTEQueue ()

    Get a reference to the finished PTE queue.
- void tick ()

    Count 1 time unit.
- void tick (int times)

    This executes ::tick() N number of times.
- sIOContext ∗ scheduleIO (sProc ∗proc, uint32_t page, eIOType iotype, sIOContext ∗release=NULL)

    Schedule an I/O event.

Private Member Functions

- void removeWait (uint32_t)

    Remove an I/O context from waiting.
- sIOContext ∗ getContext ()

    Get a context struct.
- void returnContext (sIOContext ∗ctx)

    Return a context struct for future use.
- void dumpIOError (sProc ∗)

    Generate and throw IO Error.

Private Attributes

- int **io_req_time**
- int **io_time**
- vector< queue< sIOContext ∗ > > io_out

What is being paged out?
- vector< queue< sIOContext ∗ > >::iterator **out_iter**
- queue< sIOContext ∗ > io_in

  What is the next page in?
- vector< sIOContext ∗ > io_in_wait

  I/O in requests waiting for an io_out to finish.
- vector< sIOContext ∗ >::iterator **in_wait_iter**
- queue< sIOContext ∗ > context_cache

  Cache of previously allocated contexts.
- queue< uint64_t > finished

  Queue that holds process ids that have completed and I/O.
- queue< sPTE ∗ > finishedPTE

  Holds the PTE of finished I/O in's.
- int ∗ **VC**

4.7.1 Member Function Documentation

4.7.1.1 cIOControl::dumpIOError (sProc ∗ *proc*) `[private]`

This does the work of generating an IO excetion.It was originally intended to work broadly for many different scenarios but the necessity was not there. It works but it produces a fairly generic error.

Referenced by scheduleIO().

4.7.1.2 cIOControl::getContext ( ) `[private]`

If there is a context struct cached from previous use then it is returned. Otherwise a new one is allocated.

Referenced by scheduleIO().

4.7.1.3 cIOControl::getFinishedPTEQueue ( ) `[inline]`

This is called by the core to get a reference to this specific queue to be used in processing finished I/O. The purpose for this queue was that the VMM core needed the frame of the completed I/O so it could be unpinned. Given the PTE in this queue it makes page data in the finished queue redundant.

Referenced by cVMM::start().

4.7.1.4 cIOControl::getFinishedQueue ( ) `[inline]`

This is called by the core to get a refernce to this specific queue so it can be used in processing finished I/O. This queue hold 64-bit integers representing the pid and page of for the completed I/O.

Referenced by cVMM::start().

4.7.1.5 cIOControl::removeWait (uint32_t *pid*) `[private]`

When an I/O out completes, if its release context property is not null then the controller will try to release the specified context from the waiting vector.

Referenced by tick().

4.7.1.6 cIOControl::returnContext (sIOContext ∗ *ctx*) `[private]`

The struct is not deallocated. It is put into a cache queue.

Referenced by tick().

4.7.1.7 cIOControl::scheduleIO (sProc ∗ *proc,* uint32_t *page,* eIOType *iotype,* sIOContext ∗ *release =* `NULL`)

Schedule I/O.

This is used to schedule some I/O. The type is specified by the eIOType parameter. If the type is eIOType::IO_IN_WAIT it will be put in a vector (indexed by pid) waiting to be released. If the type is eIOType::IO_OUT and the sIOContext∗ != NULL then after the I/O completes it will relese the I/O pointed to by this context.

Referenced by cVMM::pageIn(), and cVMM::pageOut().

4.7.1.8 cIOControl::tick ( )

Thiscounts 1 virtual time unite and modifies the remaining time on pending I/O correspondingly.

Referenced by scheduleIO(), cVMM::start(), tick(), and cVMM::tickController().

4.7.1.9 cIOControl::tick (int *times*)

This is useful in recording virtual time with events that take chunks of time such as the context switching of a process.

Parameters

int	Number of ticks to execute.

4.7.2 Member Data Documentation

4.7.2.1 queue<sIOContext∗> cIOControl::context_cache `[private]`

This is simply a performance boost to avoid constantly allocating/deallocating context structs.

Referenced by getContext(), and returnContext().

4.7.2.2 queue< uint64_t > cIOControl::finished [private]

This is specifically used for completed page ins. After calling tick, any completed processes are placed in here and the VMM core can read it and unblock.

I made it a 64-bit data type so that I didn't need to mess with allocating and managing some other structure. The first 32-bits represent the pid of the process whose I/O completed. The last 32-bits is the page that was brought in for the process. From this the core can get the frame and unpin it.

Referenced by tick().

4.7.2.3 queue< sIOContext∗ > cIOControl::io_in [private]

This keeps track of the next page to complete its I/O.

Referenced by removeWait(), scheduleIO(), and tick().

4.7.2.4 vector< queue< sIOContext∗ > > cIOControl::io_out [private]

This vector is indexed by pid and it holds a queue of the pages of the respective process which are being paged out. This is so that in the case that a process has a page spilled by some other process' page fault or the cleaning daemon, it is prevented from recovering the page until it is successfully written out.

Referenced by cIOControl(), scheduleIO(), and tick().

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

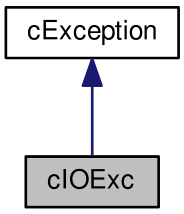
- include/io_control.h
- src/io_control.cpp

4.8 cIOExc Class Reference

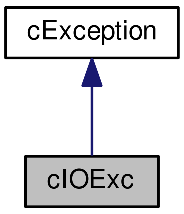
Class handling exceptions in the I/O system.

`#include <exceptions.h>`

Inheritance diagram for cIOExc:



Collaboration diagram for cIOExc:



Public Member Functions

- void **setTrace** (const string &trace)
- string & **getTrace** ()

Private Attributes

- string **IO_Data_Trace**

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

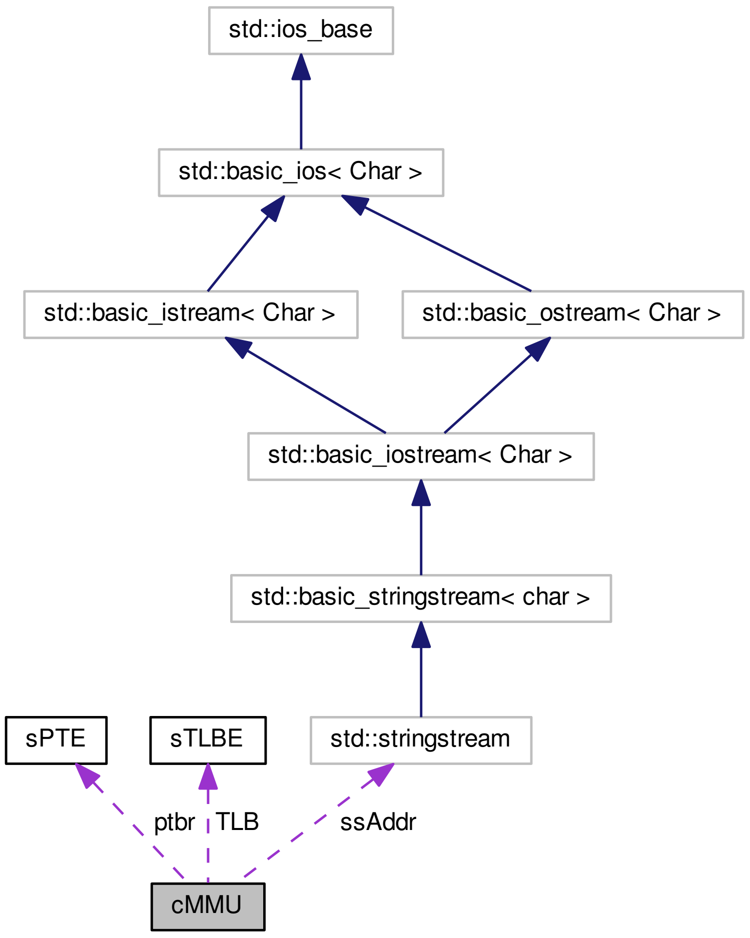
- include/exceptions.h

4.9 cMMU Class Reference

Class representing an MMU in the CPU.

```
#include <mmu.h>
```

Collaboration diagram for cMMU:


std::ios_base
std::basic_ios< Char >
std::basic_istream< Char >
std::basic_ostream< Char >
std::basic_iostream< Char >
std::basic_stringstream< char >
sPTE
sTLBE
std::stringstream
ptbr
TLB
ssAddr
cMMU

Public Member Functions

- void setPTBR (sPTE ∗_ptbr)

  Set the Page Table Base Register.

- void addVC (int ∗VC)

    Add a refernece to the main VCC for timekeeping purposes.
- void flushTLB (bool sync=true)

    Flush the tlb and optionally sync values back to page table.
- void syncTLB ()

    Sync TLB entries without flushing them.
- eMMUstate checkStatus ()

    What is the MMU status after the last translation.
- uint32_t getFaultPage ()

    What was the VPN that caused the fault.
- uint32_t getAddr (string &sVA, bool isWrite)

    Translate virtual to physical address.

Private Member Functions

- void addTLB (uint32_t, uint32_t, bool)

    Add an entry to the TLB.

