Simple Sockets Library

- You may obtain this library from my website: http://www.drchip.org/astronaut/ssl

Simple Sockets Library

- You may obtain this library from my website: http://www.drchip.org/astronaut/ssl
- Written to make using connection-oriented (tcp) sockets simple

Simple Sockets Library

- You may obtain this library from my website: http://www.drchip.org/astronaut/ssl
- Written to make using connection-oriented (tcp) sockets simple
- Written to resemble C’s file i/o:

  Sclose() Sgets() Sopen() Sprintf() Sputs() Sread() Sscanf() Swrite() Svprintf()

Simple Sockets Library

- You may obtain this library from my website: http://www.drchip.org/astronaut/ssl
- Written to make using connection-oriented (tcp) sockets simple
- Written to resemble C's file i/o:

  Sclose() Sgets() Sopen() Sprintf() Sputs() Sread() Sscanf() Swrite() Svprintf()
- Uses automatically assigned ports, so if a server can be brought up, it will be

Simple Sockets Library

- You may obtain this library from my website: http://www.drchip.org/astronaut/ssl
- Written to make using connection-oriented (tcp) sockets simple
- Written to resemble C's file i/o:

  Sclose() Sgets() Sopen() Sprintf() Sputs() Sread() Sscanf() Swrite() Svprintf()
- Uses automatically assigned ports, so if a server can be brought up, it will be
- Servers have names, not fixed ports (solves REUSEADDR problems, linger setting, zombie sockets)

SSL: Sopen()

```
#include "sockets.h"
Socket *Sopen(char *skthost, char *mode)
```

skthost Servers: name of socket.

Clients: name of socket @ hostname

SSL: Sopen()

#include “sockets.h”
Socket *Sopen(char *skthost, char *mode)

skthost Servers: name of socket.
Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

SSL: Sopen()

```
#include "sockets.h"
Socket *Sopen(char *skthost, char *mode)
```

skthost Servers: name of socket.

Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

- **s** open a server

SSL: Sopen()

#include “sockets.h”

Socket *Sopen(char *skthost, char *mode)

skthost Servers: name of socket.

Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

s open a server

S open a server (but, on failure, retry)

SSL: Sopen()

#include “sockets.h”

Socket *Sopen(char *skthost, char *mode)

skthost Servers: name of socket.

Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

s open a server

S open a server (but, on failure, retry)

c open a client

SSL: Sopen()

#include “sockets.h”

Socket *Sopen(char *skthost, char *mode)

skthost Servers: name of socket.

Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

s open a server

S open a server (but, on failure, retry)

c open a client

s### open a server with the specified port number

SSL: Sopen()

#include “sockets.h”

Socket *Sopen(char *skthost, char *mode)

skthost Servers: name of socket.

Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

s open a server

S open a server (but, on failure, retry)

c open a client

s### open a server with the specified port number

c### open a client to a server with the specified port number

SSL: Sopen()

```
#include “sockets.h”
Socket *Sopen(char *skthost, char *mode)
```

skthost Servers: name of socket.

Clients: name of socket @ hostname

mode Tells Sopen the type of socket being opened

- **s** open a server
- **S** open a server (but, on failure, retry)
- **c** open a client
- **s###** open a server with the specified port number
- **c###** open a client to a server with the specified port number

returns Servers: a 3-tuple Socket* - use Saccept() with it to get a full duplex communications socket

Clients: a 5-tuple Socket*

SSL: Saccept()

Socket *Saccept(Socket *skt)

skt a Socket* server, opened by Sopen()

SSL: Saccept()

Socket *Saccept(Socket *skt)

skt a Socket* server, opened by Sopen()

returns a full 5-tuple Socket* accept socket, useful for full duplex communications

SSL: Sclose()

void Sclose(Socket *skt)

Use this function to close a Sopen()ed Socket.

SSL: Single Server Example

```
#include "sockets.h"
Socket *server;
Socket *skt;
server= Sopen("servername","s");
skt   = Saccept(server);
...
Sclose(skt);
Sclose(server);
```

SSL: Multiple Accepts Example

```
#include "sockets.h"
Socket *server= NULL;
Socket *skt    = NULL;
server= Sopen("servername","s");
do {
  skt= Saccept(server);
  ...
  Sclose(skt);
  } while(whatever);
Sclose(server);
```

SSL: A Client Example

```
#include "sockets.h"
Socket *client;
while(1) {
   client= Sopen("servername","c");
   if(client) break;
   sleep(1);
   }
...
Sclose(client);
```

- If the “servername” server is on the local machine, just the name may be used
- For servers on non-local machines, use “servername@hostname”

SSL: I/O

Return Type	*Function Name*	*Argument List*
char *	Sgets	(char *buf, int maxbuf, Socket *skt)
int	Speek	(Socket *skt, char *buf, int buflen)
void	Sprintf	(Socket *skt, char *fmt,...)
void	Sputs	(char *buf, Socket *skt)
int	Sread	(Socket *skt, char *buf, int buflen)
int	Sreadbytes	(Socket *skt, char *buf, int buflen)
int	Sscanf	(Socket *skt, char *fmt, ...)
int	Svprintf	(Socket *skt, char *fmt, void *args)

- Sread() reads up to buflen bytes, returning only whatever is currently present on the Socket

SSL: I/O

Return Type	*Function Name*	*Argument List*
char *	Sgets	(char *buf, int maxbuf, Socket *skt)
int	Speek	(Socket *skt, char *buf, int buflen)
void	Sprintf	(Socket *skt, char *fmt,...)
void	Sputs	(char *buf, Socket *skt)
int	Sread	(Socket *skt, char *buf, int buflen)
int	Sreadbytes	(Socket *skt, char *buf, int buflen)
int	Sscanf	(Socket *skt, char *fmt, ...)
int	Svprintf	(Socket *skt, char *fmt, void *args)

- Sread() reads up to buflen bytes, returning only whatever is currently present on the Socket
- Sreadbytes() requires buflen bytes, and will block until it gets them.

