Function & Library Reference

```
void* malloc( int num_bytes );
void* calloc( size_t num, size_t size );
void *realloc( void *ptr, size_t new_size );
void free( void* p );
int atoi( const char *s ); /* Convert string to int */
int printf( const char * fmt, ... ); /* %d int, %lu unsigned long, %f double,  %s string, \n newline, ...=args */
void* memcpy( void *destination, const void *source, size_t num_bytes ); /* Returns destination */
void *memset( void *mem, int value, size_t num_bytes ); /* Set memory to 0 */
size_t strlen( const char * string ); /* returns length of null-terminated string, not counting the terminator */
int strncmp ( const char * str1, const char * str2, size_t max_size ); /* return 0 if strings equal */
```

Files

```
int open( const char *filename, int flags );  /* Returns a file descriptor if successful, -1 on error */
ssize_t read( int file_descriptor, void *buffer, size_t count ); /* Returns number of bytes read */
ssize_t write( int file_descriptor, const void *buffer, size_t count ); /* Returns number of bytes written */
int rename( const char *old_filename, const char *new_filename ); /* Returns 0 on success */
int close( int file_descriptor );
FILE* fopen( const char *filename, const char *mode );
int fclose( FILE* f );
size_t fread( void* buffer, size_t size, size_t numbytes, FILE* f );
size_t fwrite( const void *ptr, size_t size, size_t numbytes, FILE* f);
int fprintf( FILE* f, const char* format, ... ); /* Arguments go in the ...  */
int fscanf( FILE* f, const char* format, ... ); /* Arguments go in the ...   */
int fileno( FILE* f ); /* Convert FILE* to file descriptor */
int fseek ( FILE * f, long int offset, int origin ); /*SEEK_SET from file start. SEEK_CUR from current loc.*/
int remove( const char* filename );
int flock( int file_descriptor, int type ); /* LOCK_EX, LOCK_SH, or LOCK_UN for type */
int fcntl( int file_descriptor, int command, ... /* struct flock * flockptr */ );
int lockf( int file_descriptor, int command, off_t length );

struct flock {
  short l_type; /* F_RDLCK, F_WRLCK, or F_UNLCK */
  short l_whence; /* SEEK_SET, SEEK_CUR, or SEEK_END */
  off_t l_start; /* offset in bytes, relative to l_whence */
  off_t l_len; /* length, in bytes; 0 means lock to EOF */
  pid_t l_pid; /* returned with F_GETLK */
};
```

When opening a file with `fopen()`, the options are:

Mode	Meaning
r	Open the file read-only.
w	Open the file for writing (create if needed)
a	Open the file for appending (create if needed)
r+	Open the file for reading, from the start
w+	Open the file for writing (overwrite)
a+	Open the file for reading and writing, append if exists

For `open()` the following flags may be used for the `flags` parameter, and can be combined with | (bitwise OR):

Flag	Meaning
O_RDONLY	Open the file read-only
O_WRONLY	Open the file write-only
O_RDWR	Open the file for both reading and writing
O_APPEND	Append information to the end of the file
O_TRUNC	Initially clear all data from the file
O_CREAT	Create the file
O_EXCL	If used with O_CREAT, the caller MUST create the file; if the file exists it will fail

Process Management

```
pid_t fork( );
pid_t wait( int* status );
pid_t waitpid( pid_t pid, int *status, int options ); /* 0 for options fine */
void exit( int status );
```

Signals

```
int kill( pid_t pid, int signal ); /* returns 0 returned if signal sent, -1 if an error */
int raise( int signal ); /* Send signal to the current process */
void ( *signal(int signum, void (*handler)(int)) ) (int); /* handle signal */
int pause( ) /* Suspend this program until a signal arrives */

int sigemptyset( sigset_t* set ); /* Initialize an empty sigset_t */
int sigaddset( sigset_t* set, int signal ); /* Add specified signal to set */
int sigfillset( sigset_t* set ); /* Add ALL signals to set */
int sigdelset( sigset_t* set, int signal ); /* Remove specified signal from set */
int sigismember( sigset_t* set, int signal ); /* Returns 1 if true, 0 if false */
int sigprocmask( int how, const sigset_t * set, sigset_t * old_set );
```

Message Queues

```
key_t ftok( char *pathname, int proj );
int msgget( key_t key, int flag );
int msgsnd( int msqid, const void *ptr, size_t nbytes, int flag );
ssize_t msgrcv( int msqid, void *ptr, size_t nbytes, long type, int flag );
int msgctl( int msqid, int command, struct msqid_ds * buf ); /* IPC_RMID for command, NULL for buf */

/* IPC Structure can be any struct you like, as long as the first field is long */
struct ipc_msg {
  long mtype; /* Message type must be > 0 */
  char something[1]; /* Can be replaced with any type or structure */
};
```

Pipes and Shared Memory

```
int pipe( int file_descriptors[] ); /* Array should be capacity 2 */
int shmget( key_t key, size_t size, int shmflg ); /* Can use IPC_PRIVATE for key, shmflag = UNIX permissions, can
      combine with IPC_CREAT and/or IPC_EXCL */
void* shmat( int shmid, const void* shmaddr, int shmflg ); /* NULL for shmaddr and 0 for shmflg for defaults */
int shmdt( const void* shmaddr );
int shmctl( int shmid, int cmd, struct shmid_ds *buf ); /* Use cmd = IPC_RMID and buf = NULL to delete */
void* mmap( void* address, size_t length, int protection, int flag, int fd, off_t offset ); /* address = NULL */
int mprotect( void* address, size_t length, int prot ); /* PROT_NONE, PROT_READ, PROT_WRITE, PROT_EXECUTE */
int msync( void* address, size_t length, int flags ); /* Use MS_SYNC for flags */
int munmap( void* address, size_t length );
```

