mainloop Reference Manual

Generated by Doxygen 1.3.7

Mon Jul 19 16:02:06 2004

Contents

Chapter 1

mainloop Main Page

Most programs are built around some variation of a main loop. In it’s simplest form, this looks something like

```
 while ((len = read(fdin, buf, sizeof(buf)) != -1) {
...
write(fdout, buf, len);
 }
```

As the program evolves, non-blocking (or even async) I/O, timers, and signal handling are added. Soon, you either add some sort of generic framework, or you end up with an unmaintainable mess.

And even though (with some experience) you can write such a framework rather quickly, it still takes a considerable amount of time. The result usually lacks both uniformity (it is rewritten every time) and generality (it is often "optimized" for a specific case).

If you know your program is going to grow beyond the simple case outlined above, it pays of to use a generic framework like glib. This is not meant to imply that the glib implementation is better than what you would write instead (IMHO glib is tied too much to the structure of the poll() syscall and can’t be adapted to take advantage of other mechanisms like /dev/poll, epoll or kqueue). But it is good enough, and above all, it’s widely used and documented.

An additional advantage of using a standardized framework is that it (at least theoretically, because it only works if everyone uses the same framework) solves the problem of competing signal handlers in libraries.

The following (simple minded) code metric shows that while you may feel that you have to write a lot of code just to accomodate the glib framework, you actually write less (because you don’t have to write the framework itself). In any case it shows that the difference is not big enough to be used as an argument.

```
 Stmts Comnts  Funcs  Blanks Lines
   195    109      8      46    550 bad/mainloop_bad.c
   245    163     24      62    772 good/mainloop_good.c
   226    132     20      84    715 glib/mainloop_glib.c
```

Doxygen cannot cope with multiply defined (local) symbols. Local symbols having the same name are documented only once, with all links pointing to the same file.

Chapter 2

mainloop Data Structure Index

2.1 mainloop Data Structures

Here are the data structures with brief descriptions:

Chapter 3

mainloop File Index

3.1 mainloop File List

Here is a list of all files with brief descriptions:

Chapter 4

mainloop Page Index

4.1 mainloop Related Pages

Here is a list of all related documentation pages:

Chapter 5

mainloop Data Structure Documentation

5.1 alarm_t Struct Reference

alarm handler state

Data Fields

- alarmhandler_t handler
  alarm handler
- long interval
  interval in seconds
- long nexttime
  next execution at
- int flag
  1 if handler should be called

5.1.1 Detailed Description

alarm handler state

handler, interval, next execution, pending. no handler specific state...

Definition at line 114 of file mainloop_good.c.

5.1.2 Field Documentation

5.1.2.1 int alarm_t::flag

1 if handler should be called

Definition at line 118 of file mainloop_good.c.

Referenced by addalarm(), alarmhandler(), and checkalarms().

5.1.2.2 alarmhandler_t alarm_t::handler

alarm handler

Definition at line 115 of file mainloop_good.c.

Referenced by addalarm(), alarmhandler(), checkalarms(), and initalarms().

5.1.2.3 long alarm_t::interval

interval in seconds

Definition at line 116 of file mainloop_good.c.

Referenced by addalarm(), and alarmhandler().

5.1.2.4 long alarm_t::nexttime

next execution at

Definition at line 117 of file mainloop_good.c.

Referenced by addalarm(), and alarmhandler().

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

- mainloop_good.c

5.2 chargen_client_t Struct Reference

chargen client specific state

Data Fields

- int i

  index into chargen_buf

5.2.1 Detailed Description

chargen client specific state

Definition at line 75 of file mainloop_good.c.

5.2.2 Field Documentation

5.2.2.1 int chargen_client_t::i

index into chargen_buf

Definition at line 76 of file mainloop_good.c.

Referenced by writechargen().

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

- mainloop_good.c

5.3 chargen_source_t Struct Reference

chargen source state

Data Fields

- int i

  index into chargen_buf

5.3.1 Detailed Description

chargen source state

Definition at line 73 of file mainloop_glib.c.

5.3.2 Field Documentation

5.3.2.1 int chargen_source_t::i

index into chargen_buf

Definition at line 74 of file mainloop_glib.c.

Referenced by chargen_dispatch(), and getchargenclientsource().

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

- mainloop_glib.c

5.4 chargenclient_t Struct Reference

Data Fields

- int fd
  socket
- int i
  index into chargen_buf

5.4.1 Field Documentation

5.4.1.1 int chargenclient_t::fd

socket

Definition at line 76 of file mainloop_bad.c.

Referenced by main(), and writechargen().

5.4.1.2 int chargenclient_t::i

index into chargen_buf

Definition at line 77 of file mainloop_bad.c.

Referenced by main(), and writechargen().

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

- mainloop_bad.c

5.5 client_t Struct Reference

client state

Data Fields

- int fd
  filedescriptor for poll()
- eventhandler_t read
  read handler
- eventhandler_t write
  write handler
- eventhandler_t except
  exception (eg disconnect) handler

5.5.1 Detailed Description

client state

in a real program, a pointer to a type specific data area would be preferrable to a union.

Definition at line 92 of file mainloop_good.c.

5.5.2 Field Documentation

5.5.2.1 chargen_client_t client_t::chargen

chargen client state

Definition at line 99 of file mainloop_good.c.

Referenced by writechargen().

5.5.2.2 echo_client_t client_t::echo

echo client state

Definition at line 98 of file mainloop_good.c.

Referenced by flowecho(), readecho(), and writeecho().

5.5.2.3 eventhandler_t client_t::except

exception (eg disconnect) handler

Definition at line 96 of file mainloop_good.c.

Referenced by addclient(), and mainloop().

5.5.2.4 int client_t::fd

filedescriptor for poll()

Definition at line 93 of file mainloop_good.c.

Referenced by addclient(), closeclient(), delclient(), and initclients().

5.5.2.5 eventhandler_t client_t::read

read handler

Definition at line 94 of file mainloop_good.c.

Referenced by addclient(), and mainloop().

5.5.2.6 eventhandler_t client_t::write

write handler

Definition at line 95 of file mainloop_good.c.

Referenced by addclient(), and mainloop().

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

- mainloop_good.c

5.6 echo_client_t Struct Reference

echo client specific state

Data Fields

- int r

  read position

- int w

  write position

- int n

  number of bytes

- char buf [BUFSIZE]

  buffer

5.6.1 Detailed Description

echo client specific state

Definition at line 65 of file mainloop_good.c.

5.6.2 Field Documentation

5.6.2.1 char echo_client_t::buf[BUFSIZE]

buffer

Definition at line 69 of file mainloop_good.c.

5.6.2.2 int echo_client_t::n

number of bytes

Definition at line 68 of file mainloop_good.c.

Referenced by flowecho(), readecho(), and writeecho().

5.6.2.3 int echo_client_t::r

read position

Definition at line 66 of file mainloop_good.c.

Referenced by readecho(), and writeecho().

5.6.2.4 int echo_client_t::w

write position

Definition at line 67 of file mainloop_good.c.

Referenced by readecho(), and writeecho().

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

- mainloop_good.c

5.7 echo_source_t Struct Reference

echo source state

Data Fields

- int r
  read position
- int w
  write position
- int n
  number of bytes
- char buf [BUFSIZE]
  buffer

5.7.1 Detailed Description

echo source state

Definition at line 63 of file mainloop_glib.c.

5.7.2 Field Documentation

5.7.2.1 char echo_source_t::buf[BUFSIZE]

buffer

Definition at line 67 of file mainloop_glib.c.

Referenced by echo_dispatch_read(), and echo_dispatch_write().

5.7.2.2 int echo_source_t::n

number of bytes

Definition at line 66 of file mainloop_glib.c.

Referenced by echo_dispatch_read(), echo_dispatch_write(), echo_prepare(), and getechoclientsource().

5.7.2.3 int echo_source_t::r

read position

Definition at line 64 of file mainloop_glib.c.

Referenced by echo_dispatch_read(), echo_dispatch_write(), and getechoclientsource().

5.7.2.4 int echo_source_t::w

write position

Definition at line 65 of file mainloop_glib.c.

Referenced by echo_dispatch_read(), echo_dispatch_write(), and getechoclientsource().

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

- mainloop_glib.c

5.8 echoclient_t Struct Reference

echo client state

Data Fields

- int fd
  socket
- int r
  read position
- int w
  write position
- int n
  number of bytes
- char buf [BUFSIZE]
  buffer

5.8.1 Detailed Description

echo client state

Definition at line 67 of file mainloop_bad.c.

5.8.2 Field Documentation

5.8.2.1 char echoclient_t::buf[BUFSIZE]

buffer

Definition at line 72 of file mainloop_bad.c.

Referenced by readecho(), and writeecho().

5.8.2.2 int echoclient_t::fd

socket

Definition at line 68 of file mainloop_bad.c.

Referenced by main(), readecho(), and writeecho().

5.8.2.3 int echoclient_t::n

number of bytes

Definition at line 71 of file mainloop_bad.c.

Referenced by main(), readecho(), and writeecho().

5.8.2.4 int echoclient_t::r

read position

Definition at line 69 of file mainloop_bad.c.

Referenced by main(), readecho(), and writeecho().

5.8.2.5 int echoclient_t::w

write position

Definition at line 70 of file mainloop_bad.c.

Referenced by main(), readecho(), and writeecho().

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

- mainloop_bad.c

5.9 listen_source_t Struct Reference

listen source state

Data Fields

- getclientsourcefunc getclientsource

5.9.1 Detailed Description

listen source state

Definition at line 56 of file mainloop_glib.c.

5.9.2 Field Documentation

5.9.2.1 getclientsourcefunc listen_source_t::getclientsource

Definition at line 57 of file mainloop_glib.c.

Referenced by accept_dispatch(), and listensource().

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

- mainloop_glib.c

5.10 source_t Struct Reference

Data Fields

- GSource source
  glib source: list of callbacks etc
- GPollFD pollfd
  pollfd for mainloop (g_source_add_poll())
- guint id
  id (g_source_remove())

5.10.1 Field Documentation

5.10.1.1 chargen_source_t source_t::chargen

state for chargen source

Definition at line 85 of file mainloop_glib.c.

Referenced by chargen_dispatch(), and getchargenclientsource().

5.10.1.2 echo_source_t source_t::echo

state for echo source

Definition at line 84 of file mainloop_glib.c.

Referenced by echo_dispatch_read(), echo_dispatch_write(), echo_prepare(), and getechoclientsource().

5.10.1.3 guint source_t::id

id (g_source_remove())

Definition at line 80 of file mainloop_glib.c.

Referenced by main(), and source_close().

5.10.1.4 listen_source_t source_t::listen

state for listen source

Definition at line 83 of file mainloop_glib.c.

Referenced by accept_dispatch(), and listensource().

