5103 Project 1

Generated by Doxygen 1.7.6.1

Sun Feb 12 2012 14:01:16

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

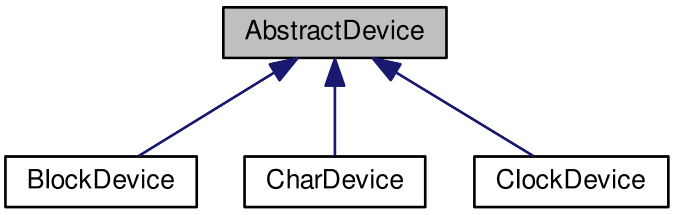
Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 AbstractDevice Class Reference

Inheritance diagram for AbstractDevice:



Public Member Functions

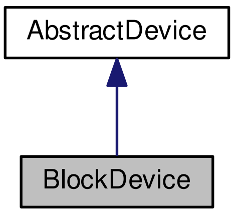
- virtual void **setTimer** (int time)=0
- virtual void **disarm** ()=0

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

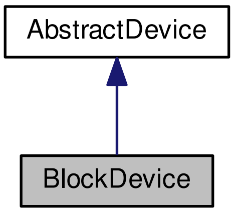
- include/devices/abstract_device.h

4.2 BlockDevice Class Reference

Inheritance diagram for BlockDevice:



Collaboration diagram for BlockDevice:



Public Member Functions

- void **setTimer** (int usec)
- void **disarm** ()

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

- include/devices/block_device.h
- src/devices/block_device.cpp

4.3 cCPU Class Reference

A class for emulating a simple cpu.

```
#include <cpu.h>
```

Public Member Functions

- void setText (char ∗text)

  Set the program text.
- unsigned int getSetPC (unsigned int newPC)

  Get/Set the program counter.
- int getSetVC (int newVC)

  Get/Set the VC.
- uint16_t getSetPSW (uint16_t newPSW)

  Get/Set the PSW.
- uint16_t getPSW ()

  Get the Program Status Word.
- void setPSW (uint16_t newPSW)

  Set a new value for the PSW.
- char ∗ getParam (int num)

  Get execution parameters from the cpu.
- char getOpcode ()

  Get the current Opcode.
- void run ()

  Start execution.

Private Member Functions

- int **tokenizeLine** ()

Private Attributes

- bool **KMode**
- unsigned int **PC**
- unsigned int **maxPC**
- int **VC**
- uint16_t PSW

  Program Status Word.
- char ∗ execText

  Text data for currently executing process.
- char tokenBuffer [2][MAX_PARAM_SIZE]

  Holds the tokenized execution parameters.
- char Opcode

  Holds the current Opcode.

4.3.1 Detailed Description

This class emulates the internals of a very simple cpu with two main registers, PC and VC. In addition, it has other state for handling system calls and program exceptions.

4.3.2 Member Function Documentation

4.3.2.1 char **cCPU::getOpcode** ( )

Get the current Opcode in the cpu. This is used by the kernel to determine which system call as being made. Used in conjunciton with cCPU::getParam the kernel can process system calls.

4.3.2.2 char ∗ **cCPU::getParam** (int *num*)

Fetch the given execution paramter from the cpu's internal buffer. When an instruction is encountered that has parameters associated with it, the cpu tokenizes them and places it in an internal buffer. This function is mainly used by the kernel in handling system calls.

Parameters

num	Must be less than MAX_PARAMS (currently 2)

Returns

Returns a char∗ which points to a string of at most MAX_PARAM_SIZE - 1 bytes.

4.3.2.3 uint16_t **cCPU::getPSW** ( )

Returns the program status word which is a unsigned 16-bit integer type with flags from ePSW set. These are used by the kernel to make action decisions.

4.3.2.4 unsigned int **cCPU::getSetPC** (unsigned int *newPC*)

Get the current value for the program counter and then set its value to the given parameter. This is useful for swapping out process values.

