Freeagent Reference Manual

0.1a

Generated by Doxygen 1.2.8.1

Thu May 30 15:12:33 2002

Contents

Chapter 1

Freeagent Hierarchical Index

1.1 Freeagent Class Hierarchy

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

Chapter 2

Freeagent Compound Index

2.1 Freeagent Compound List

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

Chapter 3

Freeagent File Index

3.1 Freeagent File List

Here is a list of all files with brief descriptions:

Chapter 4

Freeagent Page Index

4.1 Freeagent Related Pages

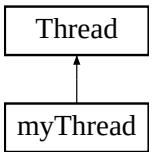
Here is a list of all related documentation pages:

Chapter 5

Freeagent Class Documentation

5.1 myThread Class Reference

Inheritance diagram for myThread::



Public Methods

- myThread ()
- ~myThread ()
- void Setup ()
- void Execute (void *arg)

5.1.1 Detailed Description

Test Thread-derived class

Definition at line 25 of file debug_test.cpp.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 myThread::myThread () [inline]

Default constructor calls Thread()

Definition at line 29 of file debug_test.cpp.

```
00030         {
00031             Thread();
00032         }
```

5.1.2.2 myThread::~myThread () [inline]

Default destructor calls pthread_exit()

Definition at line 34 of file debug_test.cpp.

```
00035         {
00036             pthread_exit(0);
00037         }
```

5.1.3 Member Function Documentation

5.1.3.1 void myThread::Execute (void * *arg*) [inline, virtual]

Execute prints out arg as if it were a character

Reimplemented from Thread.

Definition at line 44 of file debug_test.cpp.

```
00045         {
00046             {assert(arg != NULL);}
00047             char *t = (char *) arg;
00048             debug2_("argument passed to thread", *t, LOG_NO);//cout << *t << std::endl;
00049         }
```

5.1.3.2 void myThread::Setup () [inline, virtual]

Nothing to setup really

Reimplemented from Thread.

Definition at line 39 of file debug_test.cpp.

```
00040         {
00041         }
```

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

- tests/debug_test.cpp

5.2 netIDS Class Reference

`#include <ids.h>`

Public Methods

- netIDS ()
- ~netIDS ()
- int startSniffer (void *device)
- int stopSniffer ()
- int loadProtocolHandler (proHan *pro_handler)
- int unloadProtocolHandler (int pro)

Private Methods

- pcap_t* open (char *pcap_device)

Private Attributes

- unsigned int protocols_loaded
- pcap_t* pcap_structure
- char pcap_error [10000]
- void* protocol [ ]

5.2.1 Detailed Description

Class will handle the setup and execution of network sniffer and protocol handlers.

Todo:
Finish this implementation
Implement a way for protocol handlers to be pluggable
Add a way to keep track of all the protocol handlers
NONE OF THESE FUNCTIONS ARE IMPLEMENTED.

Definition at line 53 of file ids.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 netIDS::netIDS ()

5.2.2.2 netIDS::~netIDS ()

5.2.3 Member Function Documentation

5.2.3.1 int netIDS::loadProtocolHandler (proHan * *pro_handler*)

5.2.3.2 pcap_t * netIDS::open (char * *pcap_device*) [private]

5.2.3.3 int netIDS::startSniffer (void * *device*)

5.2.3.4 int netIDS::stopSniffer ()

5.2.3.5 int netIDS::unloadProtocolHandler (int *pro*)

5.2.4 Member Data Documentation

5.2.4.1 char netIDS::pcap_error [private]

last pcap error

Definition at line 70 of file ids.h.

5.2.4.2 pcap_t * netIDS::pcap_structure [private]

our pcap structure

Definition at line 68 of file ids.h.

5.2.4.3 void * netIDS::protocol [private]

array of protocol handler pointers?

Definition at line 72 of file ids.h.

5.2.4.4 unsigned int netIDS::protocols_loaded [private]

Number of protocol handlers loaded

Definition at line 66 of file ids.h.

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

- net/ids.h

5.3 proHan Struct Reference

`#include <prohandler.h>`

Public Attributes

- int protocol
- int* (* _Enqueue)(char *, int)
- void* (* _EntryPoint)(void *)

5.3.1 Detailed Description

This structure keeps track of a protocol handler's Enqueue and EntryPoint fucntions for the netIDS class to call them easily (base plugin support)

Definition at line 29 of file prohandler.h.

5.3.2 Member Data Documentation

5.3.2.1 int *(* proHan::_Enqueue)(char *, int)

Pointer to an enqueue function

5.3.2.2 void *(* proHan::_EntryPoint)(void *)

Pointer to an EntryPoint function

5.3.2.3 int proHan::protocol

protocol number

Definition at line 32 of file prohandler.h.

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

- net/prohandler.h

5.4 pronode_t Struct Reference

`#include <prohandler.h>`

Public Attributes

- pronode_t* prev
- char* payload
- pronode_t* next

5.4.1 Detailed Description

A basic linked list node structure

Definition at line 42 of file prohandler.h.