5.10.1.5 GPollFD source_t::pollfd

pollfd for mainloop (g_source_add_poll())

Definition at line 79 of file mainloop_glib.c.

Referenced by accept_dispatch(), accept_prepare(), chargen_dispatch(), chargen_prepare(), check(), echo_dispatch(), echo_dispatch_read(), echo_dispatch_write(), echo_prepare(), listensource(), and source_close().

5.10.1.6 GSource source_t::source

glib source: list of callbacks etc

Definition at line 78 of file mainloop_glib.c.

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

- mainloop_glib.c

Chapter 6

mainloop File Documentation

6.1 mainloop_bad.c File Reference

Example for "bad" main loop.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <signal.h>
#include <fcntl.h>
```

Data Structures

- struct echoclient_t
  echo client state
- struct chargenclient_t

Defines

- #define PORT_ECHO 5005
  tcp port to listen for echo clients
- #define PORT_CHARGEN 5006
  tcp port to listen for chargen clients
- #define HEARTBEAT_INTERVAL 2
  heartbeat interval (in seconds)

- #define SLOWHEARTBEAT_INTERVAL 15

  slow heartbeat interval (in seconds)

- #define BUFSIZE 16

  buffer size for echo client

- #define MAXCLIENTS 4

  maximum number of clients

- #define MSG_HEARTBEAT 0
- #define MSG_SLOWHEARTBEAT 1
- #define MSG_MAINLOOP 2
- #define MSG_ACCEPT 3
- #define MSG_TOOMANY 4
- #define MSG_CLOSE 5
- #define MSG_READ 6
- #define MSG_WRITE 7
- #define MSG_FULL 8
- #define MSG_EMPTY 9
- #define MIN(a, b) ((a)<(b)?(a):(b))
- #define MAX(a, b) ((a)>(b)?(a):(b))

Functions

- void message (int msg)

  print message describing current activity

- int listensocket (int port)

  return tcp socket listening on port specified

- void setnonblock (int fd)

  set a file descriptor to be nonblocking

- void sighandler (int signo)

  signal handler

- void readecho (echoclient_t *client)

  read data from an echo client

- void writeecho (echoclient_t *client)

  write data to an echo client

- void writechargen (chargenclient_t *client)

  write data to a chargen client

- int main ()

Variables

- char chargen_buf [ ] = "0123456789abcdefghijklmnopqrstuv"
  characters to

6.1.1 Detailed Description

Example for "bad" main loop.

Author:
Rico Pajarola

This example tries to do everything as bad as possible without doing it just plain wrong (that's not as easy as it sounds). This is done by putting all the logic into one huge main loop, and explicitly spelling out all special cases in place.

Apart from the time spent trying to find worse ways to do things, this example was completed really quick. That is, at least until I tried to add a second client and a second timer...

To emphasize that this is the bad example, select() is used.

Definition in file mainloop_bad.c.

6.1.2 Define Documentation

6.1.2.1 #define BUFSIZE 16

buffer size for echo client

Definition at line 42 of file mainloop_bad.c.

Referenced by echo_dispatch_read(), echo_dispatch_write(), echo_prepare(), flowecho(), main(), readecho(), and writeecho().

6.1.2.2 #define HEARTBEAT_INTERVAL 2

heartbeat interval (in seconds)

Definition at line 36 of file mainloop_bad.c.

Referenced by main(), and sighandler().

6.1.2.3 #define MAX(a, b) ((a)>(b)?(a):(b))

Definition at line 59 of file mainloop_bad.c.

Referenced by alarmhandler(), and sighandler().

6.1.2.4 #define MAXCLIENTS 4

maximum number of clients

Definition at line 45 of file mainloop_bad.c.

Referenced by addclient(), initclients(), and main().

6.1.2.5 #define MIN(a, b) ((a)<(b)?(a):(b))

Definition at line 58 of file mainloop_bad.c.

Referenced by alarmhandler(), and sighandler().

6.1.2.6 #define MSG_ACCEPT 3

Definition at line 50 of file mainloop_bad.c.

Referenced by accept_dispatch(), acceptecho(), main(), and message().

6.1.2.7 #define MSG_CLOSE 5

Definition at line 52 of file mainloop_bad.c.

Referenced by closeclient(), message(), readecho(), source_close(), writechargen(), and writeecho().

6.1.2.8 #define MSG_EMPTY 9

Definition at line 56 of file mainloop_bad.c.

Referenced by echo_dispatch_write(), message(), and writeecho().

6.1.2.9 #define MSG_FULL 8

Definition at line 55 of file mainloop_bad.c.

Referenced by echo_dispatch_read(), message(), and readecho().

6.1.2.10 #define MSG_HEARTBEAT 0

Definition at line 47 of file mainloop_bad.c.

Referenced by heartbeat(), message(), and sighandler().

6.1.2.11 #define MSG_MAINLOOP 2

Definition at line 49 of file mainloop_bad.c.

Referenced by main(), mainloop(), and message().

6.1.2.12 #define MSG_READ 6

Definition at line 53 of file mainloop_bad.c.

Referenced by echo_dispatch_read(), message(), and readecho().

6.1.2.13 #define MSG_SLOWHEARTBEAT 1

Definition at line 48 of file mainloop_bad.c.

Referenced by message(), sighandler(), and slowheartbeat().

6.1.2.14 #define MSG_TOOMANY 4

Definition at line 51 of file mainloop_bad.c.

Referenced by addclient(), main(), and message().

6.1.2.15 #define MSG_WRITE 7

Definition at line 54 of file mainloop_bad.c.

Referenced by chargen_dispatch(), echo_dispatch_write(), message(), writechargen(), and writeecho().

6.1.2.16 #define PORT_CHARGEN 5006

tcp port to listen for chargen clients

Definition at line 33 of file mainloop_bad.c.

Referenced by main().

6.1.2.17 #define PORT_ECHO 5005

tcp port to listen for echo clients

Definition at line 30 of file mainloop_bad.c.

Referenced by main().

6.1.2.18 #define SLOWHEARTBEAT_INTERVAL 15

slow heartbeat interval (in seconds)

Definition at line 39 of file mainloop_bad.c.

Referenced by main(), and sighandler().

6.1.3 Function Documentation

6.1.3.1 int listensocket (int *port*) [static]

return tcp socket listening on port specified

Parameters:

port port number in host byte order

Returns:

file descriptor for listening socket

The socket is set to be nonblocking

Definition at line 193 of file mainloop_bad.c.

References setnonblock().

```
194 {
195     int             serverfd;
196     struct sockaddr_in sain;
197     int             one;
198 
199     /* set up tcp socket for listening */
200     sain.sin_family = AF_INET;
201     sain.sin_port = htons(port);
202     sain.sin_addr.s_addr = INADDR_ANY;
203     if ((serverfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
204         perror("socket(AF_INET, SOCK_STREAM, 0)");
205         exit(EXIT_FAILURE);
206     }
207     one = 1;
208     if (setsockopt
209         (serverfd, SOL_SOCKET, SO_REUSEADDR, &one,
210          (int) sizeof(one)) == -1) {
211         perror("setsockopt(SO_REUSEADDR)");
212         exit(EXIT_FAILURE);
213     }
214     if (bind
215         (serverfd, (struct sockaddr *) &sain,
216          sizeof(struct sockaddr_in)) == -1) {
217         perror("bind()");
218         exit(EXIT_FAILURE);
219     }
220     if (listen(serverfd, 5) == -1) {
221         perror("listen()");
222         exit(EXIT_FAILURE);
223     }
224     setnonblock(serverfd);
225 
226     printf("listening on port %d\n", port);
227 
228     return serverfd;
229 }
```

6.1.3.2 int main ()

Definition at line 379 of file mainloop_bad.c.

References BUFSIZE, chargenclient_t::fd, echoclient_t::fd, HEARTBEAT_INTERVAL, chargenclient_t::i, listensocket(), MAXCLIENTS, message(), MSG_ACCEPT, MSG_MAINLOOP, MSG_TOOMANY, echoclient_t::n, PORT_CHARGEN, PORT_ECHO, echoclient_t::r, readecho(), setnonblock(), sighandler(), SLOWHEARTBEAT_INTERVAL, echoclient_t::w, writechargen(), and writeecho().

```
380 {
381     int             echoserverfd;
382     int             chargenserverfd;
383     echoclient_t    echoclients[MAXCLIENTS];
384     chargenclient_t chargenclients[MAXCLIENTS];
385     fd_set          rfdset, wfdset;
386     int             fdsetmax;
387     int             n, i;
388     size_t          s;
389     struct sockaddr_in sain;
390 
391     printf("example: bad main loop\n");
392 
393     /*
394      * ignore SIGPIPE (this occurs whenever a chargen client closes
395      * the connection).
396      */
397     if (signal(SIGPIPE, SIG_IGN) == SIG_ERR) {
```

```
398         perror("signal(SIGPIPE, SIG_IGN)");
399         exit(EXIT_FAILURE);
400     }
401 
402     /*
403      * install heartbeat. this is done by calling the signal
404      * handler for SIGALRM which then installs itself as a signal
405      * handler and starts the alarm clock.
406      */
407     printf("heartbeat every %d seconds\n", HEARTBEAT_INTERVAL);
408     printf("slow heartbeat every %d seconds\n", SLOWHEARTBEAT_INTERVAL);
409     sighandler(SIGALRM);
410 
411     /*
412      * create server sockets for echo and chargen services
413      */
414     echoserverfd = listensocket(PORT_ECHO);
415     chargenserverfd = listensocket(PORT_CHARGEN);
416 
417     /*
418      * reset all client state slots (mark as unused)
419      */
420     for (i = 0; i < MAXCLIENTS; i++) {
421         echoclients[i].fd = -1;
422         chargenclients[i].fd = -1;
423     }
424 
425     /*
426      * initialize fdsets for select()
427      */
428     FD_ZERO(&rfdset);
429     FD_ZERO(&wfdset);
430     FD_SET(STDIN_FILENO, &rfdset);
431     FD_SET(echoserverfd, &rfdset);      /* fd 3 */
432     FD_SET(chargenserverfd, &rfdset);   /* fd 4 */
433     fdsetmax = chargenserverfd; /* fd 4 */
434 
435     /*
436      * THE main loop
437      * Remeber, this is the bad example. On Solaris, select() is a
438      * (rather clumsy) wrapper around poll(). There is no way to efficiently
439      * emulate select() using poll().
440      */
441     while (((n = select(fdsetmax + 1, &rfdset, &wfdset, NULL, NULL)) != -1)
442            || (errno == EINTR)) {
443         message(MSG_MAINLOOP);
444 
445         if (n == -1) {
446             /* got -1 and errno==EINTR */
447             continue;
448         }
449 
450         /*
451          * check for new echo connections
452          */
453         if (FD_ISSET(echoserverfd, &rfdset)) {
454             /* find free slot */
455             for (i = 0; (i < MAXCLIENTS) && (echoclients[i].fd != -1);
456                  i++);
457             if (echoclients[i].fd != -1) {
458                 /* no free slots */
459                 s = sizeof(sain);
460                 i = accept(echoserverfd, (struct sockaddr *) &sain, &s);
461                 close(i);
462                 message(MSG_TOOMANY);
463             } else {
464                 s = sizeof(sain);
```