Private Attributes

- sTLBE ∗ **TLB**
- uint16_t **tlbSize**
- uint32_t **pageSize**
- uint32_t **off_bits**
- bool **hexAddr**
- stringstream **ssAddr**
- uint32_t replaceIndex

    For replacing old tlb entries we are simply treating it as a ring buffer.
- int ∗ **VC**
- sPTE ∗ **ptbr**
- int **tlb_hits**
- int **tlb_misses**
- eMMUstate **mmu_status**
- uint32_t **faultPage**

4.9.1 Member Function Documentation

4.9.1.1 cMMU::checkStatus ( ) `[inline]`

Since the getAddr function returns the translated address. With the exception of doing some bit packing we can't return the status in the same variable.

Referenced by cCPU::run().

4.9.1.2 cMMU::flushTLB (bool *sync* = `true`)

On context switch this is called to clear all the entries by simply setting their valid bit to false. If the sync flag is set and a page is dirty/referenced then this information is written back to the main page table.

This syncing is usually not necessary since the cpu calls sync on a page fault or quanta terminnation so that the table is up-to-date for the PR module to make decisions.

Referenced by cCPU::switchProc().

4.9.1.3 uint32_t cMMU::getAddr (string & *sVA*, bool *isWrite*)

This method checks the tlb and page table simultaneously and returns the appropriate address. Whoever receives this should first check the mmu status to make sure it is valid.

Parameters

bool	write If set, the mmu will mark this page as dirty.

Referenced by cCPU::run().

4.9.1.4 cMMU::getFaultPage ( ) `[inline]`

The cpu calls this to send back to the VMM core so it can handle it accordingly.

Referenced by cCPU::getFaultPage().

4.9.1.5 cMMU::setPTBR (sPTE * *_ptbr*) `[inline]`

This needs to be reset on context switch

Referenced by cCPU::switchProc().

4.9.1.6 cMMU::syncTLB ( )

This is the same as ::fulshTLB with the exception of not flushing/ invalidating the table entries. This is useful when an execution quanta finishes so you want to sync the entries but not flush them in-case it is the only process left.

Referenced by cCPU::run().

4.9.2 Member Data Documentation

4.9.2.1 uint32_t cMMU::replaceIndex `[private]`

This is the index where new entries are being placed

Referenced by addTLB(), and flushTLB().

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

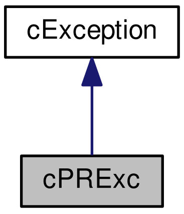
- include/mmu.h
- src/mmu.cpp

4.10 cPRExc Class Reference

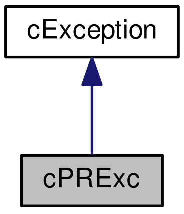
Exception specific to the PR module.

`#include <exceptions.h>`

Inheritance diagram for cPRExc:



Collaboration diagram for cPRExc:



Public Member Functions

- void **setName** (const string &_name)

Private Attributes

- string **name**

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

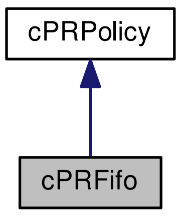
- include/exceptions.h

4.11 cPRFifo Class Reference

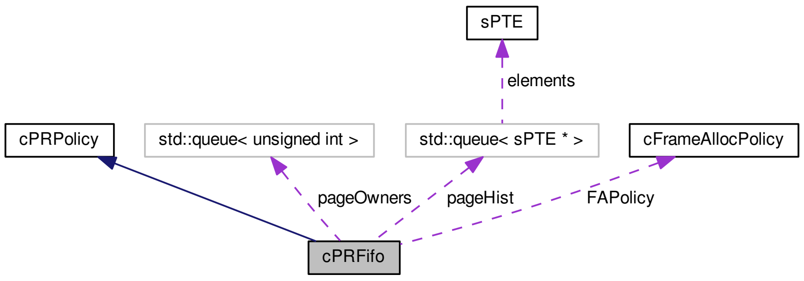
FIFO PR Policy.

`#include <pr_fifo.h>`

Inheritance diagram for cPRFifo:



Collaboration diagram for cPRFifo:



Public Member Functions

- **cPRFifo** (cFrameAllocPolicy &_FAPolicy)
- const char ∗ name ()

  Name of PR module.
- ePRStatus resolvePageFault (sProc ∗proc, uint32_t page)

  Get a page frame for this process.
- void finishedQuanta (sProc ∗proc)

  This is called after an execution quanta finishes.
- void finishedIO (sProc ∗proc, sPTE ∗page)

  This is called after an I/O (input) completes.
- bool clearPages (int numPages)

  Remove this many pages from memory.
- void unpinFrame (uint32_t frame)

  This is typically called after ::finishedIO so that the PR module unpins the frame in the frame allocation module.
- void returnFrame (uint32_t frame)

  Return a frame to the system.

Private Attributes

- queue< sPTE ∗ > **pageHist**
- queue< unsigned int > **pageOwners**
- cFrameAllocPolicy & **FAPolicy**
- uint32_t **PTSize**

4.11.1 Member Function Documentation

4.11.1.1 bool **cPRFifo::clearPages** (int *numPages*) `[virtual]`

If the cleaning daemon decides that pages need to be cleaned it will determine how many from its policy and call the PR module to clean that many. This method keeps the PR policy separate from the decision on how many pages to clean and when.

Parameters

int	Number of pages to clear.

Implements cPRPolicy.

4.11.1.2 void **cPRFifo::finishedIO** (**sProc** ∗ , **sPTE** ∗) `[virtual]`

This is another notification point for the PR module to update its data structs.

Implements cPRPolicy.

4.11.1.3 void **cPRFifo::finishedQuanta** (**sProc** ∗) `[virtual]`

This gives the PR module a chance to update its internal data-structures using the reference bits or any other data available from the process struct.

Implements cPRPolicy.

4.11.1.4 const char∗ **cPRFifo::name** ( ) `[inline, virtual]`

This is only used for logging purposes so that the PR module can be easily identified.

Implements cPRPolicy.

4.11.1.5 **ePRStatus cPRFifo::resolvePageFault** (**sProc** ∗ *proc,* uint32_t *page*) `[virtual]`

If a free one is available from the frame allocator it will be used. Otherwise, a page will have to be spilled.

Parameters

sProc∗	A pointer to the process that the frame is being requested for.
uint32_t	page The page for the given process that needs to be fetched.

Implements cPRPolicy.

4.11.1.6 void **cPRFifo::returnFrame** (uint32_t *frame*) `[virtual]`

The VMM core calls this when a process termintates to free up any of its used frames. This call could be made directly to the frame allocator but using the PR module as a middle man gives it a chance to update as needed.

Implements cPRPolicy.

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

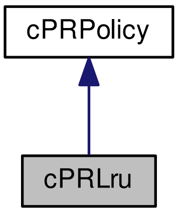
- include/Policy/pr_fifo.h
- src/Policy/pr_fifo.cpp

4.12 cPRLru Class Reference

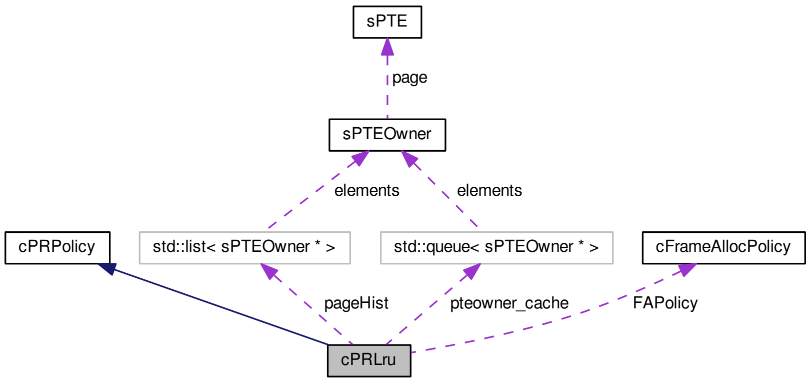
Pure LRU PR Policy.

`#include <pr_lru.h>`

Inheritance diagram for cPRLru:



Collaboration diagram for cPRLru:



Public Member Functions

- **cPRLru** (cFrameAllocPolicy &_FAPolicy)
- const char ∗ name ()

  Name of PR module.
- ePRStatus resolvePageFault (sProc ∗proc, uint32_t page)

  If no frames are free, spill the frame that has the lowest timestamp, or was "least recently used.
- void finishedQuanta (sProc ∗proc)

  Since all the times are set in software, this method does nothing.
- void finishedIO (sProc ∗proc, sPTE ∗page)

  Add the PTE to the pageHist to keep track of which frames have been given out.
- bool clearPages (int numPages)

  Sorts pageHist by timestamp (increasing order) then removes numPages from the front of the list.
- void unpinFrame (uint32_t frame)

  This is typically called after ::finishedIO so that the PR module unpins the frame in the frame allocation module.
- void printTimestamps ()

  Used for debugging purposes.
- void returnFrame (uint32_t frame)

  Return a frame to the system.

Private Member Functions

- sPTEOwner ∗ getPTEOwner ()

  Get a struct for holding both the PTE and Owner.
- void returnPTEOwner (sPTEOwner ∗pteOwner)

  Add a PTEOwner to the pteowner_cache to use later.