5.4.2 Member Data Documentation

5.4.2.1 pronode_t * pronode_t::next

Definition at line 46 of file prohandler.h.

5.4.2.2 char * pronode_t::payload

Definition at line 45 of file prohandler.h.

5.4.2.3 pronode_t * pronode_t::prev

Definition at line 44 of file prohandler.h.

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

- net/prohandler.h

5.5 protocolHandler Class Reference

`#include <prohandler.h>`

Public Methods

- protocolHandler ()
- virtual ~protocolHandler ()
- int add (char *pkt)
- virtual void threadEntryPoint (void *arg)

Protected Methods

- bool hasNext (void)
- char* next (void)

Protected Attributes

- int myPro

Private Attributes

- pronode_t* top
- pronode_t * bottom
- unsigned int packets

5.5.1 Detailed Description

Protocol Handling base class

This class must be inherited by any handler class you make

in order to work with the agent subsystem.

Test:
Test Protocol Handling subsystem
Threaded protocol handlers

Todo:
Support pluggable protocol handlers

Definition at line 60 of file prohandler.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 protocolHandler::protocolHandler ()

Default constructor Have any inherited classed call protocolHandler() before their constructor
Definition at line 32 of file prohandler.cpp.

```
00033 {
00034     //myPro = 0;
00035     packets = 0;
00036     top = bottom = NULL;
00037 }
```

5.5.2.2 protocolHandler::~protocolHandler () [virtual]

5.5.3 Member Function Documentation

5.5.3.1 int protocolHandler::add (char * *pkt*)

Basic packet enqueuing function Enqueues a packet pointed to by pkt into this protocolHandler's queue

Parameters:

pkt Pointer to packet

Todo:

Include a size variable to ensure we dont smash the stack

Returns:

Number of packets now in queue

Definition at line 48 of file prohandler.cpp.

```
00049 {
00050     {assert(pkt != NULL);}
00051     pronode_t *newNode;
00052     agent_new(&newNode);
00053     {assert(newNode != NULL);}
00054     newNode->payload = pkt;
00055     if (top == NULL)
00056         {
00057             top = newNode;
00058             bottom = top;
00059         }
00060         else //nope
00061         {
00062             //insert at end
00063             bottom->next = newNode;
00064             newNode->prev = bottom;
00065             bottom = newNode;
00066         }
00067         packets++;
00068     return packets;
00069 }
```

5.5.3.2 bool protocolHandler::hasNext (void) [protected]

Check if any packets are on the queue

Returns:

true if any packets are left to dequeue

Definition at line 75 of file prohandler.cpp.

Referenced by next().

```
00076 {
00077     return (packets > 0);
00078 }
```

5.5.3.3 char * protocolHandler::next (void) [protected]

Basic packet dequeuing function Dequeues a packet from this protocolHandler's queue

Todo:
Include a size variable to ensure we dont smash the stack

Returns:
Pointer the the packet pulled off the queue or NULL if it's empty

Definition at line 88 of file prohandler.cpp.

```
00089 {
00090     if(!hasNext())
00091         return NULL;
00092     //top is not NULL
00093     {assert(top != NULL);}
00094     char *data;
00095     data = top->payload;
00096     top->payload = NULL;
00097     if(top == bottom)
00098     {
00099         //delete it
00100         agent_zaparr(top);
00101         top = bottom = NULL;
00102     }
00103     else
00104     {
00105         pronode_t *t = top;
00106         top->next->prev = NULL;
00107         top = top->next;
00108         t->prev = t->next = NULL;
00109         agent_zaparr(t);
00110     }
00111     packets--;
00112     return data;
00113 }
```

5.5.3.4 void protocolHandler::threadEntryPoint (void * *arg*) [virtual]

5.5.4 Member Data Documentation

5.5.4.1 pronode_t * protocolHandler::bottom [private]

Pointer to bottom of queue

Definition at line 74 of file prohandler.h.

5.5.4.2 int protocolHandler::myPro [protected]

my protocol number

Definition at line 71 of file prohandler.h.

5.5.4.3 unsigned int protocolHandler::packets [private]

Number of packets on queue

Definition at line 78 of file prohandler.h.

5.5.4.4 pronode_t * protocolHandler::top [private]

Pointer to top of queue

Definition at line 74 of file prohandler.h.

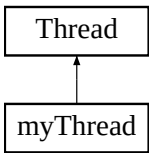
The documentation for this class was generated from the following files:

- net/prohandler.h
- net/prohandler.cpp

5.6 Thread Class Reference

`#include <thread.h>`

Inheritance diagram for Thread::



Public Methods

- Thread ()
- virtual ~Thread ()
- int Start (void *arg)

Protected Methods

- int Run (void *arg)
- virtual void Setup ()
- virtual void Execute (void *)
- void* Arg () const
- void Arg (void *a)
- void Mutex (pthread_mutex_t *a)
- pthread_mutex_t* Mutex () const
- virtual void Lock ()
- virtual void Unlock ()

Static Protected Methods

- void* EntryPoint (void *)

Private Attributes

- pthread_t a_thread
- void* Arg_
- pthread_mutex_t* p_mutex

5.6.1 Detailed Description

Thread support class

This class must be inherited by any thread class you make

in order to work with the agent subsystem.