(see ssl.c)

Input/Output Multiplexing

How to handle multiple inputs:

1. polling : use fcntl() with FNDELAY or ioctl with FIONBIO to set the socket/file descriptor to non-blocking.
   Repeatedly examine the descriptor for new data (ie. attempt to read).
   Wastes cpu.

Input/Output Multiplexing

How to handle multiple inputs:

1. polling : use fcntl() with FNDELAY or ioctl with FIONBIO to set the socket/file descriptor to non-blocking.
   Repeatedly examine the descriptor for new data (ie. attempt to read).
   Wastes cpu.

2. Use fork() and assign a separate process to handle each stream/descriptor.
   Works with pipes, fifos, message queues, sockets, etc.
   Use semaphores, shared memory, and signals to communicate between parent and child.
   May implement callback functions.

Input/Output Multiplexing

How to handle multiple inputs:

1. polling : use fcntl() with FNDELAY or ioctl with FIONBIO to set the socket/file descriptor to non-blocking.
   Repeatedly examine the descriptor for new data (ie. attempt to read).
   Wastes cpu.
2. Use fork() and assign a separate process to handle each stream/descriptor.
   Works with pipes, fifos, message queues, sockets, etc.
   Use semaphores, shared memory, and signals to communicate between parent and child.
   May implement callback functions.
3. Use threads; otherwise much like Method#2.

Input/Output Multiplexing

How to handle multiple inputs:

1. polling : use fcntl() with FNDELAY or ioctl with FIONBIO to set the socket/file descriptor to non-blocking.
   Repeatedly examine the descriptor for new data (ie. attempt to read).
   Wastes cpu.

2. Use fork() and assign a separate process to handle each stream/descriptor.
   Works with pipes, fifos, message queues, sockets, etc.
   Use semaphores, shared memory, and signals to communicate between parent and child.
   May implement callback functions.

3. Use threads; otherwise much like Method#2.

4. Use select() (BSD) or poll() (SysV, streams only). This is a powerful method for multiplexing sockets and other file descriptors – call select or poll and put your process/thread to sleep until an event occurs (typically, receipt of data or a timeout).

Multi I/O: select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

- select() works with “file descriptor sets”
  Blocks until one or more of the file descriptors becomes “ready” for i/o

Multi I/O: select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

- select() works with “file descriptor sets”
  Blocks until one or more of the file descriptors becomes “ready” for i/o
- FD_ZERO: clear all bits in the set

Multi I/O: select()

#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);

- select() works with “file descriptor sets”
  Blocks until one or more of the file descriptors becomes “ready” for i/o
- FD_ZERO: clear all bits in the set
- FD_CLR: remove the specified file descriptor fd from the set (ie. off)

Multi I/O: select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

- select() works with “file descriptor sets”
  Blocks until one or more of the file descriptors becomes “ready” for i/o
- FD_ZERO: clear all bits in the set
- FD_CLR: remove the specified file descriptor fd from the set (ie. off)
- FD_SET: add the specified file descriptor fd to the set (ie. on)

Multi I/O: select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

- select() works with “file descriptor sets”
  Blocks until one or more of the file descriptors becomes “ready” for i/o
- FD_ZERO: clear all bits in the set
- FD_CLR: remove the specified file descriptor fd from the set (ie. off)
- FD_SET: add the specified file descriptor fd to the set (ie. on)
- FD_ISSET: query if the bit associated with the specified file descriptor is set or not

Multi I/O: select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

- select() works with “file descriptor sets”
  Blocks until one or more of the file descriptors becomes “ready” for i/o
- FD_ZERO: clear all bits in the set
- FD_CLR: remove the specified file descriptor fd from the set (ie. off)
- FD_SET: add the specified file descriptor fd to the set (ie. on)
- FD_ISSET: query if the bit associated with the specified file descriptor is set or not
- fd_set: supports 32 to 1024 bits arranged in an array of long integers (via unions), though few kernels will actually support that many open files.

Multi I/O: select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

- select() works with “file descriptor sets”

  Blocks until one or more of the file descriptors becomes “ready” for i/o
- FD_ZERO: clear all bits in the set
- FD_CLR: remove the specified file descriptor fd from the set (ie. off)
- FD_SET: add the specified file descriptor fd to the set (ie. on)
- FD_ISSET: query if the bit associated with the specified file descriptor is set or not
- fd_set: supports 32 to 1024 bits arranged in an array of long integers (via unions), though few kernels will actually support that many open files.
- nfds: should be one more than the maximum file descriptor value over all the sets

Multi I/O: select(), con’t.

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
```

- timeout: specifies a maximum time for blocking

```
struct timeval {
        long tv_sec; // seconds
        long tv_usec; // microseconds
        };
```

Multi I/O: select(), con’t.

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);
```

- timeout: specifies a maximum time for blocking

  ```
  struct timeval {
          long tv_sec; // seconds
          long tv_usec; // microseconds
          };
  ```

- A timeout with zeros for tv_sec and tv_usec set returns immediately

Multi I/O: select(), con't.