Private Attributes

- list< sPTEOwner ∗ > **pageHist**
- queue< sPTEOwner ∗ > **pteowner_cache**
- cFrameAllocPolicy & **FAPolicy**
- uint32_t **PTSize**

4.12.1 Member Function Documentation

4.12.1.1 sPTEOwner ∗ cPRLru::getPTEOwner ( ) `[private]`

If the pteowner_cache has an entry, re-use it. Otherwise malloc a new one. This helps with heap performance.

Referenced by finishedIO().

4.12.1.2 const char∗ cPRLru::name ( ) `[inline, virtual]`

This is only used for logging purposes so that the PR module can be easily identified.

Implements cPRPolicy.

4.12.1.3 void cPRLru::printTimestamps ( )

Prints all of the timestamps in the pageHist list.

4.12.1.4 ePRStatus cPRLru::resolvePageFault (sProc ∗ *proc*, uint32_t *page*) `[virtual]`

"

Implements cPRPolicy.

4.12.1.5 void cPRLru::returnFrame (uint32_t *frame*) `[virtual]`

The VMM core calls this when a process termintates to free up any of its used frames. This call could be made directly to the frame allocator but using the PR module as a middle man gives it a chance to update as needed.

Implements cPRPolicy.

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

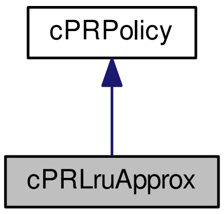
- include/Policy/pr_lru.h
- src/Policy/pr_lru.cpp

4.13 cPRLruApprox Class Reference

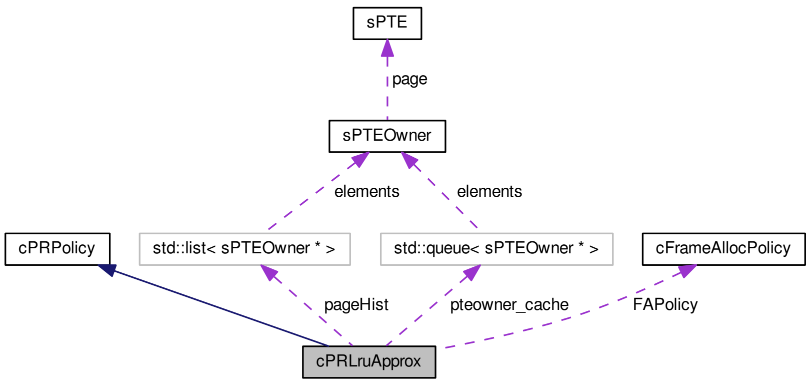
LRU approximate PR Policy.

`#include <pr_lruApprox.h>`

Inheritance diagram for cPRLruApprox:



Collaboration diagram for cPRLruApprox:



Public Member Functions

- **cPRLruApprox** (cFrameAllocPolicy &_FAPolicy)
- const char ∗ name ()

  Name of PR module.
- ePRStatus resolvePageFault (sProc ∗proc, uint32_t page)

  If no frames are free, spill the frame that has the lowest shifted time.
- void finishedQuanta (sProc ∗proc)

  Update the approx time field in software.
- void finishedIO (sProc ∗proc, sPTE ∗page)

  I/O is finished so add the PTE to the pageHist so we know what frames are in use.
- bool clearPages (int numPages)

  Clears pages by sorting the pageHist by the time field (increasing).
- void unpinFrame (uint32_t frame)

  This is typically called after ::finishedIO so that the PR module unpins the frame in the frame allocation module.
- void returnFrame (uint32_t frame)

  Return a frame to the system.

Private Member Functions

- void updateTime ()

  Updates the time field by first shifting the time field to the right and then adding a 1 in the far left position if the ref bit is set.
- sPTEOwner ∗ getPTEOwner ()

  Get a struct for holding both the PTE and Owner.
- void returnPTEOwner (sPTEOwner ∗pteOwner)

  Add a PTEOwner to the pteowner_cache to use later.

Private Attributes

- list< sPTEOwner ∗ > **pageHist**
- queue< sPTEOwner ∗ > **pteowner_cache**
- cFrameAllocPolicy & **FAPolicy**
- uint32_t **PTSize**

4.13.1 Member Function Documentation

4.13.1.1 bool **cPRLruApprox::clearPages** (int *numPages*) `[virtual]`

Then removes numPages from the front of pageHist.

Implements cPRPolicy.

4.13.1.2 sPTEOwner ∗ cPRLruApprox::getPTEOwner ( ) [private]

If the pteowner_cache has an entry, re-use it. Otherwise malloc a new one. This helps with heap performance.

Referenced by finishedIO().

4.13.1.3 const char∗ cPRLruApprox::name ( ) [inline, virtual]

This is only used for logging purposes so that the PR module can be easily identified.

Implements cPRPolicy.

4.13.1.4 ePRStatus cPRLruApprox::resolvePageFault (sProc ∗ *proc*, uint32_t *page*) [virtual]

This aging algorithm simulates LRU.

Implements cPRPolicy.

4.13.1.5 void cPRLruApprox::returnFrame (uint32_t *frame*) [virtual]

The VMM core calls this when a process termintates to free up any of its used frames. This call could be made directly to the frame allocator but using the PR module as a middle man gives it a chance to update as needed.

Implements cPRPolicy.

4.13.1.6 cPRLruApprox::updateTime ( ) [private]

This function is called on a page fault or after a quanta is finished.

Referenced by finishedQuanta(), and resolvePageFault().

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

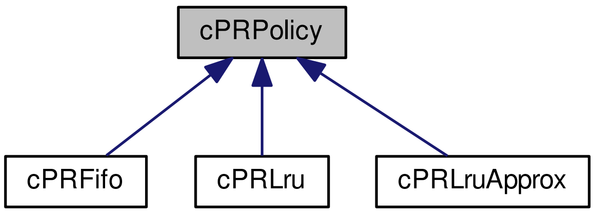
- include/Policy/pr_lruApprox.h
- src/Policy/pr_lruApprox.cpp

4.14 cPRPolicy Class Reference

Abstract Page Replacement Policy.

`#include <pageReplace.h>`

Inheritance diagram for cPRPolicy:



Public Member Functions

- **cPRPolicy** (cFrameAllocPolicy &)
- virtual const char ∗ name ()=0

    Name of PR module.
- virtual ePRStatus resolvePageFault (sProc ∗proc, uint32_t page)=0

    Get a page frame for this process.
- ePRStatus **resolveCircularPF** (sProc ∗proc, uint32_t page)
- virtual void finishedQuanta (sProc ∗)=0

    This is called after an execution quanta finishes.
- virtual void finishedIO (sProc ∗, sPTE ∗)=0

    This is called after an I/O (input) completes.
- virtual bool clearPages (int numPages)=0

    Remove this many pages from memory.
- virtual void unpinFrame (uint32_t frame)=0

    This is typically called after ::finishedIO so that the PR module unpins the frame in the frame allocation module.
- virtual void returnFrame (uint32_t frame)=0

    Return this frame to the system.

4.14.1 Member Function Documentation

4.14.1.1 cPRPolicy::clearPages (int *numPages*) `[pure virtual]`

If the cleaning daemon decides that pages need to be cleaned it will determine how many from its policy and call the PR module to clean that many. This method keeps the PR policy separate from the decision on how many pages to clean and when.

Parameters

int	Number of pages to clear.

Implemented in cPRLru, cPRLruApprox, and cPRFifo.

Referenced by cVMM::start().

4.14.1.2 cPRPolicy::finishedIO (sProc ∗ , sPTE ∗) `[pure virtual]`

This is another notification point for the PR module to update its data structs.

Implemented in cPRLru, cPRLruApprox, and cPRFifo.

Referenced by cVMM::start().

4.14.1.3 cPRPolicy::finishedQuanta (sProc ∗) `[pure virtual]`

This gives the PR module a chance to update its internal data-structures using the reference bits or any other data available from the process struct.

Implemented in cPRLru, cPRLruApprox, and cPRFifo.

Referenced by cVMM::start().

4.14.1.4 virtual const char∗ cPRPolicy::name ( ) `[pure virtual]`

This is only used for logging purposes so that the PR module can be easily identified.

Implemented in cPRLruApprox, cPRLru, and cPRFifo.

Referenced by cVMM::printResults().

4.14.1.5 virtual ePRStatus cPRPolicy::resolvePageFault (sProc ∗ *proc,* uint32_t *page*) `[pure virtual]`

If a free one is available from the frame allocator it will be used. Otherwise, a page will have to be spilled.

Parameters

sProc∗	A pointer to the process that the frame is being requested for.
uint32_t	page The page for the given process that needs to be fetched.

Implemented in cPRLru, cPRLruApprox, and cPRFifo.

Referenced by cVMM::start().

4.14.1.6 cPRPolicy::returnFrame (uint32_t *frame*) `[pure virtual]`

This is called when a process exits and its frames are being returned to the system. This is usually a wrapper to the FA policy return frame function but it gives the PR module a chance to update its datastructures if necessary.

Since we have made the assumption that this is only called on process exit, it is not necessary to do any I/O on the page.

Parameters

uint32_t	Number of the frame to clear/return.

