1 Posix Threads

```
int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_getdetachstate(const pthread_attr_t *attr,int
    *detachstate);
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int
    detachstate);
int pthread_cancel(pthread_t thread);
int pthread_detach(pthread_t th);
int pthread_cond_broadcast(pthread_cond_t* cond);
int pthread_cond_destroy(pthread_cond_t* cond);
int pthread_cond_init(pthread_cond_t* cond, const
    pthread_condattr_t* attr);
int pthread_cond_signal(pthread_cond_t* cond);
int pthread_cond_wait(pthread_mutex_t* mutex, pthread_cond_t*
    cond);
int pthread_create(pthread_t * thread, pthread_attr_t * attr,
    void *(*start_routine)(void *), void * arg);
int pthread_equal (pthread_t one_thread, pthread_t other_thread
    );
void pthread_exit(void * retval);
int pthread_join(pthread_t which_thread, void **
    thread_return_value);
int pthread_key_create(pthread_key_t *key, void (*routine)(void
     *));
int pthread_key_delete(pthread_key_t key);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *mutex, const
    pthread_mutexattr_t *attr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_setspecific(pthread_key_t key, const void *
    value_ptr);

int pthread_cond_init(&cond, NULL);
int pthread_setcancelstate (int state, int *old_state);
int pthread_setcanceltype (int cancel_type, int *old_state);
pthread_t pthread_self ();

void* pthread_getspecific(pthread_key_t key);
void pthread_testcancel();

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

2 Process Management

```
int daemon(int nochdir, int noclose);
int execl(const char * program_path, const char * arg, ...);
int execlp(const char * program_path, const char * arg, ...);
int execv(const char * program_path, char * const arg_list[]);
int execve(const char *filename, char *const argv[], char *
```

```
    const envp[]);
int execvp(const char * program_path, char * const arg_list[]);
pid_t fork();
pid_t getpid();
pid_t getppid();
pid_t wait(int * status);
pid_t waitpid(pid_t pid, int * status, int options);
FILE *popen(const char *command, const char *type);
int pclose(FILE *stream);
unsigned int sleep(unsigned int seconds);
void exit(int status);
```

3 Files and I/O

```
FILE* fdopen(int file_descriptor, const char *mode);
FILE * fopen(const char *restrict filename, const char *
    restrict mode);
int feof(FILE * stream);
int fflush(FILE * stream);
int fileno(FILE* stream);
int fcntl(int fd, int cmd, struct flock *lock);
int dup2(int oldfd, int newfd);
int dup(int oldfd);
int close(int file_descriptor);
int creat(const char *pathname, mode_t mode);
int fseek(FILE *stream, long offset, int whence);
int fsync(int fd);
size_t fwrite(const void * ptr, size_t size, size_t nmemb,FILE
    * stream);
int open(const char *pathname, int flags, mode_t mode);
int chmod(const char * path, mode_t mode);
int link(const char *path1, const char *path2);
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
int unlink(const char *pathname);
int pipe(int filedes[2]);
int select(int n, fd_set * readfds, fd_set * writefds,fd_set *
    exceptfds, struct timeval * timeout);

int fprintf(FILE * stream, const char * format, ...);
int scanf(const char * format, ...);
size_t fread(void * ptr, size_t size, size_t nmemb, FILE *
    stream);
int fscanf(FILE * stream, const char * format, ...);
int getopt_long(int argc, char * const argv[], const char *
    short_option_string,const struct option * long_option_struct
    , int *longindex);
```

4 Semaphores

```
struct shmid_ds{
          struct ipc_perm shm_perm;
          int shm_segsz;
```

```
        u_short shm_lpid;
        u_short shm_cpid;
        u_short shm_nattch;
        time_t shm_atime;
        time_t shm_dtime;
        time_t shm_ctime;
  };

union semun {
        int val;
        struct semid_ds *buf;
        unsigned short int *array;
        struct seminfo *__buf;
};

struct sembuf {
        unsigned short sem_num;
        short          sem_op;
        short          sem_flg;
};

int semctl(int semid, int semnum, int cmd, union semun arg);
        command: IPC_RMID, SETALL
int semget(key_t key, int num_sems, int sem_flags);
        key : IPC_PRIVATE, ...
        sem_flags : 0_CREAT | S_IRWXU ,...
int sem_destroy(sem_t* semaphore);
int sem_getvalue(sem_t * sem, int * sval);
int sem_init (sem_t* semaphore, 0, int initial_value);
int semop(int semid, struct sembuf *sops, unsigned nsops);
int sem_post(sem_t* semaphore);
int sem_trywait(sem_t* semaphore);
int sem_wait(sem_t* semaphore);
```

5 Signals

```
int kill(pid_t pid, int sig);
int sigaction(int sig, const struct sigaction* action,struct
    sigaction* old_act);
sighandler_t signal(int signum, sighandler_t handler);

struct sigaction {
        void (*sa_handler)(int);
        void (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t sa_mask;
        int sa_flags;
        void (*sa_restorer)(void);
}
```

6 Sockets

```
const int PF_LOCAL, PF_INET;
const int SOCK_STREAM, SOCK_DGRAM;
```

```
struct sockaddr_un {
        u_char  sun_len;
    u_char  sun_family;
    char    sun_path[104];
};

struct sockaddr_in {
    short           sin_family;
    unsigned short  sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
};
int accept(int socket_fd, struct sockaddr* address, socklen_t *
     addrlen);
int bind(int socket_fd, struct sockaddr* my_address,socklen_t
    addrlen);
int connect(int socket_fd, const struct sockaddr *serv_addr,
    socklen_t addrlen);
int listen(int socket_fd, int max_connections);
int socket(int domain, int type, int protocol);
        domain: PF_LOCAL, PF_INET, ...
        type : SOCK_STREAM, SOCK_DGRAM, ...
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
struct hostent *gethostbyname(const char *name);
struct hostent* gethostbyaddr(const char * address, int len,
    int type);
```

7 Memory and string management

```
char * strncat(char *restrict s1, const char *restrict s2,
    size_t n);
char * strsep(char **stringp, const char *delim);
char * strstr(const char *s1, const char *s2);
char * strtok(char *restrict s1, const char *restrict s2);
char * strerror(int errnum);
size_t strspn(const char *s1, const char *s2);

void free(void * ptr);
int munmap(void *start, size_t length);
void *malloc(size_t size);
void *mmap (int start_position, size_t file_length, int access,
     int flags, int file_descriptor, int offset);
void * realloc(void * ptr, size_t size);
```