#include <sys/select.h>

int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);

- timeout: specifies a maximum time for blocking

  ```
  struct timeval {
          long tv_sec; // seconds
          long tv_usec; // microseconds
          };
  ```

- A timeout with zeros for tv_sec and tv_usec set returns immediately
- select() will update the timeout argument to indicate how much time was left

Multi I/O: select(), con't.

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval
*timeout);
```

- timeout: specifies a maximum time for blocking

  ```
  struct timeval {
          long tv_sec; // seconds
          long tv_usec; // microseconds
          };
  ```

- A timeout with zeros for tv_sec and tv_usec set returns immediately
- select() will update the timeout argument to indicate how much time was left
- readfds: determine if the file descriptors in this set are ready for reading (ie. a read will not block)

Multi I/O: select(), con’t.

#include <sys/select.h>

int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);

- timeout: specifies a maximum time for blocking

  ```
  struct timeval {
          long tv_sec; // seconds
          long tv_usec; // microseconds
          };
  ```

- A timeout with zeros for tv_sec and tv_usec set returns immediately
- select() will update the timeout argument to indicate how much time was left
- readfds: determine if the file descriptors in this set are ready for reading (ie. a read will not block)
- writefds: determine if the file descriptors in this set are ready for writing (ie. a write will not block)

Multi I/O: select(), con’t.

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,fd_set *exceptfds, struct timeval *timeout);
```

- timeout: specifies a maximum time for blocking

  ```
  struct timeval {
          long tv_sec; // seconds
          long tv_usec; // microseconds
          };
  ```

- A timeout with zeros for tv_sec and tv_usec set returns immediately
- select() will update the timeout argument to indicate how much time was left
- readfds: determine if the file descriptors in this set are ready for reading (ie. a read will not block)
- writefds: determine if the file descriptors in this set are ready for writing (ie. a write will not block)
- exceptfds: determine if the file descriptors in this set have any exceptions (out-of-band)

Multi I/O: select(), con’t.

- A null timeout means block forever (until a descriptor is ready for i/o)

Multi I/O: select(), con't.

- A null timeout means block forever (until a descriptor is ready for i/o)
- Note that signals will cause select() to return -1/EINTR

Multi I/O: select(), con't.

- A null timeout means block forever (until a descriptor is ready for i/o)
- Note that signals will cause select() to return -1/EINTR
- on exit, the sets are modified in place to indicate which file descriptors are ready for i/o

Multi I/O: select(), con’t.

- A null timeout means block forever (until a descriptor is ready for i/o)
- Note that signals will cause select() to return -1/EINTR
- on exit, the sets are modified in place to indicate which file descriptors are ready for i/o
- select() returns

+	the quantity of ready descriptors,
0	on timeout,
-1	on failure (with errno suitably set)

(see select.c)

Multi I/O: pselect()

```
int pselect(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, const struct timespec *timeout,
const sigset_t *sigmask);
```

- The pselect() function resembles select()

Multi I/O: pselect()

```
int pselect(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, const struct timespec *timeout,
const sigset_t *sigmask);
```

- The pselect() function resembles select()
- sigmask: a pointer to a signal mask

Multi I/O: pselect()

```
int pselect(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, const struct timespec *timeout,
const sigset_t *sigmask);
```

- The pselect() function resembles select()
- sigmask: a pointer to a signal mask
- pselect() first replaces the current signal mask with sigmask, then does the select() function, then restores the original signal mask.

Multi I/O: pselect()

int pselect(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, const struct timespec *timeout,
const sigset_t *sigmask);

- The pselect() function resembles select()
- sigmask: a pointer to a signal mask
- pselect() first replaces the current signal mask with sigmask, then does the select() function, then restores the original signal mask.
- Equivalent to *atomically* executing the following: *(sigprocmask() is used to fetch and/or change the signal mask of the calling thread)*

```
sigprocmask(SIG_SETMASK, &sigmask, &origmask);
ready = select(nfds, &readfds, &writefds, &exceptfds, timeout);
sigprocmask(SIG_SETMASK, &origmask, NULL);
```

Multi I/O: pselect()

int pselect(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, const struct timespec *timeout, const sigset_t *sigmask);

- The pselect() function resembles select()
- sigmask: a pointer to a signal mask
- pselect() first replaces the current signal mask with sigmask, then does the select() function, then restores the original signal mask.
- Equivalent to *atomically* executing the following: *(sigprocmask() is used to fetch and/or change the signal mask of the calling thread)*

```
sigprocmask(SIG_SETMASK, &sigmask, &origmask);
ready = select(nfds, &readfds, &writefds, &exceptfds, timeout);
sigprocmask(SIG_SETMASK, &origmask, NULL);
```

- This function is needed when one wants to handle both signals and file descriptors while preventing a race condition.

