An incomplete, non-exhaustive list of useful library and system calls covered in CS241. For brevity, const and restrict keywords not shown. In written answers you may safely shorten pthread calls and macros, provided it is unambiguous to the grader. e.g. You may write `p_m_t lock = P_M_I` instead of `pthread_mutex_t lock = PTHREAD_MUTEX_INITALIZER`

```
void *memcpy(void *dest, void *src, size_t n)
void *memset(void *b, int c, size_t len)
char *strcpy(char *dest, char *src)
char *strcat(char *dest,char*src)
char *strncpy(char *dest,char*src, size_t n)
char *strncat(char *dest,char*src, size_t n)
int strcmp(char *s1, char *s2)
int strncmp(char *s1, char *s2, size_t n)
void *calloc(size_t nmemb, size_t size)
void *malloc(size_t size)
void free(void *ptr)
void *realloc(void *ptr, size_t size)
pid_t fork()
char * getenv(char *name)
int execve(char *path, char *argv[], char *envp[])
int execl(char *path, char *arg0, ... ) /* arg0 will be process name. Last arg must be (char*)0 */
pid_t getpid()
pid_t getppid()
int kill(pid_t pid, int sig) /* SIGINT, SIGKILL,SIGALRM ... */
pid_t wait(int *stat_loc)
pid_t waitpid(pid_t pid, int *stat_loc, int options) WIFEXITED ,WIFSIGNALED,WEXITSTATUS. options=WNOHANG
WIFEXITED(status) returns True if the process terminated normally by a call to _exit(2) or exit(3).
WIFSIGNALED(status) returns True if the process terminated due to receipt of a signal.
WEXITSTATUS(status) If WIFEXITED(status) is true, evaluates to the low-order 8 bits of the process's exit value.
int pthread_join(pthread_t thread, void **value_ptr)
int pthread_create(pthread_t * thread,  pthread_attr_t * attr, void *(*start_routine)(void *), void * arg)
int pthread_kill(pthread_t thread, int sig)
void pthread_exit(void *value_ptr)
pthread_mutex_t /* PTHREAD_MUTEX_INITIALIZER */
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_mutex_destroy(pthread_mutex_t *mutex)
pthread_cond_t /* PTHREAD_COND_INITIALIZER */
int pthread_cond_init(pthread_cond_t * cond, pthread_condattr_t * attr)
int pthread_cond_wait(pthread_cond_t * cond, pthread_mutex_t * mutex)
int pthread_cond_signal(pthread_cond_t *cond)
int pthread_cond_broadcast(pthread_cond_t *cond)
int pthread_cond_destroy(pthread_cond_t *cond)
int sem_init(sem_t *sem, int pshared, unsigned int value)
int sem_wait(sem_t *sem)
int sem_trywait(sem_t *sem)
int sem_post(sem_t *sem)
int sem_destroy(sem_t *sem)
int stat(char *path, struct stat *buf)  /* S_ISREG(mode), S_ISDIR(mode), S_ISLNK(mode) */
int lstat(char *path, struct stat *buf)
struct stat {
   dev_t     st_dev;     /* ID of device containing file */
   ino_t     st_ino;     /* inode number */
   mode_t    st_mode;    /* protection */
   nlink_t   st_nlink;   /* number of hard links */
   uid_t     st_uid;     /* user ID of owner */
   gid_t     st_gid;     /* group ID of owner */
   off_t     st_size;    /* total size, in bytes */
   ... };
int open(char *pathname, int flags) /*flags = O_RDONLY,O_WRONLY,O_RDWR,O_CREAT */
int open(char *pathname, int flags, mode_t mode) /*mode=octal or S_IWUSR,S_IXGRP,S_IROTH...*/
int pipe(int fds[2]) /* Write to fds[1], read from fds[0]*/
```

```
ssize_t read(int fildes, void *buf, size_t nbyte)
ssize_t write(int fildes,  void *buf, size_t nbyte)
int close(int fd)
int dup2(int oldfd, int newfd) /* An existing fd with value newfd will be closed first */
int accept(int socket, struct sockaddr * address, socklen_t * address_len) /* address,address_len can be null */
int listen(int socket, int backlog)
int socket(int domain, int type, int protocol)
int connect(int socket, const struct sockaddr *address, socklen_t address_len)
int bind(int socket,  struct sockaddr *address, socklen_t address_len)
ssize_t recv(int socket, void *buffer, size_t length, int flags)
ssize_t recvfrom(int socket, void *buffer, size_t length, int flags, struct sockaddr * address, socklen_t * address_le
ssize_t send(int socket, const void *buffer, size_t length, int flags)
ssize_t sendto(int socket, const void *buffer, size_t length, int flags, struct sockaddr *dest_addr, socklen_t dest_le
int getaddrinfo( char *hostname,  char *servname, struct addrinfo *hints, struct addrinfo **res)
void freeaddrinfo(struct addrinfo *ai)
struct addrinfo {/* When used as a hint, unused entries should be zero/null */
   int ai_flags;           /* eg. AI_PASSIVE, AI_NUMERICSERV,AI_NUMERICHOST  */
   int ai_family;          /* eg. AF_INET, AF_INET6,PF_UNSPEC... */
   int ai_socktype;        /* eg. SOCK_DGRAM, SOCK_STREAM, SOCK_RAW */
   int ai_protocol;        /* eg. IPPROTO_UDP or IPPROTO_TCP */
   socklen_t ai_addrlen;   /* length of socket-address */
   struct sockaddr *ai_addr; /* socket-address for socket */
   char *ai_canonname;     /* canonical name for service location */
   struct addrinfo *ai_next; /* pointer to next in list */
}
typedef void (*sighandler_t)(int)
sighandler_t signal(int signum, sighandler_t handler)
int sigaction(int signum,    struct sigaction *act, struct sigaction *oldact)
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t   sa_mask;
  int        sa_flags;}
int sigprocmask(int how, sigset_t *set, sigset_t *oldset) /*how=SIG_BLOCK,SIG_UNBLOCKSIG_SETMASK*/
int pthread_sigmask(int how, sigset_t *set, sigset_t *oldset)
int sigemptyset(sigset_t *set)
int sigfillset(sigset_t *set)
int sigaddset(sigset_t *set, int signo)
FILE *fopen(char *path,char *mode) /*mode=a,a+,w,w+,r,...*/
int feof(FILE *stream) /* e.g. stdin,stdout,stderr */
int ferror(FILE *stream) int fflush(FILE *stream) int fclose(FILE*stream)
size_t fread(void * ptr, size_t size, size_t nitems, FILE * stream)
int fseek(FILE *stream, long offset, int whence) /*whence=SEEK_SET, SEEK_CUR, or SEEK_END*/
int lseek(int fd, off_t offset, int whence) /*whence=SEEK_SET, SEEK_CUR, or SEEK_END*/
long ftell(FILE *stream)
int fgetpos(FILE *stream, fpos_t *pos) int fsetpos(FILE *stream, fpos_t *pos) void rewind(FILE *stream)
int fclose(FILE *fp)
ssize_t getline(char **  linep, size_t *  linecapp, FILE *  stream)
char *fgets(char *s, int size, FILE *stream)
DIR *opendir(char *name)
int closedir(DIR *dirp)
struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result)
struct dirent {
  ino_t          d_ino;       /* inode number */
  char           d_name[256]; /* filename */  /* Other entries not shown */
}
void * mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset)
prot= PROT_READ | PROT_WRITE |PROT_EXEC,  flags= MAP_ANON|MAP_PRIVATE|MAP_SHARED
int munmap(void *addr, size_t len)
```