```
465                 echoclients[i].fd =
466                     accept(echoserverfd, (struct sockaddr *) &sain, &s);
467                 message(MSG_ACCEPT);
468                 setnonblock(echoclients[i].fd);
469                 echoclients[i].r = 0;   /* start reading from buffer at pos 0 */
470                 echoclients[i].w = 0;   /* start writing to buffer at pos 0 */
471                 echoclients[i].n = 0;   /* 0 bytes in buffer */
472             }
473         }
474 
475         /*
476          * check for new chargen connections
477          */
478         if (FD_ISSET(chargenserverfd, &rfdset)) {
479             /* find free slot */
480             for (i = 0; (i < MAXCLIENTS) && (chargenclients[i].fd != -1);
481                  i++);
482             if (chargenclients[i].fd != -1) {
483                 /* no free slots */
484                 s = sizeof(sain);
485                 i = accept(chargenserverfd, (struct sockaddr *) &sain, &s);
486                 close(i);
487                 message(MSG_TOOMANY);
488             } else {
489                 s = sizeof(sain);
490                 chargenclients[i].fd =
491                     accept(chargenserverfd, (struct sockaddr *) &sain, &s);
492                 message(MSG_ACCEPT);
493                 setnonblock(chargenclients[i].fd);
494                 chargenclients[i].i = 0;        /* start sending from buffer
495                                                  * at pos 0 */
496             }
497         }
498 
499         /*
500          * try to read/write data for echo clients
501          */
502         for (i = 0; i < MAXCLIENTS; i++) {
503             if ((echoclients[i].fd != -1)
504                 && (FD_ISSET(echoclients[i].fd, &rfdset))) {
505                 readecho(&echoclients[i]);
506             }
507             if ((echoclients[i].fd != -1)
508                 && (FD_ISSET(echoclients[i].fd, &wfdset))) {
509                 writeecho(&echoclients[i]);
510             }
511         }
512 
513         /*
514          * try to write data for chargen clients
515          */
516         for (i = 0; i < MAXCLIENTS; i++) {
517             if ((chargenclients[i].fd != -1)
518                 && (FD_ISSET(chargenclients[i].fd, &wfdset))) {
519                 writechargen(&chargenclients[i]);
520             }
521         }
522 
523         /*
524          * reinitialize fdset
525          */
526         FD_ZERO(&rfdset);
527         FD_ZERO(&wfdset);
528         fdsetmax = chargenserverfd;
529         FD_SET(echoserverfd, &rfdset);
530         FD_SET(chargenserverfd, &rfdset);
531         for (i = 0; i < MAXCLIENTS; i++) {
```

```
532             if (echoclients[i].fd >= 0) {
533                 if (echoclients[i].n < BUFSIZE) {
534                     FD_SET(echoclients[i].fd, &rfdset);
535                     if (echoclients[i].fd > fdsetmax) {
536                         fdsetmax = echoclients[i].fd;
537                     }
538                 }
539                 if (echoclients[i].n > 0) {
540                     FD_SET(echoclients[i].fd, &wfdset);
541                     if (echoclients[i].fd > fdsetmax) {
542                         fdsetmax = echoclients[i].fd;
543                     }
544                 }
545             }
546         }
547         for (i = 0; i < MAXCLIENTS; i++) {
548             if (chargenclients[i].fd >= 0) {
549                 FD_SET(chargenclients[i].fd, &wfdset);
550                 if (chargenclients[i].fd > fdsetmax) {
551                     fdsetmax = chargenclients[i].fd;
552                 }
553             }
554         }
555     }
556     /* notreached */
557     exit(EXIT_FAILURE);
558 }
```

6.1.3.3 void message (int *msg*) [static]

print message describing current activity

Parameters:

msg id of message to print (MSG_XYZ)

Definition at line 96 of file mainloop_bad.c.

References MSG_ACCEPT, MSG_CLOSE, MSG_EMPTY, MSG_FULL, MSG_HEARTBEAT, MSG_-MAINLOOP, MSG_READ, MSG_SLOWHEARTBEAT, MSG_TOOMANY, and MSG_WRITE.

```
97 {
98     switch (msg) {
99     case MSG_HEARTBEAT:
100         printf("H");
101         break;
102     case MSG_SLOWHEARTBEAT:
103         printf("S");
104         break;
105     case MSG_MAINLOOP:
106         /* printf("M"); */
107         break;
108     case MSG_ACCEPT:
109         printf("A");
110         break;
111     case MSG_TOOMANY:
112         printf("T");
113         break;
114     case MSG_CLOSE:
115         printf("C");
116         break;
117     case MSG_READ:
118         printf("R");
```

```
119         break;
120     case MSG_WRITE:
121         printf("W");
122         break;
123     case MSG_FULL:
124         printf("F");
125         break;
126     case MSG_EMPTY:
127         printf("E");
128         break;
129     }
130     fflush(stdout);
131 }
```

6.1.3.4 void readecho (echoclient_t * *client*) [static]

read data from an echo client

Parameters:

client echo client state

If the buffer is not full, tries to do one read from the filedescriptor associated with the echo client.

Definition at line 257 of file mainloop_bad.c.

References echoclient_t::buf, BUFSIZE, echoclient_t::fd, message(), MSG_CLOSE, MSG_FULL, MSG_-READ, echoclient_t::n, echoclient_t::r, and echoclient_t::w.

```
258 {
259     int             nread;
260 
261     if (client->n == BUFSIZE) {
262         message(MSG_FULL);
263         return;
264     }
265 
266     if (client->r >= client->w) {
267         nread =
268             read(client->fd, client->buf + client->r, BUFSIZE - client->r);
269     } else {
270         nread =
271             read(client->fd, client->buf + client->r,
272                  client->w - client->r);
273     }
274 
275     switch (nread) {
276     case 0:
277         message(MSG_CLOSE);
278         close(client->fd);
279         client->fd = -1;
280         break;
281     case -1:
282         if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
283             perror("read()");
284             exit(EXIT_FAILURE);
285         }
286         break;
287     default:
288         message(MSG_READ);
289         client->n += nread;
290         client->r += nread;
291         client->r %= BUFSIZE;
292     }
293 }
```

6.1.3.5 void setnonblock (int *fd*) [static]

set a file descriptor to be nonblocking

Parameters:
fd file descriptor

Non-Blocking works only for Sockets, Pipes and slow devices, it has no effect when used with regular files

Definition at line 240 of file mainloop_bad.c.

```
241 {
242     int             flag;
243 
244     flag = fcntl(fd, F_GETFL);
245     fcntl(fd, F_GETFL, flag | O_NONBLOCK);
246 }
```

6.1.3.6 void sighandler (int *signo*) [static]

signal handler

Parameters:
signo signal number

Definition at line 139 of file mainloop_bad.c.

References HEARTBEAT_INTERVAL, MAX, message(), MIN, MSG_HEARTBEAT, MSG_-SLOWHEARTBEAT, sighandler(), and SLOWHEARTBEAT_INTERVAL.

Referenced by main(), and sighandler().

```
140 {
141     struct timeval  now;
142     static time_t   time_heartbeat, time_slowheartbeat;
143     int             nextalarm;
144 
145     switch (signo) {
146     case SIGALRM:
147         /*
148          * (re)install signal handler and set new alarm
149          * there is a possible race condition, but for alarm()/SIGALRM this is not a real concern
150          */
151         if (signal(SIGALRM, &sighandler) == SIG_ERR) {
152             perror("signal(SIGALRM)");
153             exit(EXIT_FAILURE);
154         }
155         gettimeofday(&now, NULL);
156         if (time_heartbeat == 0) {
157             time_heartbeat = now.tv_sec + HEARTBEAT_INTERVAL;
158             time_slowheartbeat = now.tv_sec + SLOWHEARTBEAT_INTERVAL;
159         }
160         if (time_heartbeat <= now.tv_sec) {
161             time_heartbeat += HEARTBEAT_INTERVAL;
162             message(MSG_HEARTBEAT);
163         }
164         if (time_slowheartbeat <= now.tv_sec) {
165             time_slowheartbeat += SLOWHEARTBEAT_INTERVAL;
166             message(MSG_SLOWHEARTBEAT);
167         }
```

```
168
169         nextalarm =
170             MAX(1, MIN(time_heartbeat, time_slowheartbeat) - now.tv_sec);
171
172         if (alarm(nextalarm) == -1) {
173             perror("alarm()");
174             exit(EXIT_FAILURE);
175         }
176         break;
177     default:
178         /* ignore everything */
179         break;
180     }
181 }
```

6.1.3.7 void writechargen (chargenclient_t * *client*) [static]

write data to a chargen client

Parameters:

client chargen client state

Definition at line 350 of file mainloop_bad.c.

References chargen_buf, chargenclient_t::fd, chargenclient_t::i, message(), MSG_CLOSE, and MSG_-WRITE.

```
351 {
352     int             nwrite;
353
354     nwrite =
355         write(client->fd, chargen_buf + client->i,
356               sizeof(chargen_buf) - client->i);
357
358     switch (nwrite) {
359     case -1:
360         if (errno == EPIPE) {
361             message(MSG_CLOSE);
362             close(client->fd);
363             client->fd = -1;
364         } else if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
365             perror("write()");
366             exit(EXIT_FAILURE);
367         }
368         break;
369     case 0:
370         break;
371     default:
372         message(MSG_WRITE);
373         client->i += nwrite;
374         client->i %= sizeof(chargen_buf);
375     }
376 }
```

6.1.3.8 void writeecho (echoclient_t * *client*) [static]

write data to an echo client

Parameters:

client echo client state

If the buffer is not empty, tries to do one write to the filedescriptor associated with the echo client.

Definition at line 304 of file mainloop_bad.c.

References echoclient_t::buf, BUFSIZE, echoclient_t::fd, message(), MSG_CLOSE, MSG_EMPTY, MSG_WRITE, echoclient_t::n, echoclient_t::r, and echoclient_t::w.