Multi I/O: pselect()

int pselect(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, const struct timespec *timeout, const sigset_t *sigmask);

- The pselect() function resembles select()
- sigmask: a pointer to a signal mask
- pselect() first replaces the current signal mask with sigmask, then does the select() function, then restores the original signal mask.
- Equivalent to *atomically* executing the following: *(sigprocmask() is used to fetch and/or change the signal mask of the calling thread)*

```
sigprocmask(SIG_SETMASK, &sigmask, &origmask);
ready = select(nfds, &readfds, &writefds, &exceptfds, timeout);
sigprocmask(SIG_SETMASK, &origmask, NULL);
```

- This function is needed when one wants to handle both signals and file descriptors while preventing a race condition.
- struct timespec supports seconds and nanoseconds

```
struct timespec {
        long tv_sec; // seconds
        long tv_nsec; // nanoseconds
```

Multi I/O: poll()

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- Functionality is similar to that of select()

Multi I/O: poll()

#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);

- Functionality is similar to that of select()
- fds the set of nfds file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

Multi I/O: poll()

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- Functionality is similar to that of select()
- fds the set of nfds file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

- Events

Event	*Input to*	*Result from*	*Description*
Name	*Events*	*Events*	*Description*
POLLIN	·	·	nonpriority msg present

Multi I/O: poll()

#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);

- Functionality is similar to that of `select()`
- `fds` the set of `nfds` file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

- Events

Event	*Input to*	*Result from*	*Description*
Name	*Events*	*Events*	*Description*
POLLIN	·	·	nonpriority msg present
POLLPRI	·	·	priority msg present

Multi I/O: poll()

#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);

- Functionality is similar to that of select()
- fds the set of nfds file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

- Events

Event	*Input to*	*Result from*	*Description*
Name	*Events*	*Events*	*Description*
POLLIN	·	·	nonpriority msg present
POLLPRI	·	·	priority msg present
POLLOUT	·	·	fd is writable

Multi I/O: poll()

#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);

- Functionality is similar to that of select()
- fds the set of nfds file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

- Events

Event	*Input to*	*Result from*	*Description*
Name	*Events*	*Events*	*Description*
POLLIN	·	·	nonpriority msg present
POLLPRI	·	·	priority msg present
POLLOUT	·	·	fd is writable
POLLERR		·	error msg arrived

Multi I/O: poll()

#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);

- Functionality is similar to that of select()
- fds the set of nfds file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

- Events

Event	*Input to*	*Result from*	*Description*
Name	*Events*	*Events*	*Description*
POLLIN	·	·	nonpriority msg present
POLLPRI	·	·	priority msg present
POLLOUT	·	·	fd is writable
POLLERR		·	error msg arrived
POLLHUP		·	hangup has occurred

Multi I/O: poll()

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- Functionality is similar to that of select()
- fds the set of nfds file descriptors to be monitored:

```
struct pollfd {
        int fd; // file descriptors to be checked
        short events; // (bitwise) events of interest on fd
        short revents; // (bitwise) events that occurred on fd
        };
```

- Events

Event	*Input to*	*Result from*	*Description*
Name	*Events*	*Events*	*Description*
POLLIN	·	·	nonpriority msg present
POLLPRI	·	·	priority msg present
POLLOUT	·	·	fd is writable
POLLERR		·	error msg arrived
POLLHUP		·	hangup has occurred
POLLNVAL		·	fd not an open stream

Multi I/O: poll(), con’t.

#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);

- timeout: - ⟶ block forever (until an event)

Multi I/O: poll(), con’t.

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- timeout: - ⟶ block forever (until an event)
- timeout: 0 ⟶ return immediately

Multi I/O: poll(), con’t.

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- timeout: - ⟶ block forever (until an event)
- timeout: 0 ⟶ return immediately
- timeout: + ⟶ wait this many milliseconds

Multi I/O: poll(), con’t.

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- timeout: - ⟶ block forever (until an event)
- timeout: 0 ⟶ return immediately
- timeout: + ⟶ wait this many milliseconds
- POLLIN: data ready to read

Multi I/O: poll(), con’t.

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- timeout: - ⟶ block forever (until an event)
- timeout: 0 ⟶ return immediately
- timeout: + ⟶ wait this many milliseconds
- POLLIN: data ready to read
- POLLPRI: urgent data ready to read (ie. out-of-band data on a tcp socket)

Multi I/O: poll(), con’t.

```
#include <poll.h>
int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

- timeout: - → block forever (until an event)
- timeout: 0 → return immediately
- timeout: + → wait this many milliseconds
- POLLIN: data ready to read
- POLLPRI: urgent data ready to read (ie. out-of-band data on a tcp socket)
- Returns

-1	on failure,
+	qty descriptors with non-zero revents field,
0	for timeout, no descriptors ready

(see poll.c)

Daemons

- A **daemon** is a process which is expected to execute for a long time.

Daemons

- A **daemon** is a process which is expected to execute for a long time.
- Often started when the system is brought up by `init`

Daemons

- A **daemon** is a process which is expected to execute for a long time.
- Often started when the system is brought up by init
- They have no controlling terminal. *(thus they aren't terminated by users logging in and out)*

Daemons

- A **daemon** is a process which is expected to execute for a long time.
- Often started when the system is brought up by init
- They have no controlling terminal. *(thus they aren't terminated by users logging in and out)*
- The Simple Sockets Library's PortMaster is an example of a daemon process *(that is not started by init)*

Daemons

- A **daemon** is a process which is expected to execute for a long time.
- Often started when the system is brought up by `init`
- They have no controlling terminal. *(thus they aren't terminated by users logging in and out)*
- The Simple Sockets Library's PortMaster is an example of a daemon process *(that is not started by init)*
- Use BSD: `ps -ajx` or SYSV: `ps -efjc` to see a list of such processes on your system.

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call `fork()` and exit. *(shells will then act as if the command is finished)*

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn’t start as a process group leader.
3. The nascent daemon then calls setsid() to create a new “session”
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group
   - daemon has no controlling terminal

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group
   - daemon has no controlling terminal
4. Change directory to the root directory

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group
   - daemon has no controlling terminal
4. Change directory to the root directory
   - allows mounted filesystems to be removed

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group
   - daemon has no controlling terminal
4. Change directory to the root directory
   - allows mounted filesystems to be removed
   - alternative: have a dedicated directory where the daemon's file-related work will be done

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group
   - daemon has no controlling terminal
4. Change directory to the root directory
   - allows mounted filesystems to be removed
   - alternative: have a dedicated directory where the daemon's file-related work will be done
5. Unneeded file descriptors the child inherited from the parent process should be closed

Daemon Coding Rules

1. Set file mode creation mask to zero (umask)
   - The inherited mask may override what the daemon sets
   - The daemon is thereby in control of all its read/write/execute permissions.
2. The parent process should call fork() and exit. *(shells will then act as if the command is finished)*
   - The child gets its own process ID, has a parent, and so doesn't start as a process group leader.
3. The nascent daemon then calls setsid() to create a new "session"
   - daemon becomes a session leader of a new session
   - daemon becomes a process group leader of a new process group
   - daemon has no controlling terminal
4. Change directory to the root directory
   - allows mounted filesystems to be removed
   - alternative: have a dedicated directory where the daemon's file-related work will be done
5. Unneeded file descriptors the child inherited from the parent process should be closed
6. *(optional)* use file and record locking *(lockfile())* to permit only one instance of the running daemon

Get/Set Resource Limits