4.3.2.5 uint16_t **cCPU::getSetPSW** (uint16_t *newPSW*)

Get the current value for the PSW and set its value to the given parameter.

4.3.2.6 int cCPU::getSetVC (int *newVC*)

Get the current value for VC and set its value to the given paramter. This is useful for swapping out process values.

4.3.2.7 void cCPU::run ( )

Once all appropriate process data is entered by the kernel this function is called to start execution. Any time control needs to be returned to the kernel this function will return with the appropriate PSW flags set for the kernel to act on.

4.3.2.8 void cCPU::setPSW (uint16_t *newPSW*)

Used by the kernel to reset the PSW after a system call. Any process execution which returns to the kernel but does not terminate the process should reset the PSW so subsequent exceptions/terminations are not lost by stray PSW values.

4.3.2.9 void cCPU::setText (char * *text*)

Point the cpu to the text data for the running process. This text is indexed using the program counter (PC).

Parameters

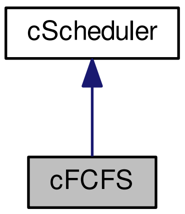
text	Program text pointer. assert(text != NULL)

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

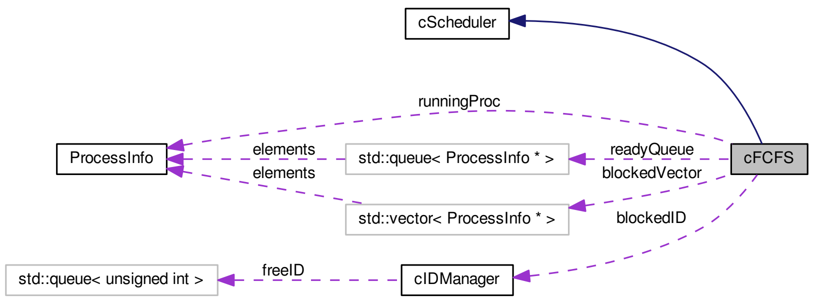
- include/cpu.h
- src/cpu.cpp

4.4 cFCFS Class Reference

Inheritance diagram for cFCFS:



Collaboration diagram for cFCFS:



Public Member Functions

- void **initProcScheduleInfo** (ProcessInfo ∗)
- void addProcess (ProcessInfo ∗)

  Transfer control of process state and scheduling.
- void **setBlocked** (ProcessInfo ∗)
- void unblockProcess (ProcessInfo ∗)

  Unblock a process and make it ready.
- void removeProcess (ProcessInfo ∗)

  Remove a process from the control of the scheduler.
- ProcessInfo ∗ getNextToRun ()

Query the scheduler for next process to run.

- pidType **numProcesses** ()

Private Attributes

- queue< ProcessInfo ∗ > **readyQueue**
- vector< ProcessInfo ∗ > **blockedVector**
- ProcessInfo ∗ **runningProc**
- pthread_mutex_t **blockedLock**
- pthread_cond_t **allBlocked**
- cIDManager **blockedID**

4.4.1 Member Function Documentation

4.4.1.1 void cFCFS::addProcess (ProcessInfo ∗) `[virtual]`

After this is called, the kernel core no longer keeps track of the given process. Once the process is created and deemed ready by the kernel it is handed off here. The scheduler is then in charge of state transitions when the kernel gives it appropriate notifications.

Parameters

ProcessInfo∗	Process to add under scheduler's control

Implements cScheduler.

4.4.1.2 ProcessInfo ∗ cFCFS::getNextToRun ( ) `[virtual]`

After this function is called, it should be assumed by any scheduler implementation that the kernel will run the given process (unless otherwise notified). The currently running process should implicitly be considered for running next (again).

If there are processes left but all are blocked. This function should block until it receives a signal that a process is unblocked.

Returns

ProcessInfo∗ Ready process to run next. May be the same as the currenlty running one.

Implements cScheduler.

