Appendix B

Restart Library

B.1 Restart Library

This appendix gives the complete specifications and code for the restart library. This library contains functions that restart when they have not finished the requested operation due to a possibly temporary event. Two main types of events are handled: interruption by a signal and incomplete I/O.

Many library functions, including `read` and `write`, return -1 and set `errno` to `EINTR` when interrupted by a signal before any I/O takes place. This is not a real error, but indicates a natural event when signals occur in the presence of blocking I/O. The functions in this library restart when this occurs, making this error transparent to the user.

Some functions such a `write` may return before a full request is satisfied. When a request is made to write `n` bytes, the `write` is considered successful when any number of bytes greater than zero have been written. A `write` may return a positive value less than `n` if a signal is caught before the requested amount has been written or if an I/O buffer is full, such as when writing to a pipe or network connection. Typically it is up to the program to handle this case and write the remaining bytes. The functions in the restart library handle this, simplifying the user code.

The restart library includes two types of functions. A function whose name start with `r_` is a restart version of a traditional library function and has prototype identical to the corresponding traditional function. For example, the `r_read` function takes the same parameters as `read`, but restarts the read if -1 is returned with `errno` set to `EINTR`. The complete list of these functions is shown in the first part of Table B.1. For these functions, the table only describes the differences between the function given and its traditional counterpart. The interpretation of the parameters does not change. Only those functions that are needed by the code in the book are included in the restart library. Other functions can easily be added, if needed.

Prototype	Description
`int r_close(int fildes)`	Similar to `close`.
`int r_dup2(int fildes, int fildes2)`	Similar to `dup2`.
`int r_open2(const char *path,) int oflag)`	Similar to `open` when called with 2 parameters.
`int r_open3(const char *path,) int oflag, mode_t mode)`	Similar to `open` when called with 3 parameters.
`int r_read(int fd, void *buf, size_t size)`	Similar to `read`.
`pid_t r_wait(int *stat_loc)`	Similar to `wait`.
`pid_t r_waitpid(pid_t pid, int *stat_loc, int options)`	Similar to `waitpid`.
`int r_write(int fd, void *buf, size_t size)`	Similar to `write` but restarts if fewer than `size` bytes are written. The only possible return values are `size` and -1.
`struct timeval add_to_current_time( double seconds)`	returns a `struct timeval` corresponding to the current time plus `seconds` seconds
`int copyfile(int fromfd, int tofd)`	Copy bytes from one open file descriptor to another until an end of file or error occurs.
`int readblock(int fd, void *buf, size_t size)`	Reads exactly `size` bytes into the buffer or return an error.
`int readline(int fd, char *buf, int nbytes)`	Reads a lint into the buffer `buf` which has size `nbytes`.
`int readwrite(int fromfd, int tofd)`	Copy at most `PIPE_BUF` bytes from one open file descriptor to another.
`int readwriteblock(int fromfd, int tofd, char *buf, int size)`	Copy exactly `bytes` from one open file descriptor to another using the given buffer.
`int waitfdtimed(int fd, struct timeval end)`	Wait for data to be available on the given file descriptor or until time `end`.

Table B.1: The functions in the restart library. The first part of the table shows the functions that correspond to traditional functions. All functions in the restart library restart when interrupted by a signal. None of them return -1 with `errno` set to `EINTR`.

The restart library includes additional functions that do not correspond to any traditional library functions. These include `readline` which handles restarting when a signal occurs, and `readblock` which restarts when the requested number by bytes has not yet been read. These functions are shown in the second part of Table B.1. The the table shows the prototypes and short descriptions of these functions.

The following is a more complete description of each of the functions in the restart library.

`struct timeval add_to_current_time(double seconds);`
: Return a `struct timeval` that corresponds to the current time plus `seconds` seconds. The `gettimeofday` function is used to get the current time. The `seconds` parameter is converted to integer values representing seconds and microseconds and added to the current time.

`int copyfile(int fromfd, int tofd);`
: Copy bytes from open file descriptor `fromfd` to open file descriptor `tofd` until either an end of file or an error occurs. Return the number of bytes copied. If some bytes are successfully copied, then no error is returned, even if an error occurs on a write that follows a successful read. If -1 is returned, `errno` is set.

`int r_close(int fildes);`
: This is similar to `close`. The function deallocates the given file descriptor. It returns 0 on success and -1 with `errno` set on error. This function just calls `close` in a loop, restarting if `close` returns -1 with `errno` set to `EINTR`.

`int r_dup2(int fildes, int fildes2);`
: This is similar to `dup2`. The function closes `fildes2` if it was open and causes `fildes2` to refer to the same file as `fildes`. It returns `fildes2` on success and -1 with `errno` set on error. The function just calls `dup2` in a loop, restarting if `dup2` returns -1 with `errno` set to `EINTR`.

