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

struct sembuf operations[...];
 operations[i].sem_num = ...
 operations[i].sem_op  = ...
 operations[i].sem_flg = ...


pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

struct sockaddr_un      name;
struct sockaddr_in      name;
struct hostent *        hostinfo;

typedef void (*sighandler_t)(int);

int                accept(int s, struct sockaddr * address, socklen_t * addrlen);
void               assert(scalar_expression);
int                bind(int sockfd, struct sockaddr * my_address,
                        socklen_t addrlen);
int                chmod(const char * path, mode_t mode);
int                close(int fd);
int                connect(int sockfd, const struct sockaddr * serv_addr,
                           socklen_t addrlen);
int                creat(const char * path, mode_t mode);
extern int         errno;
void               exit(int status);
struct hostent *   gethostbyaddr(const char * address, int len, int type);
struct hostent *   gethostbyname(const char * name);
int                getopt_long(int argc, char * const argv[],
                               const char * short_option_string,
                               const struct option * long_option_struct,
                               int *longindex);
                   next_option = getopt_long(argc, argv, short_options,
                                             long_options, NULL);
                   extern char *    optarg;
                   extern int       optind, opterr, optopt;
uint32_t           htonl(uint32_t hostlong);
uint16_t           htons(uint16_t hostshort);
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                execvp(const char * program_path, char * const arg_list[]);
int                fchmod(int fd, mode_t mode);
int                feof(FILE * stream);
int                fflush(FILE * stream);
int                fileno(FILE * stream);
pid_t              fork(void);
int                fprintf(FILE * stream, const char * format, ...);
size_t             fread(void * ptr, size_t size, size_t nmemb, FILE * stream);
void               free(void * ptr);
int                fscanf(FILE * stream, const char * format, ...);
int                fsync(int fd);
size_t             fwrite(const void * ptr, size_t size, size_t nmemb,
                          FILE * stream);
pid_t              getpid(void);
pid_t              getppid(void);
int                kill(pid_t pid, int sig);
int                link(const char * oldpath, const char * newpath);
int                listen(int s, int backlog);
void *             malloc(size_t size);
uint32_t           ntohl(uint32_t netlong);
```



```
uint16_t         ntohs(uint16_t netshort);
int              open(const char *pathname, int flags);
int              open(const char *pathname, int flags, mode_t mode);
int              pause(void);
int              pipe(int filedes[2]);
                 int pipe_fds[2], read_fd, write_fd;
                 pipe(pipe_fds);
                 read_fd  = pipe_fds[0];
                 write_fd = pipe_fds[1];
int              printf(const char * format, ...);
int              pthread_create(pthread_t * thread, pthread_attr_t * attr,
                                void *(*start_routine)(void *), void * arg);
int              pthread_cancel(pthread_t thread);
int              pthread_detach(pthread_t th);
void             pthread_exit(void * retval);
int              pthread_join(pthread_t th, void ** thread_return);
int              pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutex,
                                    attr_t * mutexattr);
int              pthread_mutex_lock(pthread_mutex_t * mutex);
int              pthread_mutex_trylock(pthread_mutex_t * mutex);
int              pthread_mutex_unlock(pthread_mutex_t * mutex);
ssize_t          read(int fd, void * buf, size_t count);
void *           realloc(void * ptr, size_t size);
int              scanf(const char * format, ...);
int              sem_destroy(sem_t * sem);
int              sem_getvalue(sem_t * sem, int * sval);
int              sem_init(sem_t * sem, int pshared, unsigned int value);
int              sem_trywait(sem_t * sem);
int              sem_post(sem_t * sem);
int              sem_wait(sem_t * sem);
int              semctl(int semid, int semnum, int command, ...);
                 command: IPC_RMID, SETALL
int              semget(key_t key, int num_sems, int sem_flags);
                 key       : IPC_PRIVATE, ...
                 sem_flags : O_CREAT | S_IRWXU ,...
int              semop(int semid, struct sembuf *sops, unsigned nsops);
int              semtimedop(int  semid, struct sembuf * sops,
                            unsigned nsops, struct timespec * timeout);
void *           shmat(int shmid, const void * shmem_address, int shmem_flags);
int              shmctl(int shmid, int command, struct shmid_ds * buffer);
                 command: IPC_STAT, IPC_RMID
int              shmdt(const void *shmem_address);
int              shmget(key_t key, size_t size, int shmem_flags);
                 key       : IPC_PRIVATE, ...
                 sem_flags : PC_CREAT | IPC_EXCL | S_IRUSR | IWUSR, ...

sighandler_t     signal(int signum, sighandler_t handler);
unsigned int     sleep(unsigned int seconds);
int              select(int n, fd_set * readfds, fd_set * writefds,
                        fd_set * exceptfds, struct timeval * timeout);
int              snprintf(char * string, size_t size, const char * format, ...);
int              socket(int domain, int type, int protocol);
                 domain: PF_LOCAL, PF_INET, ...
                 type  : SOCK_STREAM, SOCK_DGRAM, ...
int              sprintf(char * string, const char * format, ...);
int              sscanf(const char * string, const char * format, ...);
char *           strerror(int errnum);
void             sync(void);
int              system(const char * command_string);
int              unlink(const char * pathname);
pid_t            wait(int * status);
pid_t            waitpid(pid_t pid, int * status, int options);
ssize_t          write(int fd, const void *buf, size_t count);
```