4.4.1.3 void cFCFS::removeProcess (ProcessInfo ∗) `[virtual]`

When a process terminates, either through normal means or an exception, the kernel will call this function to release a process from the scheduler's control. The scheduler should clean up any internal state for the process. Deallocation of process resources is left to the kernel.

Parameters

Process-Info∗	Process to remove from scheduler

Implements cScheduler.

4.4.1.4 void cFCFS::unblockProcess (ProcessInfo ∗) `[virtual]`

When a process has completed a blocking call the kernel will notify the scheduler that it should be unblocked. This operation should be very fast since it will likely be called from a signal handler.

Parameters

Process-Info∗	Process to unblock

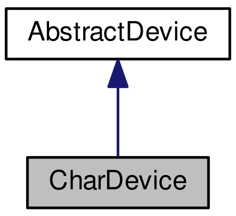
Implements cScheduler.

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

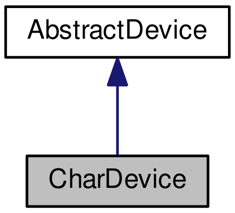
- include/scheduler/fcfs.h
- src/scheduler/fcfs.cpp

4.5 CharDevice Class Reference

Inheritance diagram for CharDevice:



Collaboration diagram for CharDevice:



Public Member Functions

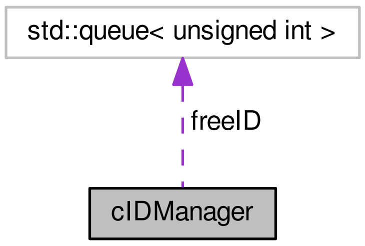
- void **setTimer** (int usec)
- void **disarm** ()

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

- include/devices/char_device.h
- src/devices/char_device.cpp

4.6 cIDManager Class Reference

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.
- void returnID (unsigned int id)

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

Private Attributes

- queue< unsigned int > **freeID**
- unsigned int **baseID**
- unsigned int **currentID**
- bool **consumeQueue**

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.

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

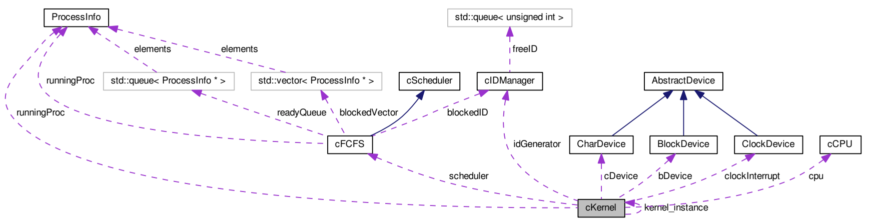
If the ID is not equal to the one last given then it is added to a 'free queue'. If it is equal to the last one reserved then the ID counter is simply decremented If this last case happens, it causes cIDManager::getID to stop consuming from the queue and return this newly availabe ID.

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

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

4.7 cKernel Class Reference

Collaboration diagram for cKernel:



Public Member Functions

- cKernel ()

  Default cKernel constructor.
- void boot ()

  Start the 'OS' Kernel.
- void initProcess (const char ∗filename, pidType parent, int priority=DEFAULT_P-RIORITY)

  Initialize a Process.
- void cleanupProcess (pidType pid)

  Cleans up a terminated process.
- void **_sysCall** (const char call)

Private Member Functions

- void swapProcesses (ProcessInfo ∗)

  Swap a process on the cpu.
- void **cleanupProcess** (ProcessInfo ∗)
- void sigHandler (int signum, siginfo_t ∗info)

  Handler for all signals.