Test:
Test threading subsystem and ensure Threads actually work
Mutex sharing

Todo:
Support more complex pthread handling

Definition at line 32 of file thread.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Thread::Thread ()

Default constructor If you overload this, Make sure your class calls Thread() to setup the mutex.

Definition at line 29 of file thread.cpp.

Referenced by myThread::myThread().

```
00030 {
00031     p_mutex = NULL;
00032     agent_new(&p_mutex);
00033     {assert(p_mutex != NULL);}
00034     pthread_mutex_init(p_mutex, NULL);
00035 }
```

5.6.2.2 Thread::~Thread () [virtual]

Default destructor Overload this, but include the following code in your destructor

Definition at line 42 of file thread.cpp.

```
00043 {
00044     {assert(p_mutex != NULL);}
00045     pthread_mutex_destroy(p_mutex);
00046 }
```

5.6.3 Member Function Documentation

5.6.3.1 void Thread::Arg (void * *a*) [inline, protected]

Set the current argument (casted to void*)

Parameters:
a A pointer to what will be passed to Run when your thread is Started

Definition at line 53 of file thread.h.

```
00053 {Arg_ = a;}
```

5.6.3.2 void * Thread::Arg () const [inline, protected]

Return the current argument casted to void*

Returns:
A pointer to thread's argument

Definition at line 47 of file thread.h.

Referenced by EntryPoint(), and Start().

```
00047 {return Arg_;}
```

5.6.3.3 void * Thread::EntryPoint (void * *pthis*) [static, protected]

Function called from Start Takes a Thread pointer as argument NOTE: Don't EVER CALL THIS. Start will call this

Parameters:
pthis Thread pointer casted to void *

Returns:
returns Thread pointer after Run was called

See also:
Setup , Execute

Definition at line 90 of file thread.cpp.

```
00091 {
00092    Thread * pt = (Thread*)pthis;
00093    pt->Run(pt->Arg());
00094    return pt;
00095 }
```

5.6.3.4 void Thread::Execute (void * *arg*) [protected, virtual]

Actual thread code Overload this in your thread, or it wont do anything

See also:
Run , Setup

Reimplemented in myThread.

Definition at line 114 of file thread.cpp.

Referenced by Run().

```
00115 {
00116         // Your code goes here
00117 }
```

5.6.3.5 void Thread::Lock () [protected, virtual]

Lock the mutex associated with this thread

See also:
Unlock

Definition at line 123 of file thread.cpp.

```
00124 {
00125     pthread_mutex_lock(p_mutex);
00126 }
```

5.6.3.6 pthread_mutex_t * Thread::Mutex () const [inline, protected]

Return the current mutex (for mutex sharing)

Returns:
A pointer to thread's mutex

Definition at line 63 of file thread.h.

```
00063 {return p_mutex;}
```

5.6.3.7 void Thread::Mutex (pthread_mutex_t * *a*) [inline, protected]

Set the mutex (for mutex sharing)

Parameters:
a Pointer to your pthread_mutex_t

Definition at line 58 of file thread.h.

```
00058 {p_mutex = a;}
```

5.6.3.8 int Thread::Run (void * *arg*) [protected]

Default entrypoint for threads Calls Setyp and Execute(arg)

Parameters:
arg Argument to pass to thread

Returns:
returns 0

See also:
Setup , Execute

Definition at line 72 of file thread.cpp.

Referenced by EntryPoint().

```
00073 {
00074    Setup();
00075    Execute(arg);
00076    return 0;
00077 }
```

5.6.3.9 void Thread::Setup () [protected, virtual]

Handle the setup of your thread Overload this in your thread, or it wont do anything

See also:
Run , Execute

Reimplemented in myThread.

Definition at line 103 of file thread.cpp.

Referenced by Run().

```
00104 {
00105         // Do any setup here
00106 }
```

5.6.3.10 int Thread::Start (void * *arg*)

Start the thread running in its own context Pass any argument you want as a void *, but make sure your thread knows how to handle it.

Parameters:
arg Argument to pass to thread

Returns:
returns the value pthread_create returned

See also:
EntryPoint

Definition at line 56 of file thread.cpp.

Referenced by main().

```
00057 {
00058    Arg(arg); // store user data
00059    int code = pthread_create(&a_thread, NULL, (void *(*) (void *))
00060         Thread::EntryPoint, (void *) this);
00061    return code;
00062 }
```

5.6.3.11 void Thread::Unlock () [protected, virtual]

Unlock the mutex associated with this thread

See also:
Lock

Definition at line 132 of file thread.cpp.

```
00133 {
00134     pthread_mutex_unlock(p_mutex);
00135 }
```

5.6.4 Member Data Documentation

5.6.4.1 void * Thread::Arg_ [private]

Argument storage

Definition at line 70 of file thread.h.