Network

```
int socket( int domain, int type, int protocol );
uint32_t htonl( uint32_t hostint32 ); /* Translate 4 byte int to network format */
uint16_t htons( uint16_t hostint16 ); /* Translate 2 byte int to network format */
uint32_t ntohl( uint32_t netint32 ); /* Translate 4 byte int to host format */
uint16_t ntohs( uint16_t netint16 ); /* Translate 2 byte int to host format */
int getaddrinfo( const char *node, const char *service, const struct addrinfo *hints,
    struct addrinfo **res ); /* node = URL or IP, service = port */
int connect( int sockfd, struct sockaddr *addr, socklen_t len );
int bind( int sockfd, const struct sockaddr *addr, socklen_t addrlen );
int listen( int sockfd, int backlog );
int accept( int sockfd, struct sockaddr *addr, socklen_t *len );
int send( int sockfd, const void* msg, int length, int flags );
int recv( int sockfd, void * buffer, int length, int flags );
int sendto( int sfd, const void* msg, int len, unsigned int flags, const struct sockaddr* to, socklen_t token );
int recvfrom( int sfd, void* buffer, int len, unsigned int flags, struct sockaddr* from, int* fromlength );

CURLcode curl_global_init( long flags ); /* use CURL_GLOBAL_DEFAULT as flags */
CURL* curl_easy_init( );
void curl_easy_cleanup( CURL* handle );
CURLcode curl_easy_setopt( CURL *handle, CURLoption option, parameter ); /* See table below */
CURLcode curl_easy_perform( CURL * easy_handle );
CURLcode curl_easy_getinfo( CURL *curl, CURLINFO info, ... );
void curl_global_cleanup( );
CURLM *curl_multi_init( );
CURLMcode curl_multi_add_handle( CURLM *multi_handle, CURL *easy_handle );
CURLMcode curl_multi_remove_handle( CURLM *multi_handle, CURL *easy_handle );
CURLMcode curl_multi_perform( CURLM *multi_handle, int *running_handles );
CURLMcode curl_multi_wait(CURLM *multi_handle, struct curl_waitfd extra_fds[],
  unsigned int extra_nfds, int timeout_ms, int *numfds );
CURLMsg *curl_multi_info_read( CURLM *multi_handle, int *msgs_in_queue );
CURLMcode curl_multi_fdset( CURLM *mh, fd_set *rd_fd_set, fd_set *wr_fd_set, fd_set *ex_fd_set, int *max_fd );
CURLMcode curl_multi_timeout( CURLM *multi_handle, long *timeout );
size_t write_callback( char *ptr, size_t size, size_t nmemb, void *userdata );
size_t read_callback( char *buffer, size_t size, size_t nitems, void *inputdata );
```

Options for `curl_easy_setopt`:

CURLoption	Meaning	Parameter
CURLOPT_URL	The URL to connect to	Character array
CURLOPT_WRITEFUNCTION	Set the write callback function	Function Pointer
CURLOPT_WRITEDATA	Set the write callback data	void*
CURLOPT_READFUNCTION	Set the read callback function	Function Pointer
CURLOPT_READDATA	Set the read callback data	void*
CURLOPT_PUT	Issue HTTP PUT request	1L
CURLOPT_POST	Issue HTTP POST request	1L
CURLOPT_HTTPGET	Issue HTTP GET request (this is the default, though)	1L

Threads and Concurrency

```
pthread_create( pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)( void * ), void *arg );
pthread_join( pthread_t thread, void **returnValue );
pthread_detach( pthread_t thread );
pthread_cancel( pthread_t thread );
pthread_testcancel( ); /* If the thread is cancelled, this function does not return (thread terminated) */
pthread_setcanceltype( int type, int *oldtype );
pthread_cleanup_push( void (*routine)(void*), void *argument ); /* Register cleanup handler, with argument */
pthread_cleanup_pop( int execute ); /* Run if execute is non-zero */
pthread_exit( void *value );

pthread_mutex_init( pthread_mutex_t *mutex, pthread_mutexattr_t *attributes );
pthread_mutex_lock( pthread_mutex_t *mutex );
pthread_mutex_trylock( pthread_mutex_t *mutex ); /* Returns 0 on success */
pthread_mutex_unlock( pthread_mutex_t *mutex );
pthread_mutex_destroy( pthread_mutex_t *mutex );
pthread_rwlock_init( pthread_rwlock_t * rwlock, pthread_rwlockattr_t * attr );
pthread_rwlock_rdlock( pthread_rwlock_t * rwlock );
pthread_rwlock_tryrdlock( pthread_rwlock_t * rwlock );
pthread_rwlock_wrlock( pthread_rwlock_t * rwlock );
pthread_rwlock_trywrlock( pthread_rwlock_t * rwlock );
pthread_rwlock_unlock( pthread_rwlock_t * rwlock );
pthread_rwlock_destroy( pthread_rwlock_t * rwlock );

sem_init( sem_t* semaphore, int shared, int initial_value); /* 0 for shared OK */
sem_destroy( sem_t* semaphore );
sem_wait( sem_t* semaphore );
sem_trywait( semt_t* semaphore );
sem_post( sem_t* semaphore );

pthread_cond_init( pthread_cond_t *cv, pthread_condattr_t *attributes );
pthread_cond_wait( pthread_cond_t *cv, pthread_mutex_t *mutex );
pthread_cond_signal( pthread_cond_t *cv );
pthread_cond_broadcast( pthread_cond_t *cv );
pthread_cond_destroy( pthread_cond_t *cv );
```