Static Private Member Functions

- static void **sig_catch** (int signum, siginfo_t ∗info, void ∗context)

Private Attributes

- cCPU **cpu**
- int **clockTick**

- BlockDevice **bDevice**
- CharDevice **cDevice**
- ClockDevice **clockInterrupt**
- ProcessInfo ∗ runningProc

  ProcessInfo
- cIDManager **idGenerator**
- schedulerType **scheduler**

Static Private Attributes

- static cKernel ∗ **kernel_instance**

4.7.1 Constructor & Destructor Documentation

4.7.1.1 cKernel::cKernel ( )

The default constructor initializes all internal datastructures and loads the initial program (default: 'main.trace') but does not run it.

4.7.2 Member Function Documentation

4.7.2.1 void cKernel::boot ( )

Starts the main kernel loop. The initial program is loaded and execution follows from there.

Exceptions

kernelError	

4.7.2.2 void cKernel::cleanupProcess (pidType *pid*)

Cleans up any memory and kernel entries associated with the terminated process. Also removes the process from the scheduler.

4.7.2.3 void cKernel::initProcess (const char ∗ *filename,* pidType *parent,* int *priority =* DEFAULT_PRIORITY)

Initializes a process by loading program file contents, setting default process values and adding it in a ready state to the scheduler.

4.7.2.4 void cKernel::sigHandler (int *signum*, siginfo_t ∗ *info*) [private]

Each device uses a timer from SIGRTMIN -> SIGRTMAX. Because these are not required to be compile time constants, a switch statement cannot be used. When a signal is received by the kernel, careful consideration must be made to the current state. Below is each signal and how it is handled:

4.7.2.5 void cKernel::swapProcesses (ProcessInfo ∗ *proc*) [private]

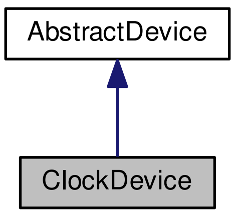
Takes the process in its parameter and swaps it with the one currently running in the cpu.

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

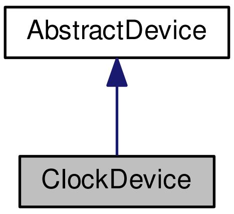
- include/kernel.h
- src/kernel.cpp

4.8 ClockDevice Class Reference

Inheritance diagram for ClockDevice:



Collaboration diagram for ClockDevice:



Public Member Functions

- void setTimer (int usec)

  Set the timer to go off.
- void disarm ()

  Disarm the timer.
- int getTime ()

  Get how much time is remaining.

4.8.1 Member Function Documentation

4.8.1.1 void ClockDevice::disarm ( ) [virtual]

Tries to disarm the timer.

Exceptions

std::string	error message

Implements AbstractDevice.

4.8.1.2 int ClockDevice::getTime ( )

Returns the remaining time until a signal is produced.

Returns

int Time left in microseconds

4.8.1.3 void ClockDevice::setTimer (int *usec*) [virtual]

Sets timer to send signal CLOCKSIG in usec microseconds.

Parameters

usec	Time in microseconds.

Exceptions

std::string	error message

Implements AbstractDevice.

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

- include/devices/clock_device.h
- src/devices/clock_device.cpp

4.9 cRoundRobin Class Reference

Public Member Functions

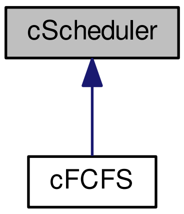
- void **initProcScheduleInfo** (ProcessInfo ∗)
- void **addProcess** (ProcessInfo ∗)
- void **setBlocked** (ProcessInfo ∗)
- void **unblockedProcess** (ProcessInfo ∗)
- void **removeProcess** (ProcessInfo ∗)
- ProcessInfo ∗ **getNextToRun** ()
- pidType **numProcesses** ()

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

- include/scheduler/round_robin.h

4.10 cScheduler Class Reference

Inheritance diagram for cScheduler:



Public Member Functions

- virtual void **initProcScheduleInfo** (ProcessInfo ∗)=0
- virtual void addProcess (ProcessInfo ∗)=0

  Transfer control of process state and scheduling.
- virtual void **setBlocked** (ProcessInfo ∗)=0
- virtual void unblockProcess (ProcessInfo ∗)=0