```
305 {
306     int             nwrite;
307 
308     if (client->n == 0) {
309         message(MSG_EMPTY);
310         return;
311     }
312 
313     if (client->r > client->w) {
314         nwrite =
315             write(client->fd, client->buf + client->w,
316                   client->r - client->w);
317     } else {
318         nwrite =
319             write(client->fd, client->buf + client->w,
320                   BUFSIZE - client->w);
321     }
322 
323     switch (nwrite) {
324     case -1:
325         if (errno == EPIPE) {
326             message(MSG_CLOSE);
327             close(client->fd);
328             client->fd = -1;
329         } else if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
330             perror("write()");
331             exit(EXIT_FAILURE);
332         }
333         break;
334     case 0:
335         break;
336     default:
337         message(MSG_WRITE);
338         client->n -= nwrite;
339         client->w += nwrite;
340         client->w %= BUFSIZE;
341     }
342 }
```

6.1.4 Variable Documentation

6.1.4.1 char chargen_buf[ ] = "0123456789abcdefghijklmnopqrstuv" [static]

characters to

Returns:
in chargen service

Definition at line 62 of file mainloop_bad.c.

Referenced by writechargen().

6.2 mainloop_glib.c File Reference

Example for main loop using glib 2.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <signal.h>
#include <fcntl.h>
#include "glib.h"
```

Data Structures

- struct listen_source_t
  listen source state
- struct echo_source_t
  echo source state
- struct chargen_source_t
  chargen source state
- struct source_t

Defines

- #define PORT_ECHO 5005
  tcp port to listen for echo clients
- #define PORT_CHARGEN 5006
  tcp port to listen for chargen clients
- #define HEARTBEAT_INTERVAL 2
  heartbeat interval (in seconds)
- #define SLOWHEARTBEAT_INTERVAL 15
  slow heartbeat interval (in seconds)
- #define BUFSIZE 16
  buffer size for echo client
- #define MSG_HEARTBEAT 0

- #define MSG_SLOWHEARTBEAT 1
- #define MSG_MAINLOOP 2
- #define MSG_ACCEPT 3
- #define MSG_TOOMANY 4
- #define MSG_CLOSE 5
- #define MSG_READ 6
- #define MSG_WRITE 7
- #define MSG_FULL 8
- #define MSG_EMPTY 9

Typedefs

- typedef GSource *(* getclientsourcefunc)()

Functions

- void message (int msg)

  print message describing current activity

- int listensocket (int port)

  return tcp socket listening on port specified

- void setnonblock (int fd)

  set a file descriptor to be nonblocking

- source_t * listensource (getclientsourcefunc getclientsource, int port)

  construct listensource listening on port specified

- GSource * getechoclientsource ()

  construct GSourceFuncs for echo client

- GSource * getchargenclientsource ()

  construct GSourceFuncs for chargen client

- gboolean heartbeat (gpointer data)

  print heartbeat message

- gboolean slowheartbeat (gpointer data)

  print "slowheartbeat" message

- gboolean check (GSource *source)

  check whether source is ready for processing

- void source_close (source_t *source)

  close a source and dispose of source

- gboolean accept_prepare (GSource *source, gint *timeout)

  Prepare GSource for polling a server socket for accept.

- gboolean accept_dispatch (GSource *source, GSourceFunc callback, gpointer user_data)
  dispatch (process) listening socket
- gboolean echo_prepare (GSource *source, gint *timeout)
  Prepare GSource for polling an echo client.
- gboolean echo_dispatch (GSource *source, GSourceFunc callback, gpointer user_data)
  dispatch (process) echo client
- void echo_dispatch_read (source_t *echosource)
  read data from echo client
- void echo_dispatch_write (source_t *echosource)
  write data to an echo client
- gboolean chargen_prepare (GSource *source, gint *timeout)
  Prepare GSource for polling an chargen client.
- gboolean chargen_dispatch (GSource *source, GSourceFunc callback, gpointer user_data)
  dispatch (process) chargen client
- void blocksigpipe ()
  block SIGPIPE
- int main ()

Variables

- char chargen_buf [ ] = "0123456789abcdefghijklmnopqrstuv"
  characters to return in chargen service

6.2.1 Detailed Description

Example for main loop using glib 2.

Author:
Rico Pajarola

This example uses glib 2 for handling events.

Definition in file mainloop_glib.c.

6.2.2 Define Documentation

6.2.2.1 #define BUFSIZE 16

buffer size for echo client

Definition at line 35 of file mainloop_glib.c.

6.2.2.2 #define HEARTBEAT_INTERVAL 2

heartbeat interval (in seconds)

Definition at line 29 of file mainloop_glib.c.

6.2.2.3 #define MSG_ACCEPT 3

Definition at line 40 of file mainloop_glib.c.

6.2.2.4 #define MSG_CLOSE 5

Definition at line 42 of file mainloop_glib.c.

6.2.2.5 #define MSG_EMPTY 9

Definition at line 46 of file mainloop_glib.c.

6.2.2.6 #define MSG_FULL 8

Definition at line 45 of file mainloop_glib.c.

6.2.2.7 #define MSG_HEARTBEAT 0

Definition at line 37 of file mainloop_glib.c.

6.2.2.8 #define MSG_MAINLOOP 2

Definition at line 39 of file mainloop_glib.c.

6.2.2.9 #define MSG_READ 6

Definition at line 43 of file mainloop_glib.c.

6.2.2.10 #define MSG_SLOWHEARTBEAT 1

Definition at line 38 of file mainloop_glib.c.

6.2.2.11 #define MSG_TOOMANY 4

Definition at line 41 of file mainloop_glib.c.

6.2.2.12 #define MSG_WRITE 7

Definition at line 44 of file mainloop_glib.c.

6.2.2.13 #define PORT_CHARGEN 5006

tcp port to listen for chargen clients

Definition at line 26 of file mainloop_glib.c.

6.2.2.14 #define PORT_ECHO 5005

tcp port to listen for echo clients

Definition at line 23 of file mainloop_glib.c.

6.2.2.15 #define SLOWHEARTBEAT_INTERVAL 15

slow heartbeat interval (in seconds)

Definition at line 32 of file mainloop_glib.c.

6.2.3 Typedef Documentation

6.2.3.1 typedef GSource*(* getclientsourcefunc)()

Definition at line 51 of file mainloop_glib.c.

6.2.4 Function Documentation

6.2.4.1 gboolean accept_dispatch (GSource * *gsource*, GSourceFunc *callback*, gpointer *user_data*) [static]

dispatch (process) listening socket

Parameters:

gsource source to process

callback callback function (unused)

user_data I have absolutely no idea how to use this... nice idea though

Returns:

always TRUE

Definition at line 347 of file mainloop_glib.c.

References listen_source_t::getclientsource, source_t::listen, message(), MSG_ACCEPT, source_t::pollfd, and setnonblock().

Referenced by listensource().

```
349 {
350     struct sockaddr_in sain;
351     int             fd;
352     socklen_t       t;
353     source_t       *source;
354     source_t       *clientsource;
355
```

```
356     source = (source_t *) gsource;
357
358     if (source->pollfd.revents & (G_IO_HUP | G_IO_ERR)) {
359         perror("accept_dispatch()");
360         exit(EXIT_FAILURE);
361     }
362
363     /* accept new connection */
364     t = sizeof(sain);
365     if ((fd = accept(source->pollfd.fd, (void *) &sain, &t)) == -1) {
366         if (errno == EWOULDBLOCK) {
367             return TRUE;
368         }
369         perror("accept(ECHO)");
370         exit(EXIT_FAILURE);
371     }
372     setnonblock(fd);
373     message(MSG_ACCEPT);
374
375     clientsource = (source_t *) source->listen.getclientsource();
376     clientsource->pollfd.fd = fd;
377     g_source_add_poll((GSource *) clientsource, &(clientsource->pollfd));
378     clientsource->id =
379         g_source_attach((GSource *) clientsource,
380                         g_source_get_context((GSource *) clientsource));
381     return TRUE;
382 }
```

6.2.4.2 gboolean accept_prepare (GSource * *source*, gint * *timeout*) [static]

Prepare GSource for polling a server socket for accept.

Parameters:

source GSource to prepare

timeout maximum timeout to set for poll() (out)

Returns:

always FALSE (use poll)

Definition at line 326 of file mainloop_glib.c.

References source_t::pollfd.

Referenced by listensource().

```
327 {
328     source_t       *xsource;
329
330     xsource = (source_t *) source;
331
332     xsource->pollfd.events = G_IO_IN;
333     *timeout = -1;              /* no timeout */
334     return FALSE;
335 }
```

6.2.4.3 void blocksigpipe (void) [static]

block SIGPIPE

Trying to write to a socket when the other end has already closed the connection results in SIGPIPE. Not usefull in this context.

Definition at line 170 of file mainloop_glib.c.

Referenced by main().

```
171 {
172     struct sigaction act;
173 
174     act.sa_handler = SIG_IGN;
175     sigemptyset(&act.sa_mask);
176     act.sa_flags = SA_RESTART;
177     if (sigaction(SIGPIPE, &act, NULL) == -1) {
178         perror("sigaction(SIGPIPE, <ignore>)");
179         exit(EXIT_FAILURE);
180     }
181 }
```

6.2.4.4 gboolean chargen_dispatch (GSource * *source*, GSourceFunc *callback*, gpointer *user_data*) [static]

dispatch (process) chargen client

Parameters:

source source to process

callback callback function (unused)

user_data ?

Returns:

always TRUE

there is no chargen_dispatch_write, writing is done directly in chargen_dispatch

Definition at line 633 of file mainloop_glib.c.

References source_t::chargen, chargen_buf, chargen_source_t::i, message(), MSG_WRITE, source_t::pollfd, and source_close().

Referenced by getchargenclientsource().

```
635 {
636     ssize_t         nwrite;
637     source_t       *chargensource;
638 
639     chargensource = (source_t *) source;
640 
641     nwrite =
642         write(chargensource->pollfd.fd,
643               chargen_buf + chargensource->chargen.i,
644               sizeof(chargen_buf) - chargensource->chargen.i);
645     switch (nwrite) {
646     case -1:
647         if (errno == EPIPE) {
648             source_close((source_t *) chargensource);
649             return TRUE;
650         } else if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
651             perror("write()");
652             exit(EXIT_FAILURE);
653         }
```

```
654         break;
655     case 0:
656         break;
657     default:
658         message(MSG_WRITE);
659         chargensource->chargen.i += nwrite;
660         chargensource->chargen.i %= sizeof(chargen_buf);
661     }
662     return TRUE;
663 }
```

6.2.4.5 gboolean chargen_prepare (GSource * *source*, gint * *timeout*) [static]

Prepare GSource for polling an chargen client.

Parameters:

source GSource to prepare

timeout maximum timeout to set for poll() (out)

Returns:

always FALSE (use poll)

Definition at line 609 of file mainloop_glib.c.

References source_t::pollfd.

Referenced by getchargenclientsource().

```
610 {
611     source_t       *chargensource;
612 
613     chargensource = (source_t *) source;
614 
615     chargensource->pollfd.events = G_IO_OUT;
616 
617     return FALSE;
618 }
```

6.2.4.6 gboolean check (GSource * *source*) [static]

check whether source is ready for processing

Parameters:

source GSource to check

Returns:

TRUE if resource is ready

Definition at line 307 of file mainloop_glib.c.