```
#include <sys/time.h>
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

- These two functions allow one to set or get resource limits

Get/Set Resource Limits

```
#include <sys/time.h>
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

- These two functions allow one to set or get resource limits
- Each resource limit is given a soft and hard limit

Get/Set Resource Limits

```
#include <sys/time.h>
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

- These two functions allow one to set or get resource limits
- Each resource limit is given a soft and hard limit

  soft kernel enforces this limit

Get/Set Resource Limits

```
#include <sys/time.h>
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

- These two functions allow one to set or get resource limits
- Each resource limit is given a soft and hard limit

  soft kernel enforces this limit

  hard a ceiling for the soft limit

Get/Set Resource Limits

```
#include <sys/time.h>
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

- These two functions allow one to set or get resource limits
- Each resource limit is given a soft and hard limit

  soft kernel enforces this limit

  hard a ceiling for the soft limit
- struct rlimit {

  ```
  rlim_t rlim_cur; /* Soft limit */
  rlim_t rlim_max; /* Hard limit (ceiling for rlim_cur) */
  };
  ```

Get/Set Resource Limits

```
#include <sys/time.h>
#include <sys/resource.h>
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);
```

- These two functions allow one to set or get resource limits
- Each resource limit is given a soft and hard limit

  soft kernel enforces this limit

  hard a ceiling for the soft limit

-
  ```
  struct rlimit {
  rlim_t rlim_cur; /* Soft limit */
  rlim_t rlim_max; /* Hard limit (ceiling for rlim_cur) */
  };
  ```
- A sampling of resource limits is included on the next slide

Get/Set Resource Limits

- A sample of resource limits: *(see man getrlimit)*

RLIMIT_AS	max size of process's virtual memory, in bytes
RLIMIT_CORE	max size of a core file (when 0, no core dumps created)
RLIMIT_CPU	cpu time limit (sec)
RLIMIT_DATA	max size of process' data segment
RLIMIT_FSIZE	max size of files a process may create
RLIMIT_LOCKS	limits qty of of flock+fcntl locks
RLIMTI_MEMLOCK	max qty of memory (bytes) that may be locked into ram
RLIMIT_MSGQUEUE	max qty of bytes allocated for POSIX message queues
RLIMTI_NICE	ceiling for a process' nice value *(priority)*
RLIMIT_NOFILE	***specifies max file descriptor + 1 that may be opened***

Get/Set Resource Limits

- A sample of resource limits: *(see man getrlimit)*

RLIMIT_AS	max size of process’s virtual memory, in bytes
RLIMIT_CORE	max size of a core file (when 0, no core dumps created)
RLIMIT_CPU	cpu time limit (sec)
RLIMIT_DATA	max size of process’ data segment
RLIMIT_FSIZE	max size of files a process may create
RLIMIT_LOCKS	limits qty of of flock+fcntl locks
RLIMTI_MEMLOCK	max qty of memory (bytes) that may be locked into ram
RLIMIT_MSGQUEUE	max qty of bytes allocated for POSIX message queues
RLIMTI_NICE	ceiling for a process’ nice value *(priority)*
RLIMIT_NOFILE	***specifies max file descriptor + 1 that may be opened***

- So, for daemons: use getrlimit() + RLIMIT_NOFILE to inquire about the maximum qty of file descriptors that the daemon process may have inherited and close them all.