  Unblock a process and make it ready.
- virtual void removeProcess (ProcessInfo ∗)=0

  Remove a process from the control of the scheduler.
- virtual ProcessInfo ∗ getNextToRun ()=0

  Query the scheduler for next process to run.
- virtual pidType **numProcesses** ()=0

4.10.1 Member Function Documentation

4.10.1.1 void cScheduler::addProcess (ProcessInfo ∗) `[pure virtual]`

After this is called, the kernel core no longer keeps track of the given process. Once the process is created and deemed ready by the kernel it is handed off here. The scheduler is then in charge of state transitions when the kernel gives it appropriate notifications.

Parameters

ProcessInfo∗	Process to add under scheduler's control

Implemented in cFCFS.

4.10.1.2 ProcessInfo ∗ cScheduler::getNextToRun ( ) [pure virtual]

After this function is called, it should be assumed by any scheduler implementation that the kernel will run the given process (unless otherwise notified). The currently running process should implicitly be considered for running next (again).

If there are processes left but all are blocked. This function should block until it receives a signal that a process is unblocked.

Returns

ProcessInfo∗ Ready process to run next. May be the same as the currenlty running one.

Implemented in cFCFS.

4.10.1.3 void cScheduler::removeProcess (ProcessInfo ∗) [pure virtual]

When a process terminates, either through normal means or an exception, the kernel will call this function to release a process from the scheduler's control. The scheduler should clean up any internal state for the process. Deallocation of process resources is left to the kernel.

Parameters

ProcessInfo∗	Process to remove from scheduler

Implemented in cFCFS.

4.10.1.4 void cScheduler::unblockProcess (ProcessInfo ∗) [pure virtual]

When a process has completed a blocking call the kernel will notify the scheduler that it should be unblocked. This operation should be very fast since it will likely be called from a signal handler.

Parameters

ProcessInfo∗	Process to unblock

Implemented in cFCFS.

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

- include/scheduler/scheduler.h

4.11 fcfsInfo Struct Reference

Public Attributes

- unsigned int **blockedIndex**

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

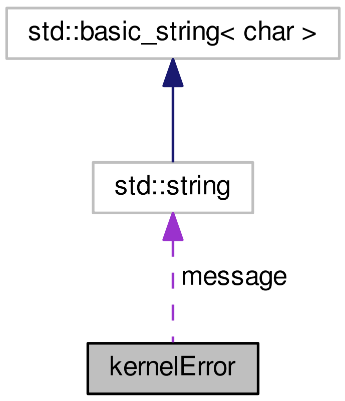
- include/scheduler/fcfs.h

4.12 kernelError Struct Reference

Struct containing kernel crash information.

`#include <kernel.h>`

Collaboration diagram for kernelError:



Public Attributes

- string **message**

4.12.1 Detailed Description

When the kernel crashes, important information is placed in here and then handled by the main function in init.cpp.

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

- include/kernel.h

4.13 ProcessInfo Struct Reference

Structure for containing process state and data.

```
#include <process.h>
```

Public Attributes

- unsigned int **parent**
- unsigned int **pid**
- unsigned int **startCPU**
- unsigned int **totalCPU**
- eProcState **state**
- uint16_t **PSW**
- int **priority**
- unsigned int **PC**
- int **VC**
- char ∗ **processText**
- void ∗ scheduleData

    Scheduler specific data.
- unsigned long **memory**

4.13.1 Detailed Description

This struture is created in the kernel when a process is initialized. It contains all process data needed for execution and for the kernel/scheduler to make desciions on it.

4.13.2 Member Data Documentation

4.13.2.1 void∗ ProcessInfo::scheduleData

Check specific scheduler docs for the contents of this pointer. Since the process struct remains static, this gives the ability for schedulers to store their own state without the kernel having to know ahead of time.