References source_t::pollfd.

Referenced by getchargenclientsource(), getechoclientsource(), and listensource().

```
308 {
309
310     source_t       *xsource;
311
312     xsource = (source_t *) source;
313
314     return xsource->pollfd.revents ? TRUE : FALSE;
315 }
```

6.2.4.7 gboolean echo_dispatch (GSource * *source*, GSourceFunc *callback*, gpointer *user_data*) [static]

dispatch (process) echo client

Parameters:

source source to process

callback callback function (unused)

user_data ?

Returns:

always TRUE

Definition at line 471 of file mainloop_glib.c.

References echo_dispatch_read(), echo_dispatch_write(), source_t::pollfd, and source_close().

Referenced by getechoclientsource().

```
472 {
473     source_t       *echosource;
474
475     echosource = (source_t *) source;
476
477     if (echosource->pollfd.revents & (G_IO_HUP | G_IO_ERR)) {
478         source_close((source_t *) echosource);
479     }
480     if (echosource->pollfd.revents & G_IO_IN) {
481         echo_dispatch_read(echosource);
482     }
483     if (echosource->pollfd.revents & G_IO_OUT) {
484         echo_dispatch_write(echosource);
485     }
486     return TRUE;
487 }
```

6.2.4.8 void echo_dispatch_read (source_t * *echosource*) [static]

read data from echo client

Parameters:

echosource echo client source

If the buffer is not full, tries to do one read from the filedescriptor associated with this echo client.

Definition at line 498 of file mainloop_glib.c.

References echo_source_t::buf, BUFSIZE, source_t::echo, message(), MSG_FULL, MSG_READ, echo_source_t::n, source_t::pollfd, echo_source_t::r, source_close(), and echo_source_t::w.

Referenced by echo_dispatch().

```
499 {
500     ssize_t         nread;
501 
502     if (echosource->echo.n == BUFSIZE) {
503         message(MSG_FULL);
504         return;
505     }
506 
507     if (echosource->echo.r >= echosource->echo.w) {
508         nread =
509             read(echosource->pollfd.fd,
510                  echosource->echo.buf + echosource->echo.r,
511                  BUFSIZE - echosource->echo.r);
512     } else {
513         nread =
514             read(echosource->pollfd.fd,
515                  echosource->echo.buf + echosource->echo.r,
516                  echosource->echo.w - echosource->echo.r);
517     }
518 
519     switch (nread) {
520     case -1:
521         if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
522             perror("read()");
523             exit(EXIT_FAILURE);
524         }
525         break;
526     case 0:
527         source_close((source_t *) echosource);
528         return;
529     default:
530         message(MSG_READ);
531         echosource->echo.n += nread;
532         echosource->echo.r += nread;
533         echosource->echo.r %= BUFSIZE;
534     }
535 }
```

6.2.4.9 void echo_dispatch_write (source_t * *echosource*) [static]

write data to an echo client

Parameters:

echosource echo client source

If the buffer is not empty, tries to do one write to the filedescriptor associated with the echo client.

Definition at line 546 of file mainloop_glib.c.

References echo_source_t::buf, BUFSIZE, source_t::echo, message(), MSG_EMPTY, MSG_WRITE, echo_source_t::n, source_t::pollfd, echo_source_t::r, source_close(), and echo_source_t::w.

Referenced by echo_dispatch().

```
547 {
548     ssize_t         nwrite;
549 
```

```
550     if (echosource->echo.n == 0) {
551         message(MSG_EMPTY);
552         return;
553     }
554 
555     if (echosource->echo.r > echosource->echo.w) {
556         nwrite =
557             write(echosource->pollfd.fd,
558                   echosource->echo.buf + echosource->echo.w,
559                   echosource->echo.r - echosource->echo.w);
560     } else {
561         nwrite =
562             write(echosource->pollfd.fd,
563                   echosource->echo.buf + echosource->echo.w,
564                   BUFSIZE - echosource->echo.w);
565     }
566 
567     switch (nwrite) {
568     case -1:
569         if (errno == EPIPE) {
570             source_close((source_t *) echosource);
571         } else if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
572             perror("write()");
573             exit(EXIT_FAILURE);
574         }
575         break;
576     case 0:
577         break;
578     default:
579         message(MSG_WRITE);
580         echosource->echo.n -= nwrite;
581         echosource->echo.w += nwrite;
582         echosource->echo.w %= BUFSIZE;
583     }
584 }
```

6.2.4.10 gboolean echo_prepare (GSource * *source*, gint * *timeout*) [static]

Prepare GSource for polling an echo client.

Parameters:

source GSource to prepare

timeout maximum timeout to set for poll() (out)

Returns:

always FALSE (use poll)

If the buffer is full, checking for read is turned off resp. if the buffer is empty, checking for write is turned off.

Definition at line 442 of file mainloop_glib.c.

References BUFSIZE, source_t::echo, echo_source_t::n, and source_t::pollfd.

Referenced by getechoclientsource().

```
443 {
444     source_t       *echosource;
445 
446     echosource = (source_t *) source;
447 
```

```
448     switch (echosource->echo.n) {
449     case 0:
450         echosource->pollfd.events = G_IO_IN;
451         break;
452     case BUFSIZE:
453         echosource->pollfd.events = G_IO_OUT;
454         break;
455     default:
456         echosource->pollfd.events = G_IO_IN | G_IO_OUT;
457     }
458     return FALSE;
459 }
```

6.2.4.11 GSource * getchargenclientsource () [static]

construct GSourceFuncs for chargen client

Returns:
GSource for chargen client

Definition at line 414 of file mainloop_glib.c.

References source_t::chargen, chargen_dispatch(), chargen_prepare(), check(), and chargen_source_t::i.

Referenced by main().

```
415 {
416     static GSourceFuncs funcs = { NULL, NULL, NULL, NULL, NULL, NULL };
417     source_t       *chargensource;
418 
419     /* cannot reference functions in static initializer */
420     funcs.prepare = chargen_prepare;
421     funcs.check = check;
422     funcs.dispatch = chargen_dispatch;
423     funcs.finalize = NULL;
424 
425     chargensource = (source_t *) g_source_new(&funcs, sizeof(source_t));
426     chargensource->chargen.i = 0;
427     return (GSource *) chargensource;
428 }
```

6.2.4.12 GSource * getechoclientsource () [static]

construct GSourceFuncs for echo client

Returns:
GSource for echo client

Definition at line 390 of file mainloop_glib.c.

References check(), source_t::echo, echo_dispatch(), echo_prepare(), echo_source_t::n, echo_source_t::r, and echo_source_t::w.

Referenced by main().

```
391 {
392     static GSourceFuncs funcs = { NULL, NULL, NULL, NULL, NULL, NULL };
393     source_t       *echosource;
```

```
394
395     /* cannot reference functions in static initializer */
396     funcs.prepare = echo_prepare;
397     funcs.check = check;
398     funcs.dispatch = echo_dispatch;
399     funcs.finalize = NULL;
400
401     echosource = (source_t *) g_source_new(&funcs, sizeof(source_t));
402     echosource->echo.r = 0;
403     echosource->echo.w = 0;
404     echosource->echo.n = 0;
405     return (GSource *) echosource;
406 }
```

6.2.4.13 gboolean heartbeat (gpointer *data*) [static]

print heartbeat message

Returns:
always TRUE

Definition at line 281 of file mainloop_glib.c.

References message(), and MSG_HEARTBEAT.

Referenced by main().

```
282 {
283     message(MSG_HEARTBEAT);
284     return TRUE;
285 }
```

6.2.4.14 int listensocket (int *port*) [static]

return tcp socket listening on port specified

Parameters:
port port number in host byte order

Returns:
file descriptor for listening socket

The socket is set to be nonblocking

Definition at line 193 of file mainloop_glib.c.

References setnonblock().

```
194 {
195     int             serverfd;
196     struct sockaddr_in sain;
197     int             one;
198
199     /* set up tcp socket for listening */
200     sain.sin_family = AF_INET;
201     sain.sin_port = htons(port);
202     sain.sin_addr.s_addr = INADDR_ANY;
```

```
203     if ((serverfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
204         perror("socket(AF_INET, SOCK_STREAM, 0)");
205         exit(EXIT_FAILURE);
206     }
207     one = 1;
208     if (setsockopt
209         (serverfd, SOL_SOCKET, SO_REUSEADDR, &one,
210          (int) sizeof(one)) == -1) {
211         perror("setsockopt(SO_REUSEADDR)");
212         exit(EXIT_FAILURE);
213     }
214     if (bind
215         (serverfd, (struct sockaddr *) &sain,
216          sizeof(struct sockaddr_in)) == -1) {
217         perror("bind()");
218         exit(EXIT_FAILURE);
219     }
220     if (listen(serverfd, 5) == -1) {
221         perror("listen()");
222         exit(EXIT_FAILURE);
223     }
224     setnonblock(serverfd);
225 
226     printf("listening on port %d\n", port);
227 
228     return serverfd;
229 }
```

6.2.4.15 source_t * listensource (getclientsourcefunc *getclientsource*, int *port*) [static]

construct listensource listening on port specified

Parameters:

getclientsource function returning clientsource for accepted connection

port tcp port to listen on in host byte order

Returns:

listensource_t

Definition at line 257 of file mainloop_glib.c.

References accept_dispatch(), accept_prepare(), check(), listen_source_t::getclientsource, source_t::listen, listensocket(), listensource(), and source_t::pollfd.

Referenced by listensource(), and main().

```
258 {
259     static GSourceFuncs listenfuncs;
260     source_t       *listensource;
261 
262     listenfuncs.prepare = &accept_prepare;
263     listenfuncs.check = &check;
264     listenfuncs.dispatch = &accept_dispatch;
265     listenfuncs.finalize = NULL;
266     listensource =
267         (source_t *) g_source_new(&listenfuncs, sizeof(source_t));
268     listensource->listen.getclientsource = getclientsource;
269 
270     listensource->pollfd.fd = listensocket(port);
271     g_source_add_poll((GSource *) listensource, &(listensource->pollfd));
272     return listensource;
273 }
```

6.2.4.16 int main ()

Definition at line 666 of file mainloop_glib.c.