Implemented in cPRLru, cPRLruApprox, and cPRFifo.

Referenced by cVMM::cleanupProcess().

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

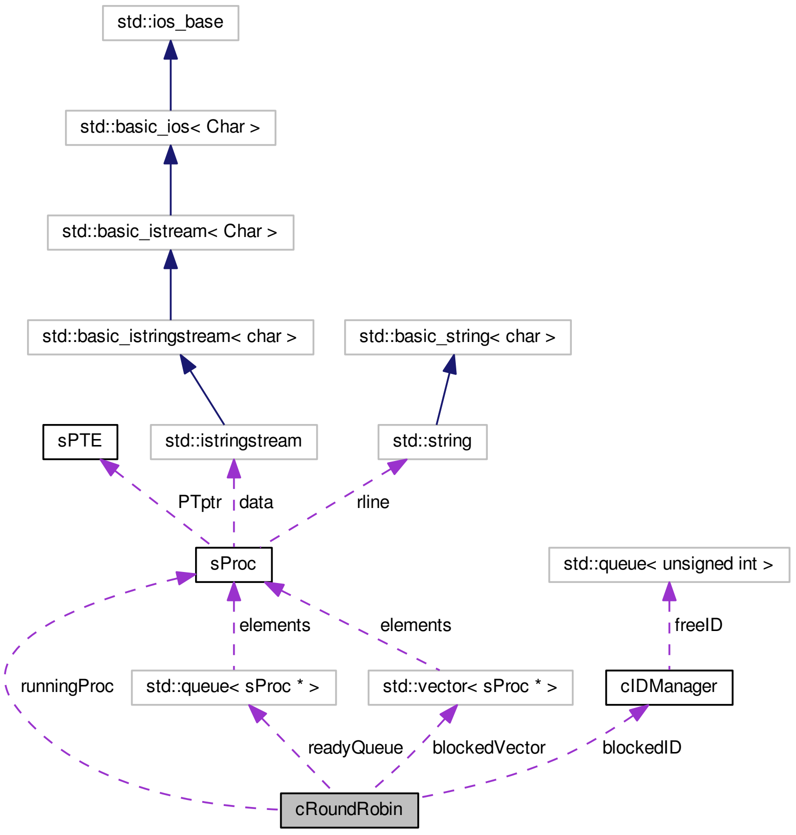
- include/Policy/pageReplace.h
- src/Policy/pageReplace.cpp

4.15 cRoundRobin Class Reference

Round Robin Scheduler.

`#include <round_robin.h>`

Collaboration diagram for cRoundRobin:



Public Member Functions

- void **initProcScheduleInfo** (sProc ∗proc)
- void **addProcesses** (vector< sProc ∗ > &procs)
- void **setBlocked** (sProc ∗)
- void **unblockProcess** (sProc ∗)
- void **removeProcess** (sProc ∗)
- sProc ∗ **getNextToRun** ()
- int **numProcesses** ()

Private Attributes

- queue< sProc ∗ > **readyQueue**

- vector< sProc ∗ > **blockedVector**
- int **totalBlocked**
- int **clockTicksUsed**
- sProc ∗ **runningProc**
- cIDManager **blockedID**

4.15.1 Detailed Description

Nothing too much different here from project1 and since it isn't a core piece to this project it won't be documented in detail.

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

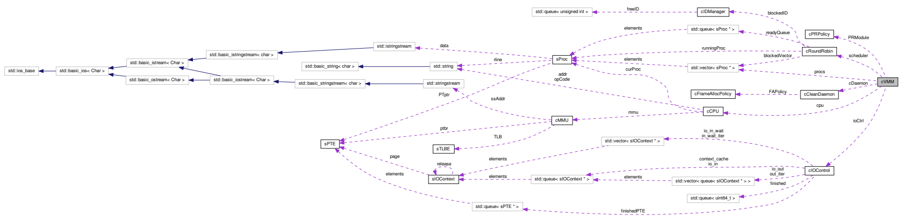
- include/round_robin.h
- src/round_robin.cpp

4.16 cVMM Class Reference

Virtual Memory Manager core class.

`#include <vmm_core.h>`

Collaboration diagram for cVMM:



Public Member Functions

- cVMM (vector< sProc ∗ > &_procs, cPRPolicy &_PRM, cCleanDaemon &_c-Daemon)

  Virtual Memory Manager Constructor.
- ∼cVMM ()

  Virtual Memory Manager Destructor.
- sIOContext ∗ pageOut (sProc ∗, uint32_t, sIOContext ∗ctx=NULL)

  Wrapper for the I/O controller (output scheduling)
- sIOContext ∗ pageIn (sProc ∗, uint32_t, eIOType, sIOContext ∗ctx=NULL)

  Wrapper for I/O controller (input scheduling)
- void tickController (int times)

Tick the I/O controller n number of times.
- sProc $*$ getProcess (unsigned int id)

  Used to get a reference to the particular process struct.
- int start ()

  Start running the processes.

Private Member Functions

- void initProcesses ()

  Initialize process data specific to the VMM.
- void cleanupProcess (sProc $*$proc)

  Cleanup process data and state.
- void clearCircChecks ()

  Set the circular fault check flag to false for each proc.
- void printResults ()

  Gather and print results to log file.

Private Attributes

- uint32_t numFrames

  Global memory frames.
- uint32_t PS

  Page size in bytes.
- uint32_t PT_Size

  Page table size (# of entries)
- vector$<$ sProc $*$ $>$ & procs

  All processes.
- cRoundRobin scheduler

  Process scheduler.
- int currentProc

  PID of current executing process.
- cCPU cpu

  Executing CPU.
- int VC

  Virtual Counter.
- cIOControl $*$ ioCtrl

  I/O Controller.
- uint32_t **pageInCount**
- uint32_t **pageOutCount**
- cPRPolicy & PRModule

  Page Replacement Module.
- cCleanDaemon & cDaemon

  Cleaning Daemon.

4.16.1 Constructor & Destructor Documentation

4.16.1.1 cVMM::cVMM (vector$<$ sProc $*$ $>$ & *_procs,* cPRPolicy & *_PRM,* cCleanDaemon & *_cDaemon*)

Initializes the Virtual Memory Manager class which is the core of the project.

Parameters

vector<s-Proc>&*	A vector contining the processes loaded in main.cpp. The page tables and other data are initialized during VMM initialization.
cPRPolicy&	A reference to a derived class of cPRPolicy (fifo, pure lru, lru apporx).
cClean-Daemon&	A reference to the cleaning daemon. This is passed in because it requires a reference to the frame allocation policy which is only available in main.cpp

4.16.1.2 cVMM::∼cVMM ( )

Cleans up the VMM.

4.16.2 Member Function Documentation

4.16.2.1 cVMM::cleanupProcess (sProc $*$ *proc*) `[private]`

This function returns all occupied frames back to the system. After this it frees the memory used for the page table.

All other data is left in tact and it is not removed from the process vector because upon termination we need to gather statistics for each process for the results trace.

See also

cPRPolicy::returnFrame

Referenced by start().

4.16.2.2 cVMM::clearCircChecks ( ) `[private]`

This addition ot the cpu is more of a safety net but as long as it is being used it needs to work correctly. This funtion is mainly called after the cleaning daemon clears pages so that the cpu doesn't think there is a circular fault when it was really just the cleaning daemon.

Referenced by start().

4.16.2.3 cVMM::getProcess (unsigned int *id*)

In some places like the PR module or the io_controller there is sometimes a need to get a reference to the process when all you have is a pid.

Referenced by cPRFifo::clearPages(), cPRLruApprox::clearPages(), cPRLru::clearPages(), cPRFifo::resolvePageFault(), cPRLru::resolvePageFault(), and cPRLruApprox::resolvePageFault().

4.16.2.4 cVMM::initProcesses ( ) [private]

The main purpose of this function is to initialize the page table for all processes.

See also

sProc

Referenced by cVMM().

4.16.2.5 cVMM::pageIn (sProc ∗ *proc,* uint32_t *page,* eIOType *iotype,* sIOContext ∗ *ctx* = NULL)

Another wrapper for the I/O controller.

See also

cIOControl::scheduleIO

Referenced by cPRFifo::resolvePageFault(), cPRLruApprox::resolvePageFault(), and cPRLru::resolvePageFault().

4.16.2.6 cVMM::pageOut (sProc ∗ *proc,* uint32_t *page,* sIOContext ∗ *ctx* = NULL)

This is a wrapper for the I/O controller. Calling this abstracts the actual I/O controller from those who will use it as well as providing a point for the VMM to record paging statistics.

See also

cIOControl::scheduleIO

Referenced by cPRFifo::clearPages(), cPRLruApprox::clearPages(), cPRLru::clearPages(), cPRFifo::resolvePageFault(), cPRLru::resolvePageFault(), and cPRLruApprox::resolvePageFault().

4.16.2.7 cVMM::printResults ( ) `[private]`

After all execution is finished, this function gathers per-process and global data and prints it to a log file. This log can be read normally or used by the python script to build graphs (assuming a .conf is present).

Referenced by start().

4.16.2.8 cVMM::start ( )

After the FMM has been initialized with some processes this is called to start execution.