5.6.4.2 pthread_t Thread::a_thread [private]

Definition at line 67 of file thread.h.

5.6.4.3 pthread_mutex_t * Thread::p_mutex [private]

Mutex for locking

Definition at line 72 of file thread.h.

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

- core/thread.h
- core/thread.cpp

Chapter 6

Freeagent File Documentation

6.1 con/print.cpp File Reference

```
#include "print.h"
```

Functions

- void agent_print (const T &var)
- void agent_println (const T &var)

Variables

- const char prntid [ ] = "\100$ Free Agent: print.cpp, v 0.1 2002/05/29 red0x Exp $"

6.1.1 Detailed Description

File contains print functions

Include con/print.cpp when you need

console printing functions

DO NOT INCLUDE PRINT.H, IT WONT LINK

PRPOPERLY. INCLUDE PRINT.CPP ONLY!

Definition in file print.cpp.

6.1.2 Function Documentation

6.1.2.1 void agent_print (const T & *var*)

(Template function) Function prints output to the console

Parameters:

var Variable to print

Definition at line 33 of file print.cpp.

```
00034 {
00035     std::cout << var;
00036 }
```

6.1.2.2 void agent_println (const T & *var*)

(Template function)

Function prints output to the console followed by newline

Parameters:

var Variable to print

Definition at line 45 of file print.cpp.

```
00046 {
00047     std::cout << var << std::endl;
00048 }
```

6.1.3 Variable Documentation

6.1.3.1 const char prntid = "\100$ Free Agent: print.cpp, v 0.1 2002/05/29 red0x Exp $" [static]

Definition at line 24 of file print.cpp.

6.2 con/print.h File Reference

```
#include <iostream.h>
```

Functions

- void agent_print (const T &var)
- void agent_println (const T &var)

6.2.1 Detailed Description

File contains print functions

Definition in file print.h.

6.2.2 Function Documentation

6.2.2.1 void agent_print (const T & *var*)

Definition at line 33 of file print.cpp.

Referenced by debug_agent().

```
00034 {
00035     std::cout << var;
00036 }
```

6.2.2.2 void agent_println (const T & *var*)

Definition at line 45 of file print.cpp.

Referenced by debug_agent(), and main().

```
00046 {
00047     std::cout << var << std::endl;
00048 }
```

6.3 core/debug.cpp File Reference

```
#include <fstream.h>
#include <time.h>
#include <sys/time.h>
#include "debug.h"
#include "con/print.cpp"
```

Functions

- void debug_agent (char *name, T &value, bool logfile=false)

Variables

- const char dbgid [ ] = "\100$ Free Agent: debug.cpp, v 0.1 2002/05/29 red0x Exp $"

6.3.1 Detailed Description

Debugging Subsystem

Include core/debug.cpp when you need

debugging susbsytem functions

DO NOT INCLUDE DEBUG.H, IT WONT LINK

PRPOPERLY. INCLUDE DEBUG.CPP ONLY!

Definition in file debug.cpp.

6.3.2 Function Documentation

6.3.2.1 void debug_agent (char * *name*, T & *value*, bool *logfile* = false)

Debugging output (template function)

This will output the value in "value",

next to the name in "name", along with a time and date.

If logfile is true (LOG_YES), it will output to the default

debug log file (/var/tmp/agent.log)

Parameters:

name Name to give the value

value Value to print

logfile Set to LOG_YES to output to logfile also.

Definition at line 41 of file debug.cpp.

```
00042 {
00043     {assert(name != NULL);}
```

```
00044     //{assert(value != NULL);}
00045     time_t tm;
00046     char *stm;
00047     agent_newarr(&stm, 50);
00048     time(&tm);
00049     strftime(stm, 50, "%a %b %e %T %Z %Y", localtime(&tm));
00050     if(logfile)
00051     {
00052 
00053         fstream fp;
00054         fp.open(debug_agent_log_file,ios::out|ios::app);
00055         if(!fp.is_open())
00056             return;
00057 
00058         fp << "[" << stm << "] " << name << ": " << value << ";" << std::endl;
00059         fp.close();
00060         //output name = value;\n
00061     }
00062     //what to do if we dont want to log it...
00063     //print it?
00064     agent_print("[");
00065     agent_print(stm);
00066     agent_print("] ");
00067     agent_print(name);
00068     agent_print(": ");
00069     agent_print(value);
00070     agent_println(";");
00071 }
```

6.3.3 Variable Documentation

6.3.3.1 const char dbgid = "\100$ Free Agent: debug.cpp, v 0.1 2002/05/29 red0x Exp $" [static]

Definition at line 27 of file debug.cpp.