References blocksigpipe(), getchargenclientsource(), getechoclientsource(), heartbeat(), HEARTBEAT_-INTERVAL, source_t::id, listensource(), PORT_CHARGEN, PORT_ECHO, slowheartbeat(), and SLOWHEARTBEAT_INTERVAL.

```
667 {
668     GMainLoop      *mainloop;
669     GMainContext   *context;
670     source_t       *listenecho, *listenchargen;
671
672     /* check glib version */
673     if (!GLIB_CHECK_VERSION(2, 0, 0)) {
674         fprintf(stderr, "glib %d.%d.%d is too old\n", GLIB_MAJOR_VERSION,
675                 GLIB_MINOR_VERSION, GLIB_MICRO_VERSION);
676         exit(EXIT_FAILURE);
677     }
678
679     printf("example: glib main loop\n");
680
681     blocksigpipe();
682
683     /* create main loop */
684     context = g_main_context_default();
685     mainloop = g_main_loop_new(context, FALSE);
686
687     /* create echo service */
688     listenecho = listensource(getechoclientsource, PORT_ECHO);
689     listenecho->id = g_source_attach((GSource *) listenecho, context);
690
691     /* create chargen service */
692     listenchargen = listensource(getchargenclientsource, PORT_CHARGEN);
693     listenchargen->id =
694         g_source_attach((GSource *) listenchargen, context);
695
696     /* install heartbeat */
697     (void) g_timeout_add(1000 * HEARTBEAT_INTERVAL, &heartbeat, NULL);
698     (void) g_timeout_add(1000 * SLOWHEARTBEAT_INTERVAL, &slowheartbeat, NULL);
699     printf("heartbeat every %d seconds\n", HEARTBEAT_INTERVAL);
700     printf("slow heartbeat every %d seconds\n", SLOWHEARTBEAT_INTERVAL);
701
702     /* run the main loop */
703     g_main_loop_run(mainloop);
704     /* notreached */
705
706     g_main_loop_unref(mainloop);
707     exit(EXIT_SUCCESS);
708 }
```

6.2.4.17 void message (int *msg*) [static]

print message describing current activity

Parameters:

msg id of message to print (MSG_XYZ)

Definition at line 126 of file mainloop_glib.c.

References MSG_ACCEPT, MSG_CLOSE, MSG_EMPTY, MSG_FULL, MSG_HEARTBEAT, MSG_-MAINLOOP, MSG_READ, MSG_SLOWHEARTBEAT, MSG_TOOMANY, and MSG_WRITE.

```
127 {
128     switch (msg) {
129     case MSG_HEARTBEAT:
130         printf("H");
131         break;
132     case MSG_SLOWHEARTBEAT:
133         printf("S");
134         break;
135     case MSG_MAINLOOP:
136         printf("M");
137         break;
138     case MSG_ACCEPT:
139         printf("A");
140         break;
141     case MSG_TOOMANY:
142         printf("T");
143         break;
144     case MSG_CLOSE:
145         printf("C");
146         break;
147     case MSG_READ:
148         printf("R");
149         break;
150     case MSG_WRITE:
151         printf("W");
152         break;
153     case MSG_FULL:
154         printf("F");
155         break;
156     case MSG_EMPTY:
157         printf("E");
158         break;
159     }
160     fflush(stdout);
161 }
```

6.2.4.18 void setnonblock (int *fd*) [static]

set a file descriptor to be nonblocking

Parameters:

fd file descriptor

Non-Blocking works only for Sockets, Pipes and slow devices, it has no effect when used with regular files

Definition at line 240 of file mainloop_glib.c.

```
241 {
242     int             flag;
243
244     flag = fcntl(fd, F_GETFL);
245     fcntl(fd, F_GETFL, flag | O_NONBLOCK);
246 }
```

6.2.4.19 gboolean slowheartbeat (gpointer *data*) [static]

print "slowheartbeat" message

Returns:

always TRUE

Definition at line 293 of file mainloop_glib.c.

References message(), and MSG_SLOWHEARTBEAT.

Referenced by main().

```
294 {
295     message(MSG_SLOWHEARTBEAT);
296     return TRUE;
297 }
```

6.2.4.20 void source_close (source_t * *source*) [static]

close a source and dispose of source

Parameters:

source source to close/dispose

Definition at line 592 of file mainloop_glib.c.

References source_t::id, message(), MSG_CLOSE, and source_t::pollfd.

Referenced by chargen_dispatch(), echo_dispatch(), echo_dispatch_read(), and echo_dispatch_write().

```
593 {
594     message(MSG_CLOSE);
595     (void) g_source_remove(source->id);
596     close(source->pollfd.fd);
597     g_source_unref((GSource *) source);
598 }
```

6.2.5 Variable Documentation

6.2.5.1 char chargen_buf[ ] = "0123456789abcdefghijklmnopqrstuv" [static]

characters to return in chargen service

Definition at line 49 of file mainloop_glib.c.

Referenced by chargen_dispatch().

6.3 mainloop_good.c File Reference

Example for "good" main loop.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <signal.h>
#include <fcntl.h>
```

Data Structures

- struct echo_client_t
  echo client specific state
- struct chargen_client_t
  chargen client specific state
- struct client_t
  client state
- struct alarm_t
  alarm handler state

Defines

- #define PORT_ECHO 5005
  tcp port to listen for echo clients
- #define PORT_CHARGEN 5006
  tcp port to listen for chargen clients
- #define HEARTBEAT_INTERVAL 2
  heartbeat interval (in seconds)
- #define SLOWHEARTBEAT_INTERVAL 15
  slow heartbeat interval (in seconds)
- #define BUFSIZE 16
  buffer size for echo client
- #define MAXCLIENTS 8

maximum number of clients

- #define MAXALARMS 8

  maximum number of different alarms

- #define MSG_HEARTBEAT 0
- #define MSG_SLOWHEARTBEAT 1
- #define MSG_MAINLOOP 2
- #define MSG_ACCEPT 3
- #define MSG_TOOMANY 4
- #define MSG_CLOSE 5
- #define MSG_READ 6
- #define MSG_WRITE 7
- #define MSG_FULL 8
- #define MSG_EMPTY 9
- #define MIN(a, b) ((a)<(b)?(a):(b))
- #define MAX(a, b) ((a)>(b)?(a):(b))

Typedefs

- typedef void(∗ eventhandler_t)(int i)

  event handler

- typedef void(∗ alarmhandler_t)(void)

  alarm handler

Functions

- void message (int msg)

  print message describing current activity

- void blocksigpipe (void)

  block SIGPIPE

- void initalarms (void)

  install non-resetting signal handler for alarms

- void addalarm (alarmhandler_t handler, long interval)

  add new alarm

- void checkalarms (void)

  check for pending alarms and execute alarm handlers

- void alarmhandler (void)

  signal handler for alarms

- void heartbeat (void)

  print heartbeat message

- void slowheartbeat (void)

  print "slowheartbeat" message

- int listensocket (int port)

  return tcp socket listening on port specified

- void setnonblock (int fd)

  set a file descriptor to be nonblocking

- void initclients ()

  initialize client state array

- void addclient (int fd, eventhandler_t read, eventhandler_t write, eventhandler_t except)

  add new client

- void delclient (int i)

  delete client

- void mainloop ()

  execute one iteration of THE mainloop

- void closeclient (int i)

  close and delete client session

- void acceptecho (int i)

  accept connection and set up new session for "echo" service

- void readecho (int i)

  read data from echo client

- void writeecho (int i)

  write data to an echo client

- void flowecho (int i)

  do "flow" control for echo

- void acceptchargen (int i)

  accept connection and set up new session for "chargen" service

- void writechargen (int i)

  write data to an chargen client

- int main ()

Variables

- char chargen_buf [ ] = "0123456789abcdefghijklmnopqrstuv"

  characters to return in chargen service

- unsigned long npollfd

  number of file descriptors to check

- client_t clients [MAXCLIENTS]

  array of client states

- pollfd pollfds [MAXCLIENTS]

  pollfds for poll()

- alarm_t alarms [MAXALARMS]

  alarms

6.3.1 Detailed Description

Example for "good" main loop.

Author:
Rico Pajarola

This example does essentially what glib would do: events are abstracted using callbacks making the main-loop generic (but not as generic as glib). Even though it is easy to add a new input or event source, the whole mechanism is still very much tied to the structure of this program).

Definition in file mainloop_good.c.

6.3.2 Define Documentation

6.3.2.1 #define BUFSIZE 16

buffer size for echo client

Definition at line 37 of file mainloop_good.c.

6.3.2.2 #define HEARTBEAT_INTERVAL 2

heartbeat interval (in seconds)

Definition at line 31 of file mainloop_good.c.

6.3.2.3 #define MAX(a, b) ((a)>(b)?(a):(b))

Definition at line 57 of file mainloop_good.c.

6.3.2.4 #define MAXALARMS 8

maximum number of different alarms

Definition at line 43 of file mainloop_good.c.

Referenced by addalarm(), alarmhandler(), checkalarms(), and initalarms().

6.3.2.5 #define MAXCLIENTS 8

maximum number of clients

Definition at line 40 of file mainloop_good.c.

6.3.2.6 #define MIN(a, b) ((a)<(b)?(a):(b))

Definition at line 56 of file mainloop_good.c.

6.3.2.7 #define MSG_ACCEPT 3

Definition at line 48 of file mainloop_good.c.

6.3.2.8 #define MSG_CLOSE 5

Definition at line 50 of file mainloop_good.c.

6.3.2.9 #define MSG_EMPTY 9

Definition at line 54 of file mainloop_good.c.

6.3.2.10 #define MSG_FULL 8

Definition at line 53 of file mainloop_good.c.

6.3.2.11 #define MSG_HEARTBEAT 0

Definition at line 45 of file mainloop_good.c.

6.3.2.12 #define MSG_MAINLOOP 2

Definition at line 47 of file mainloop_good.c.

6.3.2.13 #define MSG_READ 6

Definition at line 51 of file mainloop_good.c.

6.3.2.14 #define MSG_SLOWHEARTBEAT 1

Definition at line 46 of file mainloop_good.c.

6.3.2.15 #define MSG_TOOMANY 4

Definition at line 49 of file mainloop_good.c.

6.3.2.16 #define MSG_WRITE 7

Definition at line 52 of file mainloop_good.c.

6.3.2.17 #define PORT_CHARGEN 5006

tcp port to listen for chargen clients

Definition at line 28 of file mainloop_good.c.

6.3.2.18 #define PORT_ECHO 5005

tcp port to listen for echo clients

Definition at line 25 of file mainloop_good.c.

6.3.2.19 #define SLOWHEARTBEAT_INTERVAL 15

slow heartbeat interval (in seconds)

Definition at line 34 of file mainloop_good.c.

6.3.3 Typedef Documentation

6.3.3.1 typedef void(* alarmhandler_t)(void)

alarm handler

Definition at line 106 of file mainloop_good.c.

6.3.3.2 typedef void(* eventhandler_t)(int i)

event handler

Parameters:

i index into clients

Definition at line 84 of file mainloop_good.c.

6.3.4 Function Documentation

6.3.4.1 void acceptchargen (int *i*) [static]

accept connection and set up new session for "chargen" service

Parameters:

i index into pollfds/clients array for server descriptor

Definition at line 680 of file mainloop_good.c.

References addclient(), clients, closeclient(), and writechargen().

Referenced by main().