4.16.2.9 cVMM::tickController (int *times*)

This is used for timekeeping. For example, when wwe do a context switch on the cpu it takes 5 units of time but we still need to keep track of the passage of time within other devices which virtually should be running in parallel.

See also

cIOcontrol::tick

Referenced by cCPU::incVC().

4.16.3 Member Data Documentation

4.16.3.1 vector<sProc*>& cVMM::procs `[private]`

Referenced by clearCircChecks(), cVMM(), getProcess(), initProcesses(), print-Results(), and start().

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

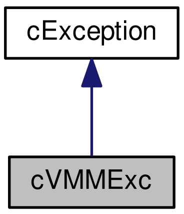
- include/vmm_core.h
- src/vmm_core.cpp

4.17 cVMMExc Class Reference

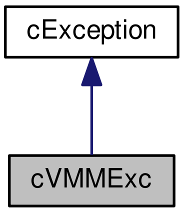
A class handling exceptions in the VMM Core.

`#include <exceptions.h>`

Inheritance diagram for cVMMExc:



Collaboration diagram for cVMMExc:



Public Member Functions

- void **setDump** (const string &dump)
- string & **getDump** ()
- void **setType** (eExType _type)

Private Attributes

- string **dumpInfo**
- eExType **type**

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

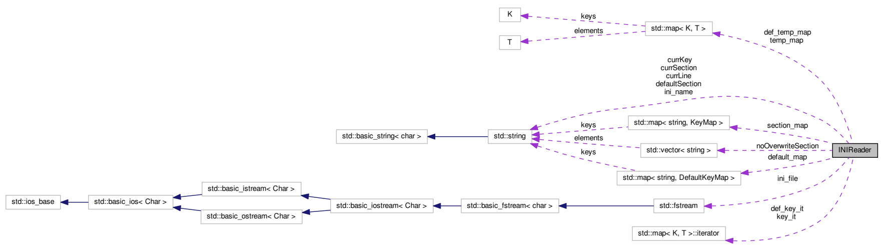
- include/exceptions.h

4.18 INIReader Class Reference

Class for parsing and storing contents of .ini files.

`#include <iniReader.h>`

Collaboration diagram for INIReader:



Public Member Functions

- **INIReader** (string filename)
- bool **load_ini** (string filename, bool auto_parse=true, bool overwrite=true)
- bool **loaded** ()
- bool **exists** (const string &section, const string &key)
- void **addDefault** (const string &section, const string &key, const string &value)
- void **addOverwriteException** (const string &section)
- bool **overWriteOp** (const string &section, const string &key, const string &value)
- template<class T >
  T **extractValue** (const string &section, const string &key)
- string **extractComment** (const string &section, const string &key)

Protected Member Functions

- void **parse_ini** ()
- int **addSection** (char ∗line, bool modCurrSection=true)
- int **addSection** (string &line, bool modCurrSection=true, bool ignoreRules=false)
- int **addKey** (string &line)
- int **addKey** (string &line, string &section)
- string & **getKeyValue** (const string &section, const string &key)
- string & **getKeyComment** (const string &section, const string &key)
- string & **getDefault** (const string &section, const string &key)

Static Protected Member Functions

- static void **strip_white_space** (string &str, const string &TrimChars=" \t\n\r", int TrimDir=0)

Protected Attributes

- fstream **ini_file**
- int **get_pointer**
- int **put_pointer**
- int **lineNumber**
- bool **overwriteMode**
- string **ini_name**
- string **currLine**
- string **currSection**
- string **currKey**
- string **defaultSection**
- SectionMap **section_map**
- SectionMap::iterator **sec_it**
- KeyMap::iterator **key_it**
- KeyMap $*$ **temp_map**
- DefaultMap **default_map**
- DefaultMap::iterator **def_sec_it**
- DefaultKeyMap::iterator **def_key_it**
- DefaultKeyMap $*$ **def_temp_map**
- vector$<$ string $>$ **noOverwriteSection**

4.18.1 Detailed Description

This was taken from another project I was working on with some small addiitions (overriding settings). There is a lot here and because it is not central to this project it won't be documented in detail.

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

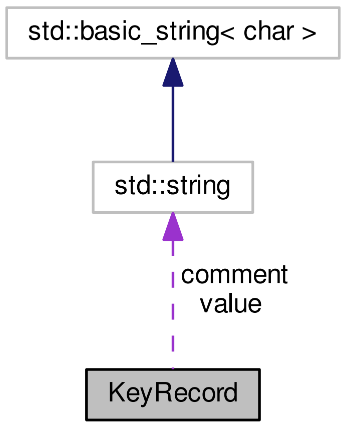
- include/iniReader.h
- src/iniReader.cpp

4.19 KeyRecord Struct Reference

Struct for holding a single settings key and its value.

```
#include <iniReader.h>
```

Collaboration diagram for KeyRecord:



Public Member Functions

- **KeyRecord** (const string value)
- **KeyRecord** (const string, const string, int)
- void **setValue** (const string)
- void **setComment** (string)
- void **setPosPtr** (int)
- string & **getValue** ()
- string & **getComment** ()
- int **getPosPtr** ()
- string **operator()** (const KeyRecord &record)

Protected Attributes

- string **value**
- string **comment**
- int **PosPtr**

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

- include/iniReader.h
- src/iniReader.cpp

4.20 roundRobinInfo Struct Reference

Struct containing process info specific for Round-Robin scheduling.

`#include <round_robin.h>`

Public Attributes

- unsigned int blockedIndex

    Index position in blocked vector.

The documentation for this struct was generated from the following file:

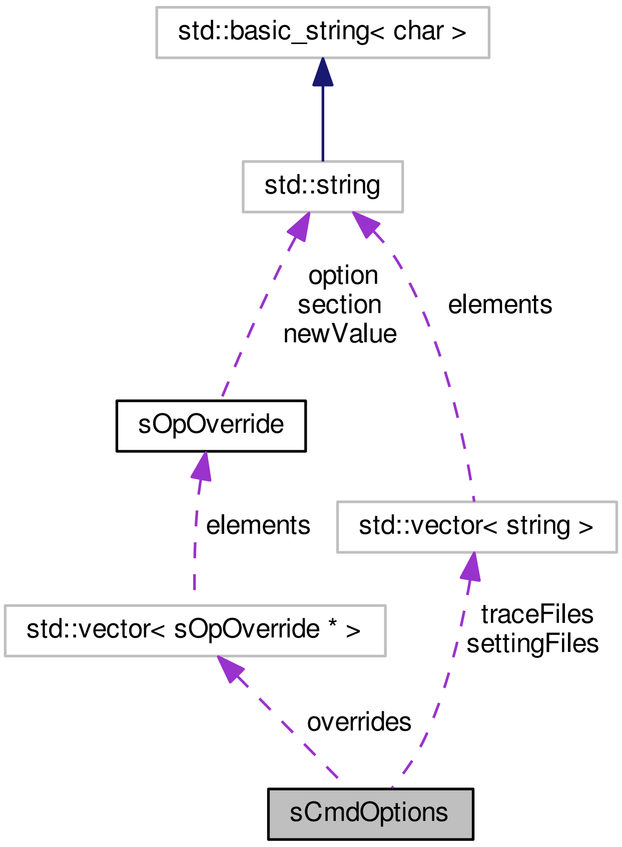
- include/round_robin.h

4.21 sCmdOptions Struct Reference

All the settings files, traces and setting overrides from command line.

`#include <data_structs.h>`

Collaboration diagram for sCmdOptions:



Public Attributes

- vector< string > **settingFiles**
- vector< string > **traceFiles**
- vector< sOpOverride $*$ > **overrides**

The documentation for this struct was generated from the following file:

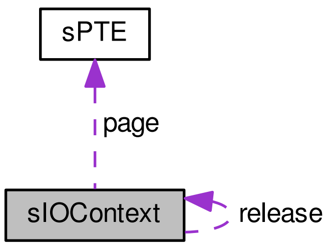
- include/data_structs.h

4.22 sIOContext Struct Reference

Struct for holding the context of an I/O operation.

`#include <io_control.h>`

Collaboration diagram for sIOContext:



Public Attributes

- uint32_t pid

  PID of process that this I/O is for.
- sPTE ∗ page

  Page that is being either moved in or out.
- int time

  Time remaining on this I/O.
- sIOContext ∗ release

  Release this waiting I/O when this one finishes.

4.22.1 Detailed Description

The documentation for this struct was generated from the following file:

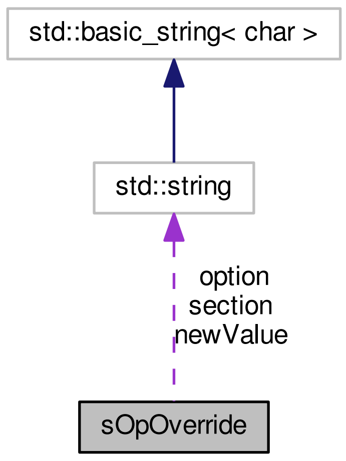
- include/io_control.h

4.23 sOpOverride Struct Reference

Struct for holding a settings overrides from command line.

