Formulario per il corso di Sistemi Operativi

Console

```
int printf(const char *format, ...);
int scanf(const char *format, ...);
```

Gestione degli Errori

```
void perror(const char *s);
char *strerror(int errnum);
```

Stringhe

```
size_t strlen(const char *s);
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t n);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
int sprintf(char *str, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
int atoi(const char *nptr);
double atof(const char *nptr);
```

File (Libreria Standard)

```
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *fp);
int fgetc(FILE *stream);
char *fgets(char *s, int size, FILE *stream);
int fputc(int c, FILE *stream);
int fputs(const char *s, FILE *stream);
int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int fseek(FILE *stream, long offset, int whence ∈ {SEEK_SET, SEEK_CUR
↪  SEEK_END});
long ftell(FILE *stream);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE * stream );
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE
↪  *stream);
void setbuf(FILE *stream, char *buf);
int fflush(FILE *stream);
int remove(const char *pathname);
```

Cartelle (System Call)

```
DIR * opendir(const char *name);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
struct dirent {
    ino_t          d_ino;
    char           d_name[] };
```

File (System Call)

```
int open(const char *path, int oflag  ∈ {O_APPEND, O_CREAT, O_RDONLY,
↪  O_WRONLY, O_RDWR}, ... );
int close(int fildes);
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, void * buf, size_t nbyte);
off_t lseek(int fildes, off_t offset, int whence ∈ {SEEK_SET, SEEK_CUR
↪  SEEK_END});
int link(const char *path1, const char *path2);
int symlink(const char *path1, const char *path2);
int unlink(const char *pathname);
int mkdir(const char *path, mode_t mode);
int rmdir(const char *path);
int truncate(const char *path, off_t length);
int ftruncate(int fildes, off_t length);
int stat(const char *restrict path, struct stat *restrict buf);
struct stat {
    ino_t     st_ino;
    mode_t    st_mode;
    uid_t     st_uid;
    gid_t     st_gid;
    off_t     st_size;
    ... };
S_ISREG(st_mode); S_ISDIR(st_mode); S_ISLNK(st_mode); S_FIFO(st_mode);
```

Processi

```
pid_t fork(void);
pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execle(const char *path, const char *arg0, ... /*,
(char *)0, char *const envp[]*/);
int execve(const char *path, char *const argv[], char *const envp[]);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execvp(const char *file, char *const argv[]);
int system(const char *command);
void exit(int status);
int atexit(void (*function)(void));
void _exit(int status);
void abort(void);
pid_t getpid(void);
pid_t getppid(void);
```

Segnali

```
int sigaction(int sig, const struct sigaction *restrict act, struct
↪  sigaction *restrict oact);
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
int sigemptyset(sigset_t *sa_mask);
int kill(pid_t pid, int sig);
int raise(int sig);
int pause(void);
unsigned alarm(unsigned seconds);
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    ...
};
```

Inter-process communication e Memoria

```
int pipe(int fildes[2]);
int mkfifo(const char *pathname, mode_t mode);
void *mmap(void *addr, size_t len, int prot, int flags, int fildes,
↪  off_t off);
int munmap(void *addr, size_t len);
int shm_open(const char *name, int oflag, mode_t mode);
int shm_unlink(const char *name);
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
int brk(void *addr);
void *sbrk(intptr_t increment);
```

Thread e Sincronizzazione

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void
↪  *(*start_routine) (void *), void *arg);
void pthread_exit(void *retval);
int pthread_cancel(pthread_t thread);
pthread_t pthread_self(void);
int pthread_join(pthread_t thread, void **retval);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex, const
↪  pthread_mutexattr_t *restrict 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);
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int
↪  value);
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
```