```
681 {
682     int             fd;
683     socklen_t       t;
684     struct sockaddr_in sain;
685 
686     t = sizeof(sain);
687     if ((fd = accept(clients[i].fd, (void *) &sain, &t)) == -1) {
688         if (errno == EWOULDBLOCK) {
689             return;
690         }
691         perror("accept(CHARGEN)");
692         exit(EXIT_FAILURE);
693     }
694     addclient(fd, NULL, &writechargen, &closeclient);
695 }
```

6.3.4.2 void acceptecho (int *i*) [static]

accept connection and set up new session for "echo" service

Parameters:

i index into pollfds/clients array for server descriptor

Definition at line 524 of file mainloop_good.c.

References addclient(), clients, closeclient(), message(), MSG_ACCEPT, readecho(), and writeecho().

Referenced by main().

```
525 {
526     int             fd;
527     socklen_t       t;
528     struct sockaddr_in sain;
529 
530     t = sizeof(sain);
531     if ((fd = accept(clients[i].fd, (void *) &sain, &t)) == -1) {
532         if (errno == EWOULDBLOCK) {
533             return;
534         }
535         perror("accept(ECHO)");
536         exit(EXIT_FAILURE);
537     }
538     message(MSG_ACCEPT);
539     addclient(fd, &readecho, &writeecho, &closeclient);
540 }
```

6.3.4.3 void addalarm (alarmhandler_t *handler*, long *interval*) [static]

add new alarm

Parameters:

handler alarm handler procedure

interval interval in seconds

Bug

no error handling, if there are no more error handler slots, the new alarm is ignored...

Definition at line 263 of file mainloop_good.c.

References alarmhandler(), alarms, alarm_t::flag, alarm_t::handler, alarm_t::interval, MAXALARMS, and alarm_t::nexttime.

Referenced by main().

```
264 {
265     struct timeval  now;
266     int             i;
267 
268     for (i = 0; i < MAXALARMS; i++) {
269         if (alarms[i].handler == NULL) {
270             gettimeofday(&now, NULL);
271             alarms[i].handler = handler;
272             alarms[i].interval = interval;
273             alarms[i].nexttime = now.tv_sec + interval;
274             alarms[i].flag = 0;
275             alarmhandler();
276             return;
277         }
278     }
279 }
```

6.3.4.4 void addclient (int *fd*, eventhandler_t *read*, eventhandler_t *write*, eventhandler_t *except*) [static]

add new client

Parameters:

fd file descriptor asociated with this client

read handler called if socket is readable

write handler called if socket is writable

except handler called on exceptions (HUP, close etc).

Definition at line 437 of file mainloop_good.c.

References clients, client_t::except, client_t::fd, MAXCLIENTS, message(), MSG_TOOMANY, npollfd, pollfds, client_t::read, and client_t::write.

Referenced by acceptchargen(), acceptecho(), and main().

```
439 {
440     if (npollfd >= (MAXCLIENTS - 1)) {
441         close(fd);
```

```
442         message(MSG_TOOMANY);
443         return;
444     }
445     clients[npollfd].fd = fd;
446     clients[npollfd].read = read;
447     clients[npollfd].write = write;
448     clients[npollfd].except = except;
449     pollfds[npollfd].fd = fd;
450     pollfds[npollfd].events = 0;
451     if (read) {
452         pollfds[npollfd].events |= POLLIN;
453     }
454     if (write) {
455         pollfds[npollfd].events |= POLLOUT;
456     }
457     npollfd++;
458 }
```

6.3.4.5 void alarmhandler (void) [static]

signal handler for alarms

mark expired alarms for execution and set new alarm timer.

the resolution of the alarm timer is one second, no attempt is made to get timing beyond this one second resolution (any sub-second timing information is discarded).

Definition at line 307 of file mainloop_good.c.

References alarms, alarm_t::flag, alarm_t::handler, alarm_t::interval, MAX, MAXALARMS, MIN, and alarm_t::nexttime.

Referenced by addalarm(), and initalarms().

```
308 {
309     struct timeval  now;
310     long            nextalarm;
311     int             i;
312 
313     gettimeofday(&now, NULL);
314 
315     nextalarm = 65536;
316     for (i = 0; i < MAXALARMS; i++) {
317         if (alarms[i].handler != NULL) {
318             if (alarms[i].nexttime <= now.tv_sec) {
319                 alarms[i].flag = 1;
320                 alarms[i].nexttime += alarms[i].interval;
321             }
322             nextalarm =
323                 MIN(nextalarm,
324                     MAX(1,
325                         (unsigned int) alarms[i].nexttime - now.tv_sec));
326         }
327     }
328     alarm((unsigned int) nextalarm);
329 }
```

6.3.4.6 void blocksigpipe (void) [static]

block SIGPIPE

Trying to write to a socket when the other end has already closed the connection results in SIGPIPE. Not usefull in this context.

Definition at line 214 of file mainloop_good.c.

```
215 {
216     struct sigaction act;
217 
218     act.sa_handler = SIG_IGN;
219     sigemptyset(&act.sa_mask);
220     act.sa_flags = SA_RESTART;
221     if (sigaction(SIGPIPE, &act, NULL) == -1) {
222         perror("sigaction(SIGPIPE, <ignore>)");
223         exit(EXIT_FAILURE);
224     }
225 }
```

6.3.4.7 void checkalarms (void) [static]

check for pending alarms and execute alarm handlers

Definition at line 285 of file mainloop_good.c.

References alarms, alarm_t::flag, alarm_t::handler, and MAXALARMS.

Referenced by mainloop().

```
286 {
287     int             i;
288 
289     for (i = 0; i < MAXALARMS; i++) {
290         if (alarms[i].flag) {
291             (*alarms[i].handler) ();
292             alarms[i].flag = 0;
293         }
294     }
295 }
```

6.3.4.8 void closeclient (int *i*) [static]

close and delete client session

Parameters:

i index into pollfds/clients array

Definition at line 666 of file mainloop_good.c.

References clients, delclient(), client_t::fd, message(), and MSG_CLOSE.

Referenced by acceptchargen(), acceptecho(), readecho(), writechargen(), and writeecho().

```
667 {
668     message(MSG_CLOSE);
669     close(clients[i].fd);
670     delclient(i);
671     clients[i].fd = -1;
672 }
```

6.3.4.9 void delclient (int *i*) [static]

delete client

Parameters:
i index into pollfds/clients array

Definition at line 466 of file mainloop_good.c.

References clients, client_t::fd, npollfd, and pollfds.

Referenced by closeclient().

```
467 {
468     clients[i].fd = -1;
469     if (i != npollfd) {
470         /* just copy the whole thing including the buffer... */
471         memcpy(&clients[i], &clients[npollfd], sizeof(clients[i]));
472         pollfds[i] = pollfds[npollfd];
473         npollfd--;
474     }
475 }
```

6.3.4.10 void flowecho (int *i*) [static]

do "flow" control for echo

Parameters:
i index into pollfds/clients array for echo session

turn off checking for read if buffer is full resp. write if buffer is empty.

Definition at line 646 of file mainloop_good.c.

References BUFSIZE, clients, client_t::echo, echo_client_t::n, and pollfds.

Referenced by readecho(), and writeecho().

```
647 {
648     switch (clients[i].echo.n) {
649     case 0:
650         pollfds[i].events = POLLIN;
651         break;
652     case BUFSIZE:
653         pollfds[i].events = POLLOUT;
654         break;
655     default:
656         pollfds[i].events = POLLIN | POLLOUT;
657     }
658 }
```

6.3.4.11 void heartbeat (void) [static]

print heartbeat message

Definition at line 335 of file mainloop_good.c.

References message(), and MSG_HEARTBEAT.

Referenced by main().

```
336 {
337     message(MSG_HEARTBEAT);
338 }
```

6.3.4.12 void initalarms (void) [static]

install non-resetting signal handler for alarms

The sigaction interface allows installing non-resetting signal handlers (ie not reset to SIG_DFL after disposition). This way, there is no race condition when reinstalling the signal handler.

Definition at line 235 of file mainloop_good.c.

References alarmhandler(), alarms, alarm_t::handler, and MAXALARMS.

Referenced by main().

```
236 {
237     struct sigaction act;
238     int             i;
239 
240     act.sa_handler = alarmhandler;
241     sigemptyset(&act.sa_mask);
242     act.sa_flags = 0;           /* no SIG_RESTART! */
243     if (sigaction(SIGALRM, &act, NULL) == -1) {
244         perror("sigaction(SIGPIPE, <ignore>)");
245         exit(EXIT_FAILURE);
246     }
247     for (i = 0; i < MAXALARMS; i++) {
248         alarms[i].handler = NULL;
249     }
250     alarm(0);                   /* cancel any previously made alarm request */
251 }
```

6.3.4.13 void initclients () [static]

initialize client state array

Definition at line 418 of file mainloop_good.c.

References clients, client_t::fd, MAXCLIENTS, and npollfd.

Referenced by main().

```
419 {
420     int             i;
421 
422     for (i = 0; i < MAXCLIENTS; i++) {
423         clients[i].fd = -1;
424     }
425     npollfd = 0;
426 }
```

6.3.4.14 int listensocket (int *port*) [static]

return tcp socket listening on port specified

Parameters:

port port number in host byte order

Returns:
file descriptor for listening socket

The socket is set to be nonblocking

Definition at line 359 of file mainloop_good.c.

References setnonblock().

Referenced by listensource(), and main().

```
360 {
361     int             serverfd;
362     struct sockaddr_in sain;
363     int             one;
364 
365     /* set up tcp socket for listening */
366     sain.sin_family = AF_INET;
367     sain.sin_port = htons(port);
368     sain.sin_addr.s_addr = INADDR_ANY;
369     if ((serverfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
370         perror("socket(AF_INET, SOCK_STREAM, 0)");
371         exit(EXIT_FAILURE);
372     }
373     one = 1;
374     if (setsockopt
375         (serverfd, SOL_SOCKET, SO_REUSEADDR, &one,
376          (int) sizeof(one)) == -1) {
377         perror("setsockopt(SO_REUSEADDR)");
378         exit(EXIT_FAILURE);
379     }
380     if (bind
381         (serverfd, (struct sockaddr *) &sain, sizeof(struct sockaddr_in))
382         == -1) {
383         perror("bind()");
384         exit(EXIT_FAILURE);
385     }
386     if (listen(serverfd, 5) == -1) {
387         perror("listen()");
388         exit(EXIT_FAILURE);
389     }
390     setnonblock(serverfd);
391 
392     printf("listening on port %d\n", port);
393 
394     return serverfd;
395 }
```

6.3.4.15 int main ()

Definition at line 734 of file mainloop_good.c.

References acceptchargen(), acceptecho(), addalarm(), addclient(), blocksigpipe(), heartbeat(), HEARTBEAT_INTERVAL, initalarms(), initclients(), listensocket(), mainloop(), PORT_CHARGEN, PORT_ECHO, slowheartbeat(), and SLOWHEARTBEAT_INTERVAL.