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

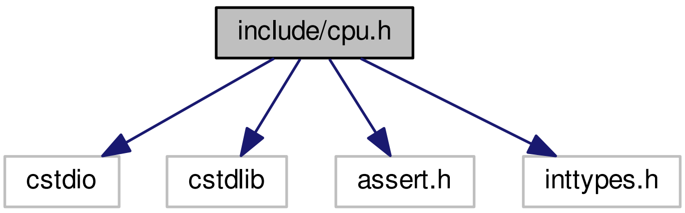
- include/process.h

Chapter 5

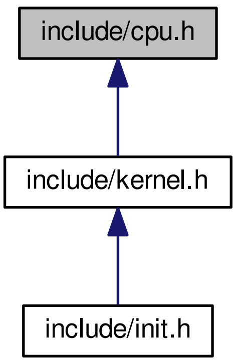
File Documentation

5.1 include/cpu.h File Reference

`#include <cstdio>` `#include <cstdlib>` `#include <assert.-h>` `#include <inttypes.h>` Include dependency graph for cpu.h:



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



Classes

- class cCPU

  A class for emulating a simple cpu.

Defines

- #define MAX_PARAMS 2

  Max number of execution parameters for any Opcode.
- #define MAX_PARAM_SIZE 256

  Maximum size in bytes for an execution parameter.

Enumerations

- enum ePSW { PS_EXCEPTION = 0x1, PS_TERMINATE = PS_EXCEPTION << 1, PS_SYSCALL = PS_TERMINATE << 1 }

  Enumeration of Program Status Word Flags.

5.1.1 Detailed Description

5.1.2 Define Documentation

5.1.2.1 #define MAX_PARAM_SIZE 256

Creates exception if exceeded.

5.1.3 Enumeration Type Documentation

5.1.3.1 enum ePSW

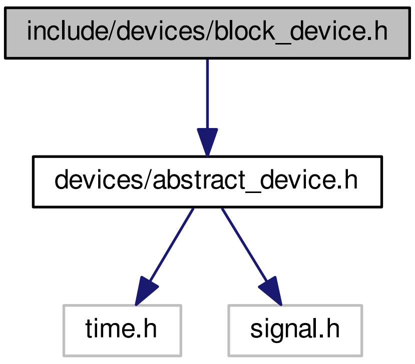
The program status word is a bit vector and this enumeration defines the meaning of particular bits. This is used in the interpretation of execution status.

Enumerator:

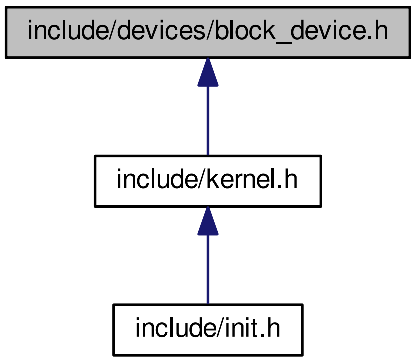
PS_EXCEPTION	Executing process has created an exception.
PS_TERMINATE	Executing process has finished.
PS_SYSCALL	Executing process has made a system call.

5.2 include/devices/block_device.h File Reference

`#include "devices/abstract_device.h"` Include dependency graph for block_device.h:



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



Classes

- class BlockDevice

Defines

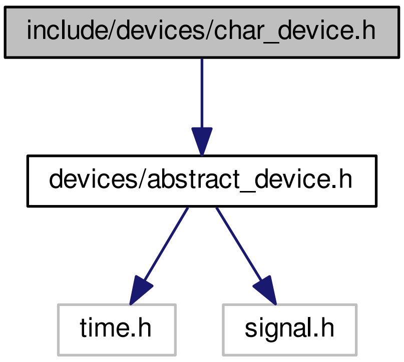
- #define BLOCKSIG SIGRTMIN + 1

  Signal generated by BlockDevice.