6.4 core/debug.h File Reference

```
#include <assert.h>
#include <sys/types.h>
```

Defines

- #define DEBUG
- #define debug_(NM, VL) debug_agent(NM, VL, LOG_NO)
- #define debug2_(NM, VL, LOG_FILE) debug_agent(NM, VL, LOG_FILE)

Functions

- void debug_agent (char name[ ], T &value, bool logfile=false)

Variables

- const char debug_agent_log_file [ ] = ”/var/tmp/agent.log\0\0”
- const bool LOG_YES = true
- const bool LOG_NO = false

6.4.1 Detailed Description

Debugging Subsystem

Todo:
Move the DEBUG define into toplevel config.h

Definition in file debug.h.

6.4.2 Define Documentation

6.4.2.1 #define DEBUG

Value:

Definition at line 18 of file debug.h.

6.4.2.2 #define debug2_(NM, VL, LOG_FILE) debug_agent(NM, VL, LOG_FILE)

Definition at line 31 of file debug.h.

Referenced by myThread::Execute(), and main().

6.4.2.3 #define debug_(NM, VL) debug_agent(NM, VL, LOG_NO)

Definition at line 30 of file debug.h.

6.4.3 Function Documentation

6.4.3.1 void debug_agent (char *name*[ ], T & *value*, bool *logfile* = false)

6.4.4 Variable Documentation

6.4.4.1 const bool LOG_NO = false

Definition at line 28 of file debug.h.

6.4.4.2 const bool LOG_YES = true

Definition at line 27 of file debug.h.

6.4.4.3 const char debug_agent_log_file = "/var/tmp/agent.log\0\0"

Definition at line 26 of file debug.h.

6.5 core/thread.cpp File Reference

```
#include "thread.h"
#include "mem/memory.cpp"
```

Variables

- const char thdid [ ] = ”\100$ Free Agent: thread.cpp, v 0.1 2002/05/29 red0x Exp $”

6.5.1 Detailed Description

Core threading support class

To make a new thread, build a class that inherits Thread as public. Implement your own Execute and Setup functions To do what your thread should.

Definition in file thread.cpp.

6.5.2 Variable Documentation

6.5.2.1 const char thdid = ”\100$ Free Agent: thread.cpp, v 0.1 2002/05/29 red0x Exp $” [static]

Definition at line 22 of file thread.cpp.

6.6 core/thread.h File Reference

```
#include <assert.h>
#include <pthread.h>
```

Compounds

- class Thread

6.6.1 Detailed Description

Core threading support class

To make a new thread, build a class that inherits Thread as public. Implement your own Execute and Setup functions To do what your thread should.

Definition in file thread.h.

6.7 mem/alloc.cpp File Reference

```
#include "alloc.h"
```

Functions

- T* agent_malloc (T **aa, size_t size)
- T* agent_realloc (T **aa, size_t size)
- void agent_free (void *ptr)

Variables

- const char alcid [ ] = "\100$ Free Agent: alloc.cpp, v 0.1 2002/05/29 red0x Exp $"

6.7.1 Detailed Description

C-Style memory handling wrappers INCLUDE mem/alloc.cpp ONLY. DO NOT INCLUDE ALLOC.H, IT WONT LINK CORRECTLY

Definition in file alloc.cpp.

6.7.2 Function Documentation

6.7.2.1 void agent_free (void * *ptr*)

Free memory This function does C-Style memory de-allocation (free)

Parameters:

ptr Pointer to destination memory

Definition at line 78 of file alloc.cpp.

```
00079 {
00080     {assert(ptr != NULL);}
00081     free(ptr);
00082     ptr = NULL;
00083 }
```

6.7.2.2 T * agent_malloc (T ** *arg*, size_t *size*)

Allocate memory (Template function) This function does C-Style memory allocation (malloc) and puts it in to a pointer.

Parameters:

aa Double pointer to destination memory

size Amount of memory to allocate

Returns:

Pointer to memory

Definition at line 34 of file alloc.cpp.

```
00035 {
00036     {assert(size > 0);}
00037     size_t tmpii = size + SAFE_MEM;
00038     *aa = NULL;
00039     *aa = (T *) malloc(tmpii);
00040     if (*aa == NULL)
00041         return NULL;//exit(-1);
00042     memset(*aa, 0, tmpii);
00043     return *aa;
00044 }
```

6.7.2.3 T * agent_realloc (T ** *aa*, size_t *size*)

Reallocate memory (Template Function) This function does C-Style memory reallocation (realloc) and puts it in to a pointer.

Parameters:

aa Double pointer to destination memory

size Amount of memory to allocate

Returns:

Pointer to memory

Definition at line 56 of file alloc.cpp.

```
00057 {
00058     {assert(size > 0);}
00059     {assert(aa != NULL);}
00060     unsigned long kk, tmpqq = size + SAFE_MEM;
00061     size_t tmpii = sizeof(T) * (tmpqq);
00062     *aa = (T *) realloc(*aa, tmpii);
00063     if (*aa == NULL)
00064         return NULL;//exit(-1);
00065     // do not memset memset(aa, 0, tmpii);
00066     *aa[tmpqq-1] = 0x00;  //NULL terminator...
00067     for (kk = size; kk < tmpqq; kk++)
00068         *aa[kk] = 0;  // a few NULL terminators, for safety
00069     return *aa;
00070 }
```

6.7.3 Variable Documentation

6.7.3.1 const char alcid = "\100$ Free Agent: alloc.cpp, v 0.1 2002/05/29 red0x Exp $" [static]

Definition at line 22 of file alloc.cpp.