Daemon Example

```
void daemonize(const char *cmd) {
   int i,fd0,fd1,fd2;
   pid_t pid;
   struct rlimit rl;
   struct sigaction sa;
   umask(0);                            /* clear file creation mask      */
   if(getrlimit(RLIMIT_NOFILE,&rl) < 0) /* handle error                  */;
   if      ((pid= fork()) < 0)          /* handle error                  */;
   else if(pid != 0) exit(0);           /* parent process                */
   setsid();
   s.sa_handler= SIG_IGN;               /* don't allow controlling TTYs  */
   sigemptyset(&sa.sa_mask);
   sa.sa_flags= 0;
   if(sigaction(SIGHUP,&sa,NULL) < 0)   /* handle error                  */;
   if((pid= fork()) < 0)                /* handle error                  */;
   else if(pid != 0) exit(0);           /* parent                        */
   if(chdir("/") < 0)                   /* handle error                  */;
   /* close all open file descriptors */
   if(rl.rlim_max == RLIM_INFINIITY) rl.rlim_max= 1024;
   for(i= 0; i < rl.rlim_max; ++i) close(i);
   fd0= open("/dev/null",O_RDWR); fd1= dup(0); fd2= dup(0);
   openlog(cmd,LOG_CONS,LOG_DAEMON);
   if(fd0 != 0 || fd1 != 1 || fd2 != 2) {
      syslog(LOG_ERR,"unexpected␣file␣descriptors␣%d␣%d␣%d",fd0,fd1,fd2);
      exit(1);
      }
}
```

Error Logging: openlog

- Daemons can’t use stderr or stdout; *there’s no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include “syslog.h”
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

Error Logging: openlog

- Daemons can’t use stderr or stdout; *there’s no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include “syslog.h”
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

Error Logging: openlog

- Daemons can't use stderr or stdout; *there's no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include "syslog.h"
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

Error Logging: openlog

- Daemons can’t use stderr or stdout; *there’s no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

#include “syslog.h”
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

LOG_CONS	Write directly to the system console when unable to write to system logger

Error Logging: openlog

- Daemons can’t use stderr or stdout; *there’s no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include “syslog.h”
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

LOG_CONS	Write directly to the system console when unable to write to system logger
LOG_NDELAY	Open connection immediately

Error Logging: openlog

- Daemons can’t use stderr or stdout; *there’s no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

#include “syslog.h”
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

LOG_CONS	Write directly to the system console when unable to write to system logger
LOG_NDELAY	Open connection immediately
LOG_NOWAIT	Do not wait for child processes (safer for signal handling)

Error Logging: openlog

- Daemons can't use stderr or stdout; *there's no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include "syslog.h"
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

LOG_CONS	Write directly to the system console when unable to write to system logger
LOG_NDELAY	Open connection immediately
LOG_NOWAIT	Do not wait for child processes (safer for signal handling)
LOG_ODELAY	Delay connecting to syslogd until first message is logged

Error Logging: openlog

- Daemons can't use stderr or stdout; *there's no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include "syslog.h"
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

LOG_CONS	Write directly to the system console when unable to write to system logger
LOG_NDELAY	Open connection immediately
LOG_NOWAIT	Do not wait for child processes (safer for signal handling)
LOG_ODELAY	Delay connecting to syslogd until first message is logged
LOG_PERROR	Also write message to stderr

Error Logging: openlog

- Daemons can’t use stderr or stdout; *there’s no controlling terminal!*
- Solution: use syslog() to write messages to a *logging* socket on /dev/log

```
#include “syslog.h”
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon.

ident This string is prepended to every message (usually the name of the originating program).

option Controls writing target, connection handling, extra information

LOG_CONS	Write directly to the system console when unable to write to system logger
LOG_NDELAY	Open connection immediately
LOG_NOWAIT	Do not wait for child processes (safer for signal handling)
LOG_ODELAY	Delay connecting to syslogd until first message is logged
LOG_PERROR	Also write message to stderr
LOG_PID	Log pid with each message

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

- **facility** Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

Error Logging: openlog, con't.

#include "syslog.h"

void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

- **facility** Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc

Error Logging: openlog, con’t.

#include “syslog.h”
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)

Error Logging: openlog, con’t.

#include “syslog.h”
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon

Error Logging: openlog, con't.

#include "syslog.h"

void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages
LOG_LPR	line printer daemons (lpd, lpc, etc)

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages
LOG_LPR	line printer daemons (lpd, lpc, etc)
LOG_MAIL	mail system

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages
LOG_LPR	line printer daemons (lpd, lpc, etc)
LOG_MAIL	mail system
LOG_NEWS	usenet network news system

Error Logging: openlog, con't.

#include "syslog.h"
void openlog(const char *ident,int option,int facility)

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages
LOG_LPR	line printer daemons (lpd, lpc, etc)
LOG_MAIL	mail system
LOG_NEWS	usenet network news system
LOG_SYSLOG	messages from syslogd itself

Error Logging: openlog, con’t.

```
#include “syslog.h”
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages
LOG_LPR	line printer daemons (lpd, lpc, etc)
LOG_MAIL	mail system
LOG_NEWS	usenet network news system
LOG_SYSLOG	messages from syslogd itself
LOG_USER	messages from user processes *(default)*

Error Logging: openlog, con’t.

```
#include “syslog.h”
void openlog(const char *ident,int option,int facility)
```

openlog opens a connection to the syslogd daemon *(continued)*

facility Lets the configuration specify how messages from different facilities are to be handled. The following is a sample, not a complete list:

LOG_AUTH	authorization programs such as login, su, etc
LOG_CRON	cron and at
LOG_DAEMON	system daemons (inetd, routed, etc)
LOG_FTP	ftpd daemon
LOG_KERN	kernel messages
LOG_LPR	line printer daemons (lpd, lpc, etc)
LOG_MAIL	mail system
LOG_NEWS	usenet network news system
LOG_SYSLOG	messages from syslogd itself
LOG_USER	messages from user processes *(default)*
LOG_UUCP	uucp system

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

- **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

Error Logging: syslog

#include “syslog.h”
void syslog(int priority,const char *format,...)