`#include <data_structs.h>`

Collaboration diagram for sOpOverride:



Public Attributes

- string **section**
- string **option**
- string **newValue**

The documentation for this struct was generated from the following file:

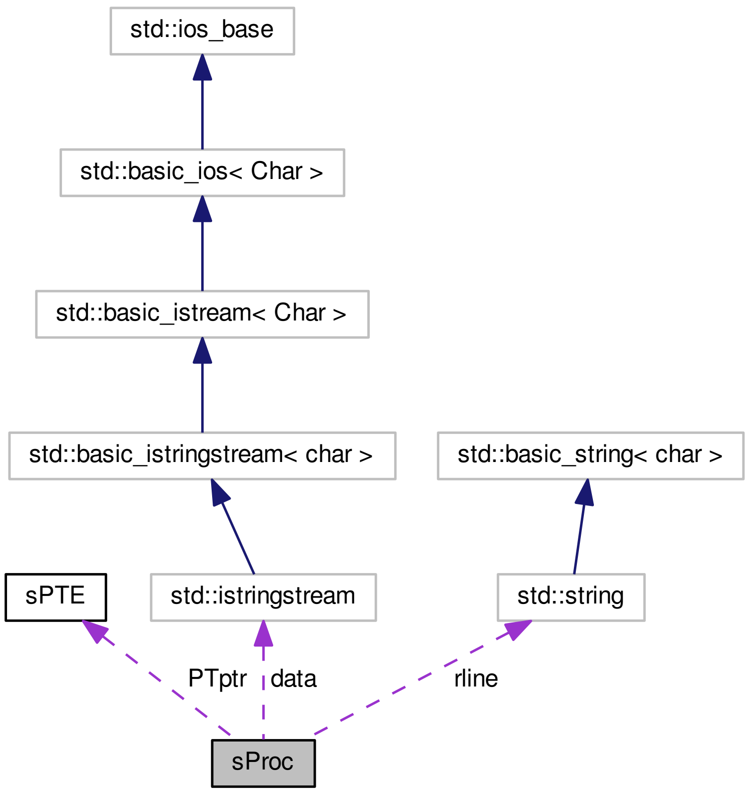
- include/data_structs.h

4.24 sProc Struct Reference

Struct representing a process.

`#include <data_structs.h>`

Collaboration diagram for sProc:



Public Attributes

- unsigned int pid

    Process PID.
- uint16_t cswitches

    # of Context Switches
- int **pageFaults**
- int **tlbhit**
- int **tlbmiss**
- int clockTime

    Time spent executing.
- int finishTime

    VC when process finished.

- istringstream ∗ data

  Text data of process.
- int PC

  Program Counter.
- int maxPC

  Max PC.
- bool restart

  Used to flag an instruction restart.
- bool circularFaultCheck

  If the same process faults on the same instruction more than once the cpu notifies the core.
- string rline

  Process instruction for the restart.
- sPTE ∗ PTptr

  Pointer to process' page table.
- void ∗ scheduleData

  Scheduler specific data.

4.24.1 Member Data Documentation

4.24.1.1 int **sProc::maxPC**

Based on size of process file

Referenced by cVMM::cleanupProcess(), and cCPU::run().

4.24.1.2 void∗ **sProc::scheduleData**

Always round robin for this project

The documentation for this struct was generated from the following file:

- include/data_structs.h

4.25 sPTE Struct Reference

A struct representing a single page table entry.

```
#include <data_structs.h>
```

Public Attributes

- uint32_t frame

  Frame that this PTE maps to.

- int timestamp

    This gets set in hardware by the MMU.

- bool flags [3]

    Page Table Entry (PTE) VM flags.

- uint8_t time

    This is set in software by the pr_lruApprox after each quanta or on a page fault.

4.25.1 Member Data Documentation

4.25.1.1 bool **sPTE::flags**[3]

flags[0]: Present/absent flags[1]: Dirty flags[2]: Referenced

Referenced by cMMU::addTLB(), cVMM::cleanupProcess(), cPRFifo::clearPages(), cPRLruApprox::clearPages(), cPRLru::clearPages(), cMMU::flushTLB(), cPRFifo::resolvePageFault(), cPRLruApprox::resolvePageFault(), cPRLru::resolvePageFault(), cMMU::syncTLB(), cIOControl::tick(), and cPRLruApprox::updateTime().

The documentation for this struct was generated from the following file:

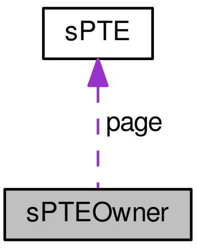
- include/data_structs.h

4.26 sPTEOwner Struct Reference

A struct used to associate a process with a particular page.

```
#include <data_structs.h>
```

Collaboration diagram for sPTEOwner:



Public Attributes

- uint32_t **pid**
- sPTE ∗ **page**

4.26.1 Detailed Description

The documentation for this struct was generated from the following file:

- include/data_structs.h

4.27 sTLBE Struct Reference

Struct representing a single entry in the TLB.

```
#include <mmu.h>
```

Public Attributes

- uint32_t **VPN**
- uint32_t **frame**
- int **timestamp**
- bool valid

  Is this a valid translation?
- bool **dirty**
- bool **ref**

4.27.1 Member Data Documentation

4.27.1.1 bool sTLBE::valid

If false, the MMU will ignore any cache hits on this entry. This is also used in flushing the tlb.

Referenced by cMMU::addTLB(), and cMMU::flushTLB().

The documentation for this struct was generated from the following file:

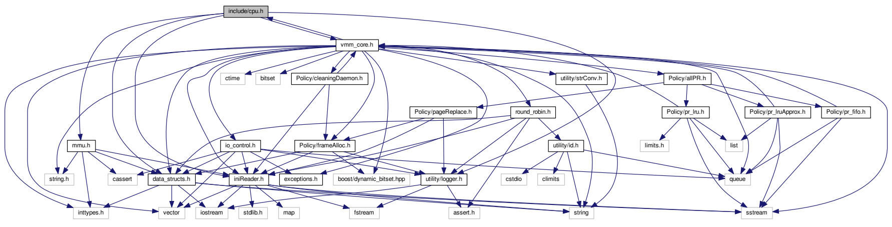
- include/mmu.h

Chapter 5

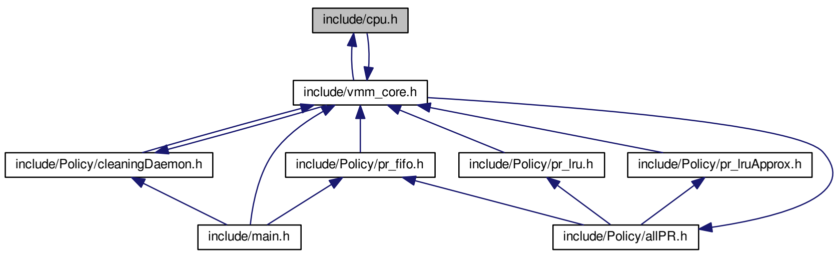
File Documentation

5.1 include/cpu.h File Reference

`#include "mmu.h"` `#include "data_structs.h"` `#include "iniReader.h"` `#include "utility/logger.h"` `#include "vmm_core.h"` Include dependency graph for cpu.h:



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



Classes

- class cCPU

    Very Simple CPU.

Enumerations

- enum eCPUState { CPU_OK = 0x1, CPU_PF = CPU_OK << 1, CPU_TERM = CPU_PF << 1, CPU_EX = CPU_TERM << 1, CPU_CIRC_PF = CPU_EX << 1 }

    Status of the execution quantum termination.

5.1.1 Detailed Description

5.1.2 Enumeration Type Documentation

5.1.2.1 enum **eCPUState**

Enumerator:

CPU_OK CPU finished executing instruction ok.

CPU_PF CPU/MMU incured a page fault.

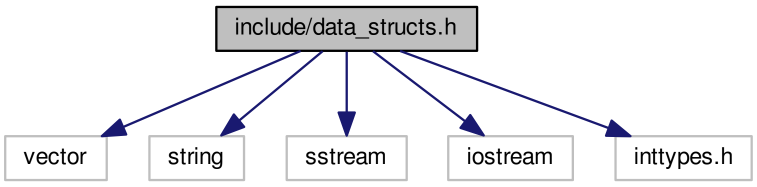
CPU_TERM Process execution termination.

CPU_EX Process exception.

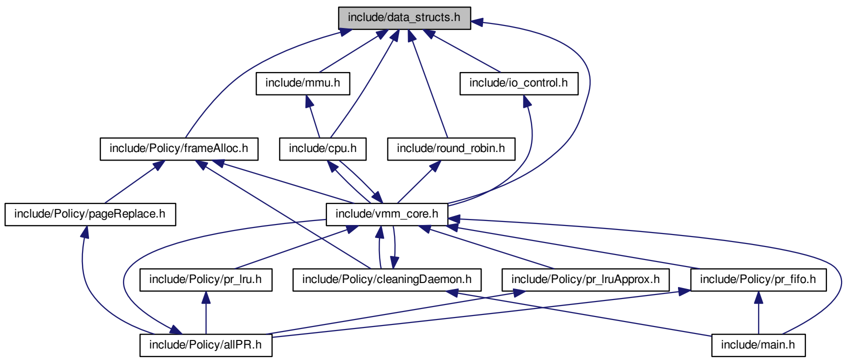
CPU_CIRC_PF Potential circular page fault detected by cpu.