6.8 mem/alloc.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <malloc.h>
#include <assert.h>
#include <sys/types.h>
```

Defines

- #define SAFE_MEM 5

Functions

- T* agent_malloc (T **arg, size_t size)
- T* agent_realloc (T **aa, size_t size)
- void agent_free (void *ptr)

6.8.1 Detailed Description

C-Style memory handling wrappers

INCLUDE mem/alloc.cpp ONLY. DO NOT INCLUDE ALLOC.H, IT WONT LINK CORRECTLY

Todo:
Move SAFE_MEM to toplevel config.h

Definition in file alloc.h.

6.8.2 Define Documentation

6.8.2.1 #define SAFE_MEM 5

Definition at line 27 of file alloc.h.

6.8.3 Function Documentation

6.8.3.1 void agent_free (void * *ptr*)

Definition at line 78 of file alloc.cpp.

Referenced by main().

```
00079 {
00080     {assert(ptr != NULL);}
00081     free(ptr);
00082     ptr = NULL;
00083 }
```

6.8.3.2 T* agent_malloc (T ** *arg*, size_t *size*)

Definition at line 34 of file alloc.cpp.

Referenced by main().

```
00035 {
00036     {assert(size > 0);}
00037     size_t tmpii = size + SAFE_MEM;
00038     *aa = NULL;
00039     *aa = (T *) malloc(tmpii);
00040     if (*aa == NULL)
00041         return NULL;//exit(-1);
00042     memset(*aa, 0, tmpii);
00043     return *aa;
00044 }
```

6.8.3.3 T* agent_realloc (T ** *aa*, size_t *size*)

Definition at line 56 of file alloc.cpp.

```
00057 {
00058     {assert(size > 0);}
00059     {assert(aa != NULL);}
00060     unsigned long kk, tmpqq = size + SAFE_MEM;
00061     size_t tmpii = sizeof(T) * (tmpqq);
00062     *aa = (T *) realloc(*aa, tmpii);
00063     if (*aa == NULL)
00064         return NULL;//exit(-1);
00065     // do not memset memset(aa, 0, tmpii);
00066     *aa[tmpqq-1] = 0x00;  //NULL terminator...
00067     for (kk = size; kk < tmpqq; kk++)
00068         *aa[kk] = 0;  // a few NULL terminators, for safety
00069     return *aa;
00070 }
```

6.9 mem/memory.cpp File Reference

```
#include "memory.h"
```

Functions

- void agent_zap (T &x)
- void agent_zaparr (T &x)
- T* agent_newarr (T **dest, size_t size)
- T* agent_new (T **dest)

Variables

- const char memid [ ] = "\100$ Free Agent: memory.cpp, v 0.1 2002/05/29 red0x Exp $"

6.9.1 Detailed Description

C++ Memory Hanlind Wrappers

Include mem/memory.cpp when you need

C++ memory handling functions

DO NOT INCLUDE MEMORY.H, IT WONT LINK

PRPOPERLY.

Definition in file memory.cpp.

6.9.2 Function Documentation

6.9.2.1 T * agent_new (T ** *dest*)

Allocate new memory (Template Function)

Parameters:
dest Double pointer to destination memory

Returns:
Pointer to memory

Definition at line 79 of file memory.cpp.

```
00080 {
00081     *dest = new T;
00082     return *dest;
00083 }
```

6.9.2.2 T * agent_newarr (T ** *dest*, size_t *size*)

Allocate a new array (Template Function)

Parameters:

dest Double pointer to destination memory

size Size of array

Returns:

Pointer to memory

Definition at line 61 of file memory.cpp.

```
00062 {
00063     {assert(size > 0);}
00064     //add a pad, for security
00065     size_t tmpii = size + SAFE_MEM;
00066     *dest = new T[tmpii];
00067     //clear it
00068     memset(*dest, 0, tmpii);
00069     return *dest;
00070 }
```

6.9.2.3 void agent_zap (T & *x*) [inline]

Delete memory (Template Function)

Parameters:

x Reference to memory to delete

Definition at line 33 of file memory.cpp.

```
00034 {
00035     {assert(x != NULL);}
00036     delete x;
00037     x = NULL;
00038 }
```

6.9.2.4 void agent_zaparr (T & *x*) [inline]

Delete an array (Template Function)

Parameters:

x Reference to array to delete

Definition at line 46 of file memory.cpp.