```
735 {
736     printf("example: better main loop\n");
737 
738     blocksigpipe();
739 
740     initclients();
741     addclient(listensocket(PORT_ECHO), &acceptecho, NULL, NULL);
```

```
742     addclient(listensocket(PORT_CHARGEN), &acceptchargen, NULL, NULL);
743     initalarms();
744     addalarm(heartbeat, HEARTBEAT_INTERVAL);
745     addalarm(slowheartbeat, SLOWHEARTBEAT_INTERVAL);
746     printf("heartbeat every %d seconds\n", HEARTBEAT_INTERVAL);
747     printf("slow heartbeat every %d seconds\n", SLOWHEARTBEAT_INTERVAL);
748
749     /* main loop */
750     while (1) {
751         mainloop();
752     }
753     /* notreached */
754 }
```

6.3.4.16 void mainloop () [static]

execute one iteration of THE mainloop

Definition at line 481 of file mainloop_good.c.

References checkalarms(), clients, client_t::except, message(), MSG_MAINLOOP, npollfd, pollfds, client_t::read, and client_t::write.

Referenced by main().

```
482 {
483     int             i;
484
485     message(MSG_MAINLOOP);
486     i = poll(pollfds, npollfd, -1);
487
488     /* check for alarm handlers to be executed */
489     checkalarms();
490
491     if (i == -1) {
492         if ((errno != EAGAIN) && (errno != EINTR)) {
493             perror("poll()");
494             exit(EXIT_FAILURE);
495         }
496         return;
497     }
498
499     /*
500      * handle i/o events. work off exceptions first. then read (try to fill
501      * buffer) and write (try to empty buffer)
502      */
503     for (i = 0; i <= npollfd; i++) {
504         if ((clients[i].except)
505             && (pollfds[i].revents & (POLLERR | POLLHUP | POLLNVAL))) {
506             (*clients[i].except) (i);
507             continue;
508         }
509         if ((clients[i].read) && (pollfds[i].revents & POLLIN)) {
510             (*clients[i].read) (i);
511         }
512         if ((clients[i].write) && (pollfds[i].revents & POLLOUT)) {
513             (*clients[i].write) (i);
514         }
515     }
516 }
```

6.3.4.17 void message (int *msg*) [static]

print message describing current activity

Parameters:

msg id of message to print (MSG_XYZ)

Definition at line 170 of file mainloop_good.c.

References MSG_ACCEPT, MSG_CLOSE, MSG_EMPTY, MSG_FULL, MSG_HEARTBEAT, MSG_-MAINLOOP, MSG_READ, MSG_SLOWHEARTBEAT, MSG_TOOMANY, and MSG_WRITE.

Referenced by accept_dispatch(), acceptecho(), addclient(), chargen_dispatch(), closeclient(), echo_-dispatch_read(), echo_dispatch_write(), heartbeat(), main(), mainloop(), readecho(), sighandler(), slow-heartbeat(), source_close(), writechargen(), and writeecho().

```
171 {
172     switch (msg) {
173     case MSG_HEARTBEAT:
174         printf("H");
175         break;
176     case MSG_SLOWHEARTBEAT:
177         printf("S");
178         break;
179     case MSG_MAINLOOP:
180         /* printf("M"); */
181         break;
182     case MSG_ACCEPT:
183         printf("A");
184         break;
185     case MSG_TOOMANY:
186         printf("T");
187         break;
188     case MSG_CLOSE:
189         printf("C");
190         break;
191     case MSG_READ:
192         printf("R");
193         break;
194     case MSG_WRITE:
195         printf("W");
196         break;
197     case MSG_FULL:
198         printf("F");
199         break;
200     case MSG_EMPTY:
201         printf("E");
202         break;
203     }
204     fflush(stdout);
205 }
```

6.3.4.18 void readecho (int *i*) [static]

read data from echo client

Parameters:

i index into pollfds/clients array for echo session

If the buffer is not full, tries to do one read from the filedescriptor associated with this echo client.

Definition at line 551 of file mainloop_good.c.

References BUFSIZE, clients, closeclient(), client_t::echo, flowecho(), message(), MSG_FULL, MSG_-READ, echo_client_t::n, echo_client_t::r, and echo_client_t::w.

Referenced by acceptecho(), and main().

```
552 {
553     ssize_t         nread;
554 
555     if (clients[i].echo.n == BUFSIZE) {
556         message(MSG_FULL);
557         flowecho(i);
558         return;
559     }
560     if (clients[i].echo.r >= clients[i].echo.w) {
561         nread =
562             read(clients[i].fd, clients[i].echo.buf + clients[i].echo.r,
563                  BUFSIZE - clients[i].echo.r);
564     } else {
565         nread =
566             read(clients[i].fd, clients[i].echo.buf + clients[i].echo.r,
567                  clients[i].echo.w - clients[i].echo.r);
568     }
569 
570     switch (nread) {
571     case -1:
572         if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
573             perror("read()");
574             exit(EXIT_FAILURE);
575         }
576         break;
577     case 0:
578         closeclient(i);
579         return;
580     default:
581         message(MSG_READ);
582         clients[i].echo.n += nread;
583         clients[i].echo.r += nread;
584         clients[i].echo.r %= BUFSIZE;
585         flowecho(i);
586     }
587 }
```

6.3.4.19 void setnonblock (int *fd*) [static]

set a file descriptor to be nonblocking

Parameters:

fd file descriptor

Non-Blocking works only for Sockets, Pipes and slow devices, it has no effect when used with regular files or "fast" devices.

Definition at line 406 of file mainloop_good.c.

Referenced by accept_dispatch(), listensocket(), and main().

```
407 {
408     int             flag;
409 
410     flag = fcntl(fd, F_GETFL);
```

```
411     fcntl(fd, F_GETFL, flag | O_NONBLOCK);
412 }
```

6.3.4.20 void slowheartbeat (void) [static]

print "slowheartbeat" message

Definition at line 344 of file mainloop_good.c.

References message(), and MSG_SLOWHEARTBEAT.

Referenced by main().

```
345 {
346     message(MSG_SLOWHEARTBEAT);
347 }
```

6.3.4.21 void writechargen (int *i*) [static]

write data to an chargen client

Parameters:

i index into pollfds/clients array for chargen session

If the buffer is not empty, tries to do one write to the filedescriptor associated with the chargen client.

Definition at line 706 of file mainloop_good.c.

References client_t::chargen, chargen_buf, clients, closeclient(), chargen_client_t::i, message(), and MSG_WRITE.

Referenced by acceptchargen(), and main().

```
707 {
708     ssize_t         nwrite;
709 
710     nwrite =
711         write(clients[i].fd, chargen_buf + clients[i].chargen.i,
712               sizeof(chargen_buf) - clients[i].chargen.i);
713 
714     switch (nwrite) {
715     case -1:
716         if (errno == EPIPE) {
717             closeclient(i);
718             return;
719         } else if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
720             perror("write()");
721             exit(EXIT_FAILURE);
722         }
723         break;
724     case 0:
725         break;
726     default:
727         message(MSG_WRITE);
728         clients[i].chargen.i += nwrite;
729         clients[i].chargen.i %= sizeof(chargen_buf);
730     }
731 }
```

6.3.4.22 void writeecho (int *i*) [static]

write data to an echo client

Parameters:
i index into pollfds/clients array for echo session

If the buffer is not empty, tries to do one write to the filedescriptor associated with the echo client.

Definition at line 598 of file mainloop_good.c.

References BUFSIZE, clients, closeclient(), client_t::echo, flowecho(), message(), MSG_EMPTY, MSG_-WRITE, echo_client_t::n, echo_client_t::r, and echo_client_t::w.

Referenced by acceptecho(), and main().

```
599 {
600     ssize_t         nwrite;
601 
602     if (clients[i].echo.n == 0) {
603         message(MSG_EMPTY);
604         flowecho(i);
605         return;
606     }
607     if (clients[i].echo.r > clients[i].echo.w) {
608         nwrite =
609             write(clients[i].fd, clients[i].echo.buf + clients[i].echo.w,
610                   clients[i].echo.r - clients[i].echo.w);
611     } else {
612         nwrite =
613             write(clients[i].fd, clients[i].echo.buf + clients[i].echo.w,
614                   BUFSIZE - clients[i].echo.w);
615     }
616 
617     switch (nwrite) {
618     case -1:
619         if (errno == EPIPE) {
620             closeclient(i);
621         } else if ((errno != EINTR) && (errno != EWOULDBLOCK)) {
622             perror("write()");
623             exit(EXIT_FAILURE);
624         }
625         break;
626     case 0:
627         break;
628     default:
629         message(MSG_WRITE);
630         clients[i].echo.n -= nwrite;
631         clients[i].echo.w += nwrite;
632         clients[i].echo.w %= BUFSIZE;
633         flowecho(i);
634     }
635 }
```

6.3.5 Variable Documentation

6.3.5.1 alarm_t alarms[MAXALARMS] [static]

alarms

Definition at line 134 of file mainloop_good.c.

Referenced by addalarm(), alarmhandler(), checkalarms(), and initalarms().

6.3.5.2 char chargen_buf[ ] = "0123456789abcdefghijklmnopqrstuv" [static]

characters to return in chargen service

Definition at line 60 of file mainloop_good.c.

Referenced by writechargen().

6.3.5.3 client_t clients[MAXCLIENTS] [static]

array of client states

Definition at line 132 of file mainloop_good.c.

Referenced by acceptchargen(), acceptecho(), addclient(), closeclient(), delclient(), flowecho(), initclients(), mainloop(), readecho(), writechargen(), and writeecho().

6.3.5.4 unsigned long npollfd [static]

number of file descriptors to check

this is not as bad as it sounds because there's a static array for the struct pollfd anyway (pollfds). using an array for the client states allows using the same index for pollfds and clients.

in a real application, the buffer would be bigger, and therefore wouldn't be placed into the array ;) for demonstration purposes, the buffer is small to make it more interesting

Definition at line 131 of file mainloop_good.c.

Referenced by addclient(), delclient(), initclients(), and mainloop().

6.3.5.5 struct pollfd pollfds[MAXCLIENTS] [static]

pollfds for poll()

Definition at line 133 of file mainloop_good.c.

Referenced by addclient(), delclient(), flowecho(), and mainloop().

6.4 README File Reference

Chapter 7

mainloop Page Documentation

7.1 Bug List

Global addalarm(alarmhandler_t handler, long interval) no error handling, if there are no more error handler slots, the new alarm is ignored...

Index