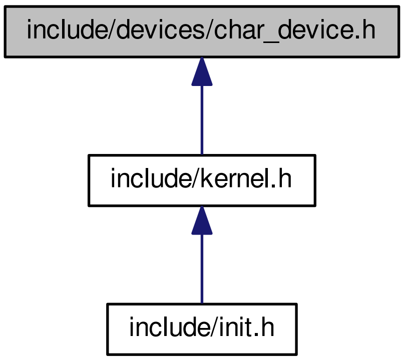
5.2.1 Detailed Description

5.3 include/devices/char_device.h File Reference

`#include "devices/abstract_device.h"` Include dependency graph for char_device.h:



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



Classes

- class CharDevice

Defines

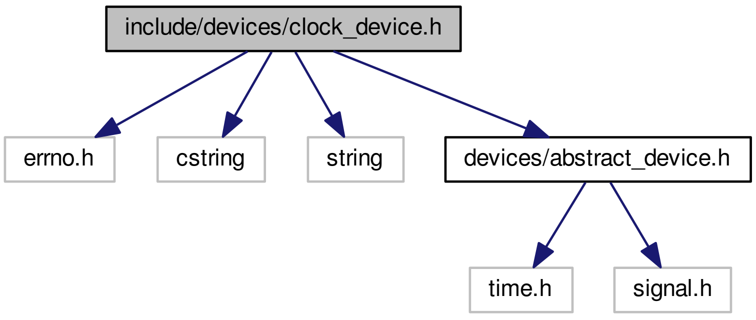
- #define CHARSIG SIGRTMIN + 2

  Signal generated by CharDevice.

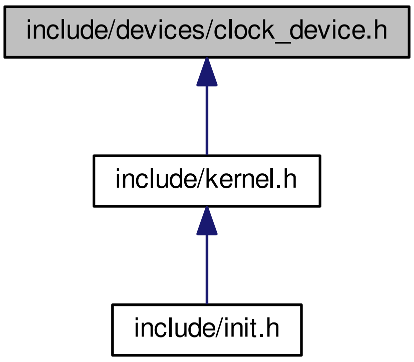
5.3.1 Detailed Description

5.4 include/devices/clock_device.h File Reference

`#include <errno.h>` `#include <cstring>` `#include <string>` ×
`#include "devices/abstract_device.h"` Include dependency graph for clock_device.h:



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



Classes

- class ClockDevice

Defines

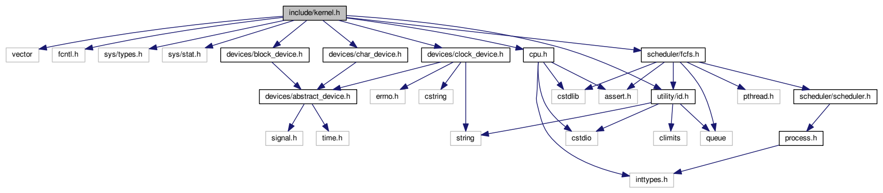
- #define **CLOCKID** CLOCK_REALTIME
- #define CLOCKSIG SIGRTMIN

  Signal generated by ClockDevice.

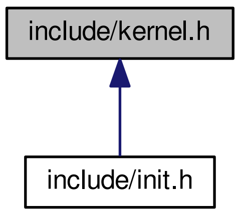
5.4.1 Detailed Description

5.5 include/kernel.h File Reference

`#include <vector>` `#include <fcntl.h>` `#include <sys/types.h>` `#include <sys/stat.h>` `#include "cpu.h"` `#include "devices/char_device.h"` `#include "devices/block_device.h"` `#include "devices/clock_device.h"` `#include "utility/id.h"` `#include` `"scheduler/fcfs.h"` Include dependency graph for kernel.h:



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



Classes

- class cKernel
- struct kernelError

  Struct containing kernel crash information.

Defines

- #define DEFAULT_TIMER 1000

  Default timer value for devices.
- #define DEFAULT_PRIORITY 5

  Default priority assigned to newly created processes.