```
00047 {
00048     {assert(x != NULL);}
00049     delete [] x;
00050     x = NULL;
00051 }
```

6.9.3 Variable Documentation

6.9.3.1 const char memid = "\100$ Free Agent: memory.cpp, v 0.1 2002/05/29 red0x Exp $" [static]

Definition at line 25 of file memory.cpp.

6.10 mem/memory.h File Reference

```
#include <assert.h>
#include <sys/types.h>
```

Defines

- #define SAFE_MEM 5

Functions

- void agent_zap (T &x)
- void agent_zaparr (T &x)
- T* agent_new (T **dest)
- T* agent_newarr (T **dest, size_t size)

6.10.1 Detailed Description

C++ Memory Hanlind Wrappers

Include mem/memory.cpp when you need

C++ memory handling functions

DO NOT INCLUDE MEMORY.H, IT WONT LINK

PRPOPERLY.

Todo:
Move SAFE_MEM to toplevel config.h

Definition in file memory.h.

6.10.2 Define Documentation

6.10.2.1 #define SAFE_MEM 5

Definition at line 25 of file memory.h.

6.10.3 Function Documentation

6.10.3.1 T* agent_new (T ** *dest*)

Definition at line 79 of file memory.cpp.

Referenced by Thread::Thread(), protocolHandler::add(), and main().

```
00080 {
00081     *dest = new T;
00082     return *dest;
00083 }
```

6.10.3.2 T* agent_newarr (T ** *dest*, size_t *size*)

Definition at line 61 of file memory.cpp.

Referenced by debug_agent().

```
00062 {
00063     {assert(size > 0);}
00064     //add a pad, for security
00065     size_t tmpii = size + SAFE_MEM;
00066     *dest = new T[tmpii];
00067     //clear it
00068     memset(*dest, 0, tmpii);
00069     return *dest;
00070 }
```

6.10.3.3 void agent_zap (T & *x*) [inline]

Definition at line 33 of file memory.cpp.

Referenced by main().

```
00034 {
00035     {assert(x != NULL);}
00036     delete x;
00037     x = NULL;
00038 }
```

6.10.3.4 void agent_zaparr (T & *x*) [inline]

Definition at line 46 of file memory.cpp.

Referenced by protocolHandler::next().

```
00047 {
00048     {assert(x != NULL);}
00049     delete [] x;
00050     x = NULL;
00051 }
```

6.11 net/ids.cpp File Reference

```
#include "ids.h"
#include "con/print.cpp"
```

Defines

- #define DEBUG

Variables

- const char idsid [ ] = ”\100$ Free Agent: ids.cpp, v 0.1 2002/05/29 red0x Exp $”
- unsigned long int icmp = 0
- unsigned long int udp = 0
- unsigned long int tcp = 0
- unsigned long int ip = 0
- unsigned long int max_icmp = 0
- unsigned long int max_udp = 0
- unsigned long int max_tcp = 0
- unsigned long int max_ip = 0

6.11.1 Detailed Description

Network IDS subsystem (sniffer)

Todo:

Add a startup function to setup a libpcap sniffer. Add a function to get a list of protocol handlers and Start them running. Then, start capturing packets and passing them off to the protocol handlers.
Add a startup function to setup a libpcap sniffer. Add a function to get a list of protocol handlers and Start them running. Then, start capturing packets and passing them off to the protocol handlers.
Move DEBUG define to toplevel config.h

Definition in file ids.cpp.

6.11.2 Define Documentation

6.11.2.1 #define DEBUG

Value:

Definition at line 29 of file ids.cpp.

6.11.3 Variable Documentation

6.11.3.1 unsigned long int icmp = 0

Definition at line 35 of file ids.cpp.

6.11.3.2 const char idsid = "\100$ Free Agent: ids.cpp, v 0.1 2002/05/29 red0x Exp $" [static]

Definition at line 32 of file ids.cpp.

6.11.3.3 unsigned long int ip = 0

Definition at line 35 of file ids.cpp.

6.11.3.4 unsigned long int max_icmp = 0

Definition at line 36 of file ids.cpp.

6.11.3.5 unsigned long int max_ip = 0

Definition at line 36 of file ids.cpp.

6.11.3.6 unsigned long int max_tcp = 0

Definition at line 36 of file ids.cpp.

6.11.3.7 unsigned long int max_udp = 0

Definition at line 36 of file ids.cpp.

6.11.3.8 unsigned long int tcp = 0

Definition at line 35 of file ids.cpp.

6.11.3.9 unsigned long int udp = 0

Definition at line 35 of file ids.cpp.

6.12 net/ids.h File Reference

```
#include <pcap.h>
#include "prohandler.h"
```

Compounds

- class netIDS

Defines

- #define IFACE ”eth0”
- #define SNAPLEN 1600
- #define PROMISC 0
- #define TIMEOUT 1000
- #define COUNT 50

6.12.1 Detailed Description

Network IDS subsystem (sniffer)

Todo:
Add a startup function to setup a libpcap sniffer. Add a function to get a list of protocol handlers and Start them running. Then, start capturing packets and passing them off to the protocol handlers.
Move DEBUG define to toplevel config.h
Move IFACE, SNAPLEN, PROMISC, TIMEOUT, and COUNT to toplevel config.h

Bug:
Had to workaround libpcap’s pcap.h linkage mangler.

Test:
Make sure libpcap still works

Definition in file ids.h.

6.12.2 Define Documentation

6.12.2.1 #define COUNT 50

Definition at line 39 of file ids.h.

6.12.2.2 #define IFACE ”eth0”

Definition at line 34 of file ids.h.

6.12.2.3 #define PROMISC 0

Definition at line 37 of file ids.h.

6.12.2.4 #define SNAPLEN 1600

Definition at line 35 of file ids.h.

6.12.2.5 #define TIMEOUT 1000

Definition at line 38 of file ids.h.

6.13 net/prohandler.cpp File Reference