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

- **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately
LOG_CRIT	critical condition *(ie. hard device error)*

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately
LOG_CRIT	critical condition *(ie. hard device error)*
LOG_ERR	error condition

Error Logging: syslog

#include “syslog.h”

void syslog(int priority,const char *format,...)

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately
LOG_CRIT	critical condition *(ie. hard device error)*
LOG_ERR	error condition
LOG_WARNING	warning condition

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately
LOG_CRIT	critical condition *(ie. hard device error)*
LOG_ERR	error condition
LOG_WARNING	warning condition
LOG_NOTICE	normal *but significant* condition

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately
LOG_CRIT	critical condition *(ie. hard device error)*
LOG_ERR	error condition
LOG_WARNING	warning condition
LOG_NOTICE	normal *but significant* condition
LOG_INFO	informational

Error Logging: syslog

```
#include “syslog.h”
void syslog(int priority,const char *format,...)
```

syslog Call this to generate a log message

priority Priorities; setlogmask() may be used to restrict logging to specified levels of priority

LOG_EMERG	emergency *(system unusable) (highest priority)*
LOG_ALERT	condition must be fixed immediately
LOG_CRIT	critical condition *(ie. hard device error)*
LOG_ERR	error condition
LOG_WARNING	warning condition
LOG_NOTICE	normal *but significant* condition
LOG_INFO	informational
LOG_DEBUG	debug message *(lowest priority)*

Error Logging: setlogmask

```
#include “syslog.h”
int setlogmask(int maskpri)
```

setlogmask **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

- Processes possess a *log priority mask* that filters which messages syslog() will actually log

Error Logging: setlogmask

#include "syslog.h"
int setlogmask(int maskpri)

setlogmask **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

- Processes possess a *log priority mask* that filters which messages syslog() will actually log
- Other calls to syslog() will be ignored.

Error Logging: setlogmask

#include “syslog.h”
int setlogmask(int maskpri)

setlogmask **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

- Processes possess a *log priority mask* that filters which messages syslog() will actually log
- Other calls to syslog() will be ignored.
- The mask has a bit assigned to each priority.

Error Logging: setlogmask

#include “syslog.h”
int setlogmask(int maskpri)

setlogmask **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

- Processes possess a *log priority mask* that filters which messages syslog() will actually log
- Other calls to syslog() will be ignored.
- The mask has a bit assigned to each priority.
- The function sets the new maskpri mask and returns the prior mask

Error Logging: setlogmask

#include “syslog.h”
int setlogmask(int maskpri)

setlogmask **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

- Processes possess a *log priority mask* that filters which messages syslog() will actually log
- Other calls to syslog() will be ignored.
- The mask has a bit assigned to each priority.
- The function sets the new maskpri mask and returns the prior mask
- Set up the mask by bitwise or-ing your selection of LOG_EMERG, . . . LOG_DEBUG priorities.

Error Logging: setlogmask

#include "syslog.h"

int setlogmask(int maskpri)

setlogmask **priority** Priorities; setlogmask() may be used to restrict logging to specified levels of priority

- Processes possess a *log priority mask* that filters which messages syslog() will actually log
- Other calls to syslog() will be ignored.
- The mask has a bit assigned to each priority.
- The function sets the new maskpri mask and returns the prior mask
- Set up the mask by bitwise or-ing your selection of LOG_EMERG, . . . LOG_DEBUG priorities.
- Setting the mask to 0 will have no effect. *(other than returning the current logging priority mask)*

Error Logging: closelog

```
#include “syslog.h”
void closelog(void)
```

closelog closes the descriptor being used to write to syslog. *(using this is optional)*

Error Logging: Conventions

If the daemon uses a lock file, its generally located in /var/run/name.pid, where name is the daemon's name

Error Logging: Conventions

- If the daemon uses a lock file, its generally located in /var/run/name.pid, where name is the daemon's name
- Daemon configuration options are usually stored in /etc/name.conf.

Error Logging: Conventions

If the daemon uses a lock file, its generally located in /var/run/name.pid, where name is the daemon's name

Daemon configuration options are usually stored in /etc/name.conf.

Daemons are usually started from one of the system initialization scripts *(/etc/rc* or /etc/init.d/*)*

Error Logging: Conventions

- If the daemon uses a lock file, its generally located in /var/run/name.pid, where name is the daemon's name
- Daemon configuration options are usually stored in /etc/name.conf.
- Daemons are usually started from one of the system initialization scripts *(/etc/rc* or /etc/init.d/*)*
- Daemons that need to restart automatically upon exiting can be restarted by including a respawn entry for it in /etc/inittab/name.conf.

Error Logging: Conventions

If the daemon uses a lock file, its generally located in /var/run/name.pid, where name is the daemon's name

Daemon configuration options are usually stored in /etc/name.conf.

Daemons are usually started from one of the system initialization scripts *(/etc/rc* or /etc/init.d/*)*

Daemons that need to restart automatically upon exiting can be restarted by including a respawn entry for it in /etc/inittab/name.conf.

Daemons typically use the SIGHUP signal to tell them to re-read their configuration file. *(as daemons don't have terminals this signal is safe them to use in this fashion)*

Unix Initialization

- Use who -r or runlevel from the command line to determine the current runlevel.

