ROB4 - INFORMATIQUE SYSTÈME
Function prototypes

stdlib.h

```
#define RAND_MAX 0x7FFF
int     abs (int i);
long    labs (long i);
int     atoi (const char * str);
long    atol (const char * str);
double  atof (const char * str);
int  rand (void);
void    srand (unsigned int seed);
void    exit (int status);
void*   malloc (size_t size);
void    free (void *ptr);
void*   calloc (size_t nelem, size_t size);
void*   realloc (void *ptr, size_t size);

int     system (const char * command);
The value returned is -1 on error (e.g.  fork(2) failed),
and the return status of the command otherwise.
This latter  return  status  is  in  the  format
specified in wait(2). Thus, the exit code of the
command will be WEXITSTATUS(status).
```

string.h

```
void*  memccpy (void *dest, const void *src, int c, size_t nbytes);
int    memcmp (const void *str1, const void *str2, size_t nbytes);
void*  memcpy (void *dest, const void *src, size_t nbytes);
void*  memmove (void *dest, const void *src, size_t nbytes);
void*  memset (void *str, int c, size_t nbytes);

char*  strcat (char *s1, const char *s2);
The strcat() and strncat() functions return a pointer to the resulting
string dest.
```

```
char*  strchr (const char *str, int c);

int    strcmp (const char *s1, const char *s2);
The strcmp() and strncmp() functions return an integer less than,
equal to, or greater than zero if s1 (or the first n bytes thereof)
is found, respectively, to be less than, to match, or be greater than s2.

char*  strcpy (char *dest, const char *src);
The strcpy() and strncpy() functions return a pointer to the
destination string dest.

size_t strlen (const char *str);
char*  strncpy (char *dest, const char *src, size_t nbytes);
char*  strtok (char *src, const char *strcut);
```

stdio.h

```
int    fclose (FILE *stream);

FILE*  fopen (const char *filename, const char *mode);
FILE*  fdopen (int fildes, const char *mode);
Upon  successful  completion fopen(), fdopen() and freopen()
return a FILE pointer.  Otherwise, NULL is returned and
errno is set to indicate the error.

int    fflush (FILE *stream);

int    fgetc (FILE *stream);
char*  fgets (char *s, int n, FILE *stream);
fgetc(),  getc() and getchar() return the character
read as an unsigned char cast to an int or EOF on end of file or
error.
gets() and fgets() return s on success, and NULL on error or when
end of file occurs while no characters  have  been read.

int    fileno (FILE *stream);
int    fprintf (FILE *stream, const char *format, ...);
int    printf (const char *format, ...);
int    sprintf (char *s, const char *format, ...);

int    fputc (int c, FILE *stream);
int    fputs (const char *s, FILE *stream);
```

```
fputc(), putc() and putchar() return the character written as an
unsigned char cast to an int or EOF on error.
puts() and fputs() return a nonnegative number on success, or EOF on error.

size_t fwrite (const void *ptr, size_t size, size_t nitems, FILE *stream);
size_t fread (void *ptr, size_t size, size_t nitems, FILE *stream);
fread() and fwrite() return the number of items successfully read or
written (i.e., not the number of characters). If an error occurs, or
the end-of-file is reached, the return value is a short item count (or zero).
fread() does not distinguish between end-of-file and error, and callers
must use feof(3) and ferror(3) to determine which occurred.

int    fscanf (FILE *stream, const char *format, ... );
int    scanf (const char *format, ... );
int    sscanf (const char *s, const char *format, ... );
int    fseek (FILE *stream, long int offset, int whence);
long int ftell (FILE *stream);
FILE*  popen (const char *command, const char *mode);
void   perror (const char *s);
void   rewind (FILE *stream);
```

assert.h

```
void  assert (int expression);
```

math.h

```
double  pow (double base, double exponent);
double  sqrt (double x);
```

time.h

```
time_t  time (time_t *t);
```

unistd.h

```
int  chdir(const char *path);
int  chroot(const char *path);
int  chown(const char *path, uid_t owner, gid_t group);
int  rmdir(const char *path);
int  unlink(const char *path);
```

```
int  execv(const char *path, char *const argv[]);
The  exec()  functions only return if an error has have occurred.
The return value is -1, and errno is set to indicate the error.

int  pause(void);
unsigned int sleep(unsigned int seconds);
int  usleep(useconds_t useconds);
int  pipe(int fildes[2]);

int  close(int fildes);
close() returns zero on success. On error, -1 is returned,
and errno is set appropriately.

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset);
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t write(int fildes, const void *buf, size_t nbyte);
ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset);
```

the following also need **sys/types.h**

```
pid_t fork (void);
gid_t getgid (void);
pid_t getpid (void);
pid_t getppid (void);
uid_t getuid (void);
off_t lseek (int fildes, off_t offset, int whence);
int   setgid (gid_t gid);
int   setuid (uid_t uid);
```

sys/wait.h

```
pid_t wait(int *stat_loc);
pid_t waitpid (pid_t pid, int *stat_loc, int options);
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
wait(): on success, returns the process ID of the terminated child;
on error, -1 is returned.
waitpid(): on success, returns the process ID of the child whose state
has changed; if WNOHANG was specified and one or more child(ren) specified
by pid exist, but have not yet changed state, then 0 is  returned.
On error, -1  is returned.
```

```
waitid(): returns 0 on success or if WNOHANG was specified and no child(ren)
specified by id has yet changed state; on error, -1 is returned.
Each of these calls sets errno to an appropriate value in the case of an error.
```

signal.h

```
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
int kill(pid_t pid, int sig);
int sigwait (const sigset_t *set, int *sig);
int sigaction (int signum, const struct sigaction *act, struct sigaction *oldact);
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

sys/stat.h

the following also need **sys/types.h**

```
int  chmod(const char *path, mode_t mode);
int  fchmod(int fildes, mode_t mode);
int  stat(const char *path, struct stat *buf);
int  fstat(int fildes, struct stat *buf);
int  mkfifo(const char *path, mode_t mode);
int  mkdir(const char *path, mode_t mode);
int  mknod(const char *path, mode_t mode, dev_t dev);
```

fcntl.h

the following also need **sys/stat.h**

```
int open (const char *pathname, int flags);
int creat (const char *pathname, mode_t mode);
open() and creat() return the new file descriptor, or -1 if
an error occurred (in which case, errno is set appropriately).
```

sys/types.h

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond;
pid_t pid;
```

pthread.h

```
int  pthread_create (pthread_t *thread, const pthread_attr_t *attr,
    void *(*start_routine)(void*), void *arg);
On  success, pthread_create() returns 0; on error, it returns an error
number, and the contents of *thread are undefined.

void pthread_exit (void *value_ptr);
int  pthread_detach (pthread_t thread);
int  pthread_cancel (pthread_t thread);

int  pthread_join (pthread_t thread, void **value_ptr);
On success, pthread_join() returns 0; on error, it returns an error number.

int  pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
int  pthread_cond_signal(pthread_cond_t *cond);
int  pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int  pthread_mutex_destroy (pthread_mutex_t *mutex);
int  pthread_mutex_lock (pthread_mutex_t *mutex);
int  pthread_mutex_trylock (pthread_mutex_t *mutex);
int  pthread_mutex_unlock (pthread_mutex_t *mutex);
```