5.2 include/data_structs.h File Reference

`#include <vector>` `#include <string>` `#include <sstream>` × `#include <iostream>` `#include <inttypes.h>` Include dependency graph for data_structs.h:



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



Classes

- struct sPTE

  A struct representing a single page table entry.
- struct sPTEOwner

  A struct used to associate a process with a particular page.
- struct sProc

  Struct representing a process.
- struct sOpOverride

  Struct for holding a settings overrides from command line.
- struct sCmdOptions

  All the settings files, traces and setting overrides from command line.

Enumerations

- enum eFlagIndex { FI_PRESENT, FI_DIRTY, FI_REF }

  Enum used for indexing PTE flags.

5.2.1 Detailed Description

5.2.2 Enumeration Type Documentation

5.2.2.1 enum **eFlagIndex**

Enumerator:

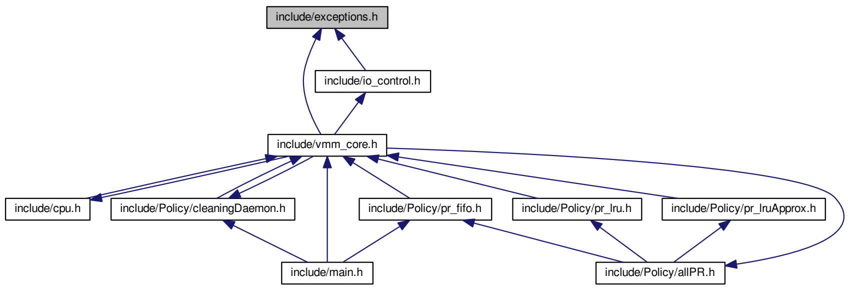
FI_PRESENT Access PTE present bit.

FI_DIRTY Access PTE dirty bit.

FI_REF Access PTE reference bit.

5.3 include/exceptions.h File Reference

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



Classes

- class cException

  Generic exception class.
- class cVMMExc

  A class handling exceptions in the VMM Core.
- class cIOExc

  Class handling exceptions in the I/O system.
- class cPRExc

  Exception specific to the PR module.

Enumerations

- enum eExType { PR_NO_FRAMES_AVAIL }

  Distinguish between exceptions in a generic class.

5.3.1 Detailed Description

5.3.2 Enumeration Type Documentation

5.3.2.1 enum **eExType**

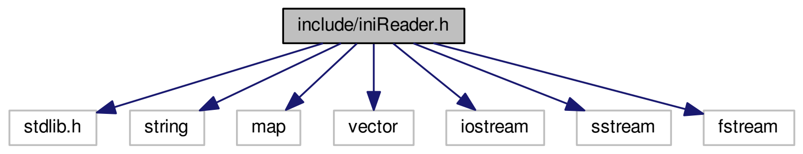
Since we didn't really have time to expand the exception system for this project this never really expanded.

Enumerator:

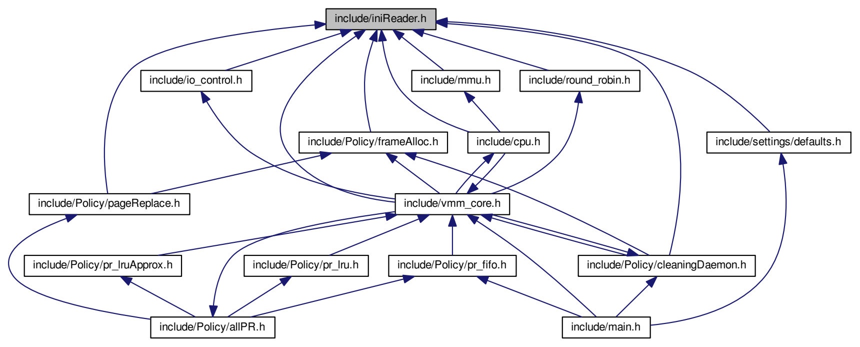
PR_NO_FRAMES_AVAIL No frames are available. Thrown by PRModule

5.4 include/iniReader.h File Reference

`#include <stdlib.h>` `#include <string>` `#include <map>` `#include <vector>` `#include <iostream>` `#include <sstream>` × `#include <fstream>` Include dependency graph for iniReader.h:



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



Classes

- struct KeyRecord

  Struct for holding a single settings key and its value.

- class INIReader

  Class for parsing and storing contents of .ini files.

Defines

- #define **STR**(x) #x
- #define **EXTRACT**(type, sec, opt) settings.extractValue<type>(STR(sec),ST-R(opt))
- #define **EXTRACTP**(type, sec, opt) settings->extractValue<type>(STR(sec),S-TR(opt))
- #define **EXTRACT_**(obj, type, sec, opt) settings.extractValue<type>(STR(sec),-STR(opt))
- #define **EXTRACTP_**(obj, type, sec, opt) settings->extractValue<type>(ST-R(sec),STR(opt))

Typedefs

- typedef map< string, KeyRecord > **KeyMap**
- typedef map< string, KeyMap > **SectionMap**
- typedef map< string, string > **DefaultKeyMap**
- typedef map< string, DefaultKeyMap > **DefaultMap**

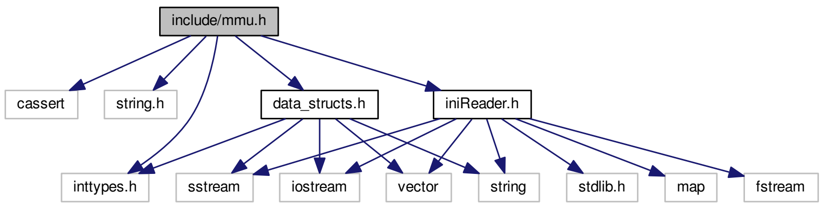
Functions

- **__attribute__** ((unused)) static bool strEq(const string &s1

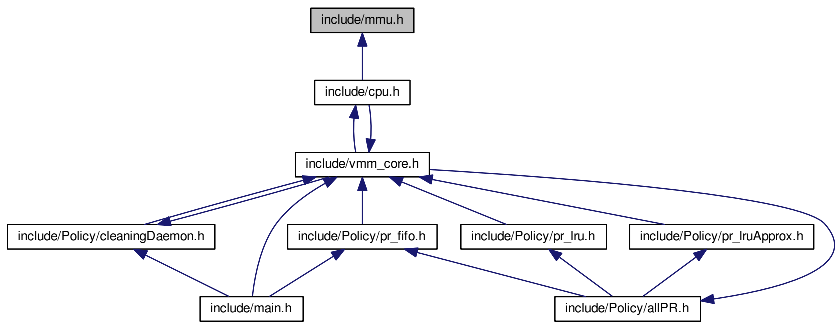
5.4.1 Detailed Description

5.5 include/mmu.h File Reference

`#include <cassert>` `#include <string.h>` `#include <inttypes.-h>` `#include "data_structs.h"` `#include "iniReader.h"` Include dependency graph for mmu.h:



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



Classes

- struct sTLBE

    Struct representing a single entry in the TLB.

- class cMMU

    Class representing an MMU in the CPU.

Enumerations

- enum eMMUstate { MMU_THIT, MMU_TMISS, MMU_PF }

    State of the MMU after translation (or attempted)

5.5.1 Detailed Description

5.5.2 Enumeration Type Documentation

5.5.2.1 enum eMMUstate

Enumerator:

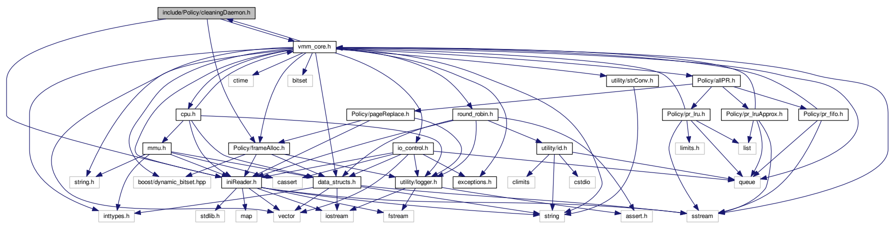
MMU_THIT TLB Hit.

MMU_TMISS TLB Miss.

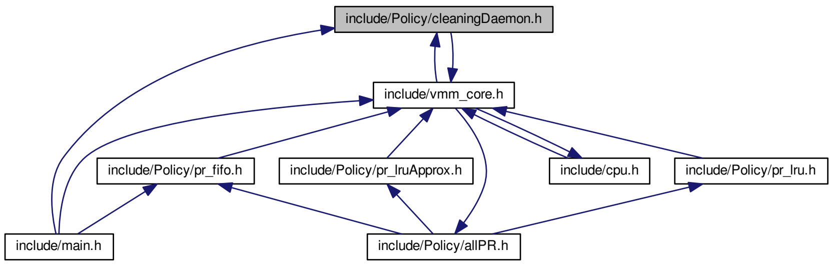
MMU_PF Page Fault.