Unix Initialization

- Use who -r or runlevel from the command line to determine the current runlevel.
- Use init 1 to change to runlevel 1 *(as the superuser)*

Unix Initialization

- Use who -r or runlevel from the command line to determine the current runlevel.
- Use init 1 to change to runlevel 1 *(as the superuser)*
- To change the default runlevel to 5, for example, edit /etc/inittab and include the line

  id:5:initdefault:

  Save and close the file, then issue the reboot command

Unix Initialization

- Use `who -r` or `runlevel` from the command line to determine the current runlevel.
- Use `init 1` to change to runlevel 1 *(as the superuser)*
- To change the default runlevel to 5, for example, edit `/etc/inittab` and include the line

  ```
  id:5:initdefault:
  ```

  Save and close the file, then issue the `reboot` command
- Some services cannot be used until the system has certain other services running; runlevels provide this organization.

Unix Initialization

- Use `who -r` or `runlevel` from the command line to determine the current runlevel.
- Use `init 1` to change to runlevel 1 *(as the superuser)*
- To change the default runlevel to 5, for example, edit `/etc/inittab` and include the line

  ```
  id:5:initdefault:
  ```

  Save and close the file, then issue the `reboot` command
- Some services cannot be used until the system has certain other services running; runlevels provide this organization.
- Use `shutdown -h now` *(as superuser)* to bring the system down

Unix Initialization

- Use who -r or runlevel from the command line to determine the current runlevel.
- Use init 1 to change to runlevel 1 *(as the superuser)*
- To change the default runlevel to 5, for example, edit /etc/inittab and include the line

  ```
  id:5:initdefault:
  ```

  Save and close the file, then issue the reboot command
- Some services cannot be used until the system has certain other services running; runlevels provide this organization.
- Use shutdown -h now *(as superuser)* to bring the system down
- Use shutdown -r now *(as superuser)* to reboot the system

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*
1. Single User Mode

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*
1. Single User Mode
2. Multi-user *(without NFS, same as 3 without networking)*

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*
1. Single User Mode
2. Multi-user *(without NFS, same as 3 without networking)*
3. Full multi-user

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*
1. Single User Mode
2. Multi-user *(without NFS, same as 3 without networking)*
3. Full multi-user
4. unused

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*
1. Single User Mode
2. Multi-user *(without NFS, same as 3 without networking)*
3. Full multi-user
4. unused
5. X11

Unix Init: Runlevels

Runlevel The first six runlevels appear when Linux is “coming up”.

0. Halt *(do not set initdefault to this!)*
1. Single User Mode
2. Multi-user *(without NFS, same as 3 without networking)*
3. Full multi-user
4. unused
5. X11
6. reboot *(do not set initdefault to this!)*

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf
- In /etc/rc.d/rc#.d, where # is the current runlevel, are scripts which will be executed in alphabetical order. Consequently, they have names such as K46radvd, K50dnsmasq, etc: the leading letter+number combination provides sequence control.

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf
- In /etc/rc.d/rc#.d, where # is the current runlevel, are scripts which will be executed in alphabetical order. Consequently, they have names such as K46radvd, K50dnsmasq, etc: the leading letter+number combination provides sequence control.
- Each of these /etc/rc.d/rc#.d/* scripts control the starting and stopping of a service *(system daemon)*

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf
- In /etc/rc.d/rc#.d, where # is the current runlevel, are scripts which will be executed in alphabetical order. Consequently, they have names such as K46radvd, K50dnsmasq, etc: the leading letter+number combination provides sequence control.
- Each of these /etc/rc.d/rc#.d/* scripts control the starting and stopping of a service *(system daemon)*
- Scripts beginning with “K” are *kill* scripts; those starting with “S” are *start* scripts

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf
- In /etc/rc.d/rc#.d, where # is the current runlevel, are scripts which will be executed in alphabetical order. Consequently, they have names such as K46radvd, K50dnsmasq, etc: the leading letter+number combination provides sequence control.
- Each of these /etc/rc.d/rc#.d/* scripts control the starting and stopping of a service *(system daemon)*
- Scripts beginning with “K” are *kill* scripts; those starting with “S” are *start* scripts
- When the system begins a new runlevel, all the files beginning with “K” are executed, passing the “stop” parameter to the associated daemon

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf
- In /etc/rc.d/rc#.d, where # is the current runlevel, are scripts which will be executed in alphabetical order. Consequently, they have names such as K46radvd, K50dnsmasq, etc: the leading letter+number combination provides sequence control.
- Each of these /etc/rc.d/rc#.d/* scripts control the starting and stopping of a service *(system daemon)*
- Scripts beginning with "K" are *kill* scripts; those starting with "S" are *start* scripts
- When the system begins a new runlevel, all the files beginning with "K" are executed, passing the "stop" parameter to the associated daemon
- When the system begins a new runlevel, all the files beginning with "S" are then executed, passing the "start" parameter to the associated daemon

Unit Init: Starting/Stopping Daemons

- After the Linux kernel has booted, the init program reads /etc/inittab *(or /etc/init/rcS.conf)*
- Individual runlevels are then started by /etc/init/rc.conf
- In /etc/rc.d/rc#.d, where # is the current runlevel, are scripts which will be executed in alphabetical order. Consequently, they have names such as K46radvd, K50dnsmasq, etc: the leading letter+number combination provides sequence control.
- Each of these /etc/rc.d/rc#.d/* scripts control the starting and stopping of a service *(system daemon)*
- Scripts beginning with "K" are *kill* scripts; those starting with "S" are *start* scripts
- When the system begins a new runlevel, all the files beginning with "K" are executed, passing the "stop" parameter to the associated daemon
- When the system begins a new runlevel, all the files beginning with "S" are then executed, passing the "start" parameter to the associated daemon
- When one leaves a runlevel, nothing happens – only when a runlevel is started