```
#include <assert.h>
#include <stdio.h>
#include "prohandler.h"
#include "mem/memory.cpp"
```

Variables

- const char phdid [ ] = ”\100$ Free Agent: prohandler.cpp, v 0.1 2002/05/29 red0x Exp $”

6.13.1 Detailed Description

Protocol Handler Base class

To make a new handler, build a class that inherits protocolHandler as public.

Todo:
Actually make the protocolHandler work

Test:
Inheriting a protocolHandler to extend it

Definition in file prohandler.cpp.

6.13.2 Variable Documentation

6.13.2.1 const char phdid = ”\100$ Free Agent: prohandler.cpp, v 0.1 2002/05/29 red0x Exp $” [static]

Definition at line 25 of file prohandler.cpp.

6.14 net/prohandler.h File Reference

Compounds

- struct proHan
- struct pronode_t
- class protocolHandler

Defines

- #define MKPROHANDLER(x, y, pro)

6.14.1 Detailed Description

Protocol Handler Base class

To make a new handler, build a class that inherits protocolHandler as public.

Definition in file prohandler.h.

6.14.2 Define Documentation

6.14.2.1 #define MKPROHANDLER(x, y, pro)

Value:

```
proHan x; x.protocol = pro; x._Enqueue = y->add; \
x._EntryPoint = y->threadEntrypPoint;
```

make a new proHan structure named x, fill it with y's enqueuing and entrypoint functions

Definition at line 82 of file prohandler.h.

6.15 tests/debug_test.cpp File Reference

```
#include <pthread.h>
#include <unistd.h>
#include <assert.h>
#include "core/debug.cpp"
#include "core/thread.cpp"
#include "mem/memory.cpp"
#include "mem/alloc.cpp"
```

Compounds

- class myThread

Functions

- int main (void)

6.15.1 Function Documentation

6.15.1.1 int main (void)

Definition at line 52 of file debug_test.cpp.

```
00053 {
00054     myThread *th_test;
00055     int i = 2;
00056     char *j;
00057     agent_new(&j);
00058     *j = 'a';
00059     debug2_("i", i, LOG_NO);
00060     debug2_("j", *j, LOG_NO);
00061     agent_zap(j);
00062     agent_malloc(&j, sizeof(char));
00063     *j = 'b';
00064     debug2_("j after malloc", *j, LOG_NO);
00065
00066     agent_println("creating thread");
00067     agent_new(&th_test);
00068     th_test->Start(j);
00069     agent_println("destroying thread");
00070     agent_free((void *)j);
00071     agent_zap(th_test);
00072     return 0;
00073 }
```

Chapter 7

Freeagent Page Documentation

7.1 Todo List

Class netIDS Finish this implementation

Implement a way for protocol handlers to be pluggable

Add a way to keep track of all the protocol handlers

NONE OF THESE FUNCTIONS ARE IMPLEMENTED.

Class protocolHandler Support pluggable protocol handlers

Class Thread Support more complex pthread handling

File debug.h Move the DEBUG define into toplevel config.h

File alloc.h Move SAFE_MEM to toplevel config.h

File memory.h Move SAFE_MEM to toplevel config.h

File ids.cpp Add a startup function to setup a libpcap sniffer. Add a function to get a list of protocol handlers and Start them running. Then, start capturing packets and passing them off to the protocol handlers.

Add a startup function to setup a libpcap sniffer. Add a function to get a list of protocol handlers and Start them running. Then, start capturing packets and passing them off to the protocol handlers.

Move DEBUG define to toplevel config.h

File ids.h Add a startup function to setup a libpcap sniffer. Add a function to get a list of protocol handlers and Start them running. Then, start capturing packets and passing them off to the protocol handlers.

Move DEBUG define to toplevel config.h

Move IFACE, SNAPLEN, PROMISC, TIMEOUT, and COUNT to toplevel config.h

File prohandler.cpp Actually make the protocolHandler work

Member protocolHandler::add(char *pkt) Include a size variable to ensure we dont smash the stack

Member protocolHandler::next(void) Include a size variable to ensure we dont smash the stack

7.2 Test List

Class protocolHandler Test Protocol Handling subsystem

Threaded protocol handlers

Class Thread Test threading subsystem and ensure Threads actually work

Mutex sharing

File ids.h Make sure libpcap still works

File prohandler.cpp Inheriting a protocolHandler to extend it

7.3 Bug List

File ids.h Had to workaround libpcap's pcap.h linkage mangler.

Index