5.6 include/Policy/cleaningDaemon.h File Reference

`#include "iniReader.h"` `#include "Policy/frameAlloc.h"` × `#include "vmm_core.h"` Include dependency graph for cleaningDaemon.h:



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



Classes

- class cCleanDaemon

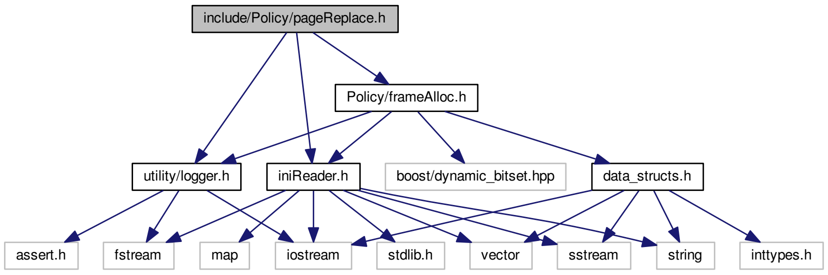
    Cleaning Daemon for system.

5.6.1 Detailed Description

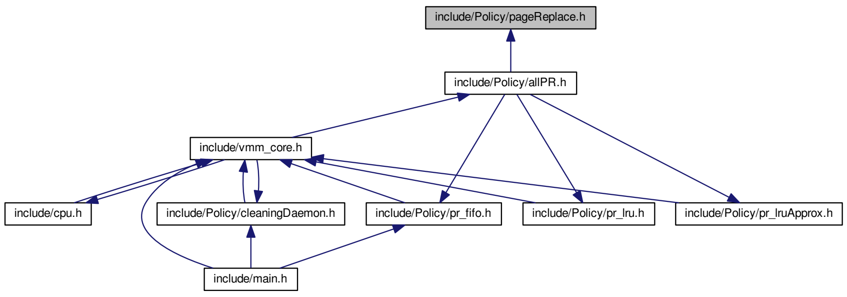
5.7 include/Policy/pageReplace.h File Reference

`#include "iniReader.h"` `#include "utility/logger.h"` `#include`

`"Policy/frameAlloc.h"` Include dependency graph for pageReplace.h:



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



Classes

- class cPRPolicy

  Abstract Page Replacement Policy.

Enumerations

- enum ePRStatus { PR_SERVICED, PR_SERVICED_IO, PR_NO_AVAIL, PR_N-O_ACTION }

5.7.1 Detailed Description

5.7.2 Enumeration Type Documentation

5.7.2.1 enum **ePRStatus**

Enumerator:

PR_SERVICED Page fault serviced.

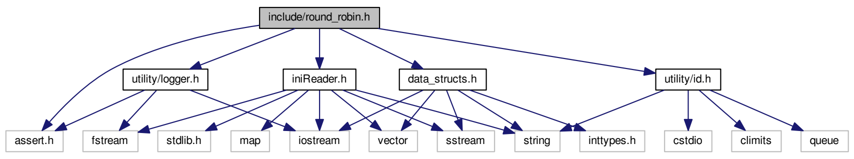
PR_SERVICED_IO Page fault serviced with I/O required.

PR_NO_AVAIL Page fault not serviced. No frames available (all pinned)

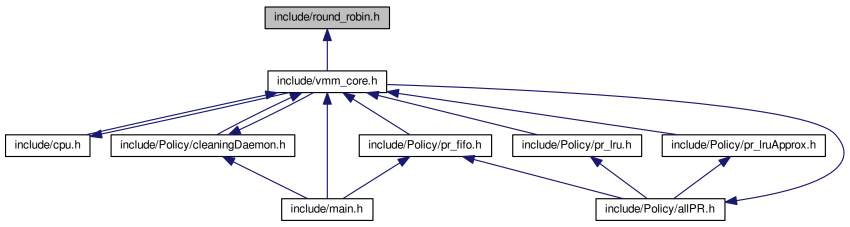
PR_NO_ACTION Used in returning from circular page fault handler.

5.8 include/round_robin.h File Reference

`#include <assert.h>` `#include "iniReader.h"` `#include "data-_structs.h"` `#include "utility/id.h"` `#include "utility/logger.-h"` Include dependency graph for round_robin.h:



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



Classes

- class cRoundRobin

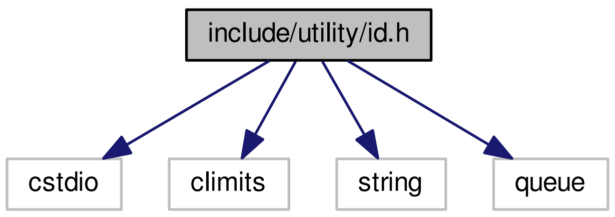
  Round Robin Scheduler.
- struct roundRobinInfo

  Struct containing process info specific for Round-Robin scheduling.

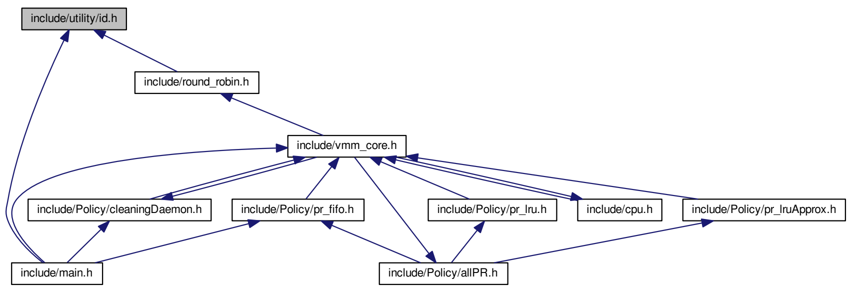
5.8.1 Detailed Description

5.9 include/utility/id.h File Reference

`#include <cstdio> #include <climits> #include <string>` ×
`#include <queue>` Include dependency graph for id.h:



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



Classes

- class cIDManager

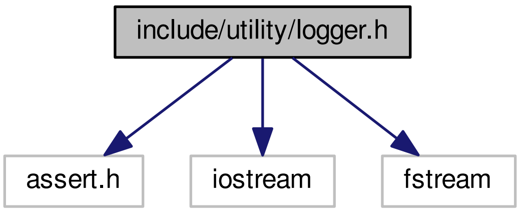
  A class for managing unique IDs.

5.9.1 Detailed Description

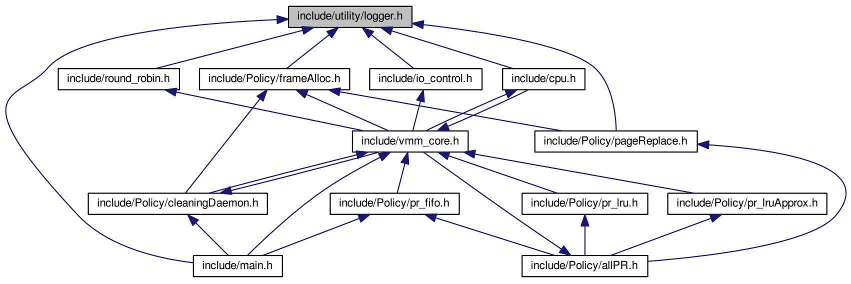
5.10 include/utility/logger.h File Reference

```
#include <assert.h> #include <iostream> #include <fstream> ×
```

Include dependency graph for logger.h:



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



Functions

- FILE ∗ initLog (const char ∗filename)

  Initialize a trace log at filename.
- void closeLog ()

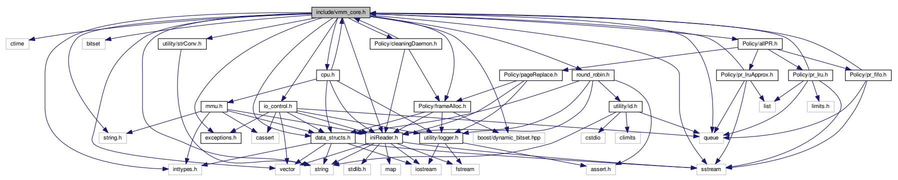
  Close the file stream for the trace log.
- FILE ∗ getStream ()

  Get the file stream to write to.

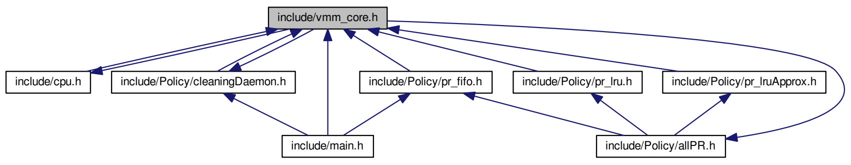
5.10.1 Detailed Description

5.11 include/vmm_core.h File Reference

`#include <ctime>` `#include <vector>` `#include <queue>` ×
`#include <bitset>` `#include <string>` `#include <string.h>` `#include <inttypes.h>` `#include <sstream>` `#include <boost/dynamic_bitset.hpp>` `#include "cpu.h"` `#include "data_structs.h"` `#include "iniReader.h"` `#include "exceptions.h"` `#include "round_robin.h"` `#include "io_control.h"` `#include "Policy/allPR.h"` `#include "utility/strConv.h"` `#include "Policy/frameAlloc.h"` `#include "Policy/cleaningDaemon.h"` ×
Include dependency graph for vmm_core.h:



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



Classes

- class cVMM

  Virtual Memory Manager core class.

Variables

- INIReader ∗ **settings**

5.11.1 Detailed Description