Typedefs

- typedef cFCFS **schedulerType**

Variables

- static const char initProcessName [] = "main.trace"

  Name of the first program to run on the system.

5.5.1 Detailed Description

5.5.2 Define Documentation

5.5.2.1 #define DEFAULT_PRIORITY 5

Only used if no other priority is provided.

5.5.2.2 #define DEFAULT_TIMER 1000

Measured in microseconds.

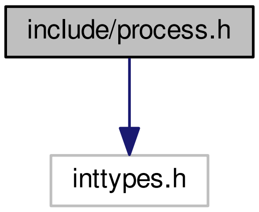
5.5.3 Variable Documentation

5.5.3.1 static const char initProcessName[] = "main.trace" `[static]`

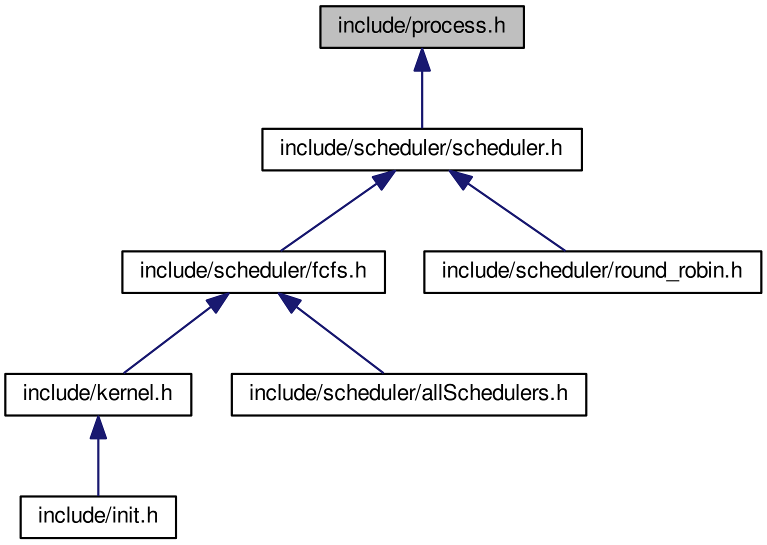
When the kernel object is created, this program is loaded. It is run once cKernel::boot is called.

5.6 include/process.h File Reference

`#include <inttypes.h>` Include dependency graph for process.h:



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


include/process.h
include/scheduler/scheduler.h
include/scheduler/fcfs.h
include/scheduler/round_robin.h
include/kernel.h
include/scheduler/allSchedulers.h
include/init.h

Classes

- struct ProcessInfo

  Structure for containing process state and data.

Typedefs

- typedef unsigned int **pidType**

Enumerations

- enum eProcState { ready, running, blocked, terminated }

  Enumeration for process states.

5.6.1 Detailed Description

5.6.2 Enumeration Type Documentation

5.6.2.1 enum eProcState

Each values defines a current state and possible transitions.

Enumerator:

ready Process is ready to be run. Invariant State:

- Kernel has initialized it at some point
- Process should be preparred to run

Potential Transitions:

- running - Scheduler picks it to run next

running Process is currently running. A running process should implicilty be considered ready. The kernel may not notify the scheduler to transition the process to ready before asking for a a scheduling decision. It is acceptable for the scheduler to make a process ready without the kernel's consent when it is being asked for a scheduling decision.

Invariant State:

- Process is on the cpu

Potential Transitions:

- blocked - Makes blocking system call
- terminated - Causes exception in cpu or finished normally

blocked Process is blocked and cannot run. Invariant State:

- Process is blocked (for now it can only block on I/O)

Potential Transitions:

- ready - Kernel notifies scheduler that I/O has finished

terminated Process has been terminated. It will be cleaned up soon.

Invariant State:

- Process either caused cpu exception or finished
- Process can no longer run

Potential Transitions:

- None