`int r_open2(const char *path, int oflag);`
: This is similar to `open` when called with two parameters. It establishes a connection between a file a and a file descriptor. The `oflag` should not have the `O_CREAT` bit set. It returns an open file descriptor on success or -1 with `errno` set on error. The function just calls `open` in a loop, restarting if `open` returns -1 with `errno` set to `EINTR`.

`int r_open3(const char *path, int oflag, mode_t mode);`
: This is similar to `open` when called with three parameters. It establishes a connection between a file a and a file descriptor. The `oflag` should have the

O_CREAT bit set. It returns an open file descriptor on success or -1 with errno set on error. The function just calls open in a loop, restarting if open returns -1 with errno set to EINTR.

```
ssize_t r_read(int fd, void *buf, size_t size);
```
This is similar to read. It reads at most size bytes from the open file descriptor fd into buf. It returns the number of bytes read on success or -1 with errno set on error. The function just calls read in a loop, restarting if read returns -1 with errno set to EINTR.

```
pid_t r_wait(int *stat_loc);
```
This is similar to wait. It suspends execution of the calling thread until status information for one of its terminated children is available. It returns the process ID of a terminated child process on success or -1 with errno set on error. The function just calls wait in a loop, restarting if wait returns -1 with errno set to EINTR.

```
pid_t r_waitpid(pid_t pid, int *stat_loc, int options);
```
This is similar to waitpid. It suspends execution of the calling thread until status information is available for an indicated child process. It returns the process ID of a child process on success or -1 with errno set on error. The function just calls waitpid in a loop, restarting if waitpid returns -1 with errno set to EINTR.

```
ssize_t r_write(int fd, void *buf, size_t size);
```
This is similar to write, but it attempts to write exactly size bytes from buf to the open file descriptor fd. It calls write in a loop, restarting if write returns -1 with errno set to EINTR. If the write does not write all of the requested bytes, write is called again until all of the bytes have been written or an error occurs. The only possible return values are size and -1. If -1 is returned, errno is set.

```
ssize_t readblock(int fd, void *buf, size_t size);
```
Attempt to read exactly size bytes from the open file descriptor fd into the buf. Return 0 if an end of file is reached on fd before any bytes are read. If successful, return size. If an error occurs, return -1 if errno set. If an end of file occurs after some bytes have been read, return -1 with errno set to EINVAL.

```
int readline(int fd, void *buf, size_t size);
```
Attempt to read a line from the open file descriptor fd into buf, a buffer of size size. Return 0 if an end of file is reached on fd before any bytes are read. If

successful, `buf` will contain a string ending with a newline, the length of the string will be the number of bytes read and this value will be returned. If an error occurs, -1 is returned and `errno` is set. Two errors are possible other than an error reading from `fd`, causes `errno` to be set to `EINVAL`. It is an error if some bytes are read and no newline is found before and end of file is reached. An error also occurs if no newline is found in the first `size-1` bytes read.

```
ssize_t readtimed(int fd, void *buf, size_t nbyte,
                  double seconds);
```

Attempt to read at most `nbyte` bytes from the open file descriptor `fd` into the buffer `buf`. This has the same behavior as `r_read` unless no bytes are available in a number of seconds given by `seconds`. If no bytes are available within the timeout period, -1 is returned and `errno` is set to `ETIME`. The timeout period is restarted if this is interrupted by a signal.

```
int readwrite(int fromfd, int tofd);
```

Read at most `PIPE_BUF` bytes from open file descriptor `fromfd` and write the number of bytes read to the open file descriptor `tofd`. Return the number of bytes copied. The value 0 is returned if an end of file is reached on `fromfd` before any bytes are read. If an error occurs, -1 is returned with `errno` set.

```
int readwriteblock(int fromfd, int tofd, char *buf, int size);
```

Read exactly `size` bytes from the open file descriptor `fromfd` and write them to the open file descriptor `tofd`. Use the buffer, `buf`, of size at least `size` to hold the data read. Return 0 if an end of file is reached on `fromfd` before any bytes are read. If successful, return `size`. Otherwise, return -1 with `errno` set.

```
int waitfdtimed(int fd, struct timeval end);
```

Wait until data is available to be read from file descriptor `fd` or until the current time is later than the time in `end`. Return 0 if a read on `fd` will not block. Return -1 with `errno` set to `ETIME` if a `read` on `fd` will still block when time `end` occurs. Return -1 with `errno` set to `EINVAL` if `fd` is negative or greater than or equal to `FD_SETSIZE`.

Program B.1 is the header file containing the prototype for these functions.