getpid :- header file :- unistd.h

Return type :- pid_t

Parameter :- void

It Return proccess Id

Return value :- It always be successful and no return value is reserved to indicate an error

Error :- no error

```
pid_t getpid(void);
```

getppid :- header file :- unistd.h

Return type :- pid_t

Parameter :- void

It return parent proccess Id

Return value :- It always be successful and no return value is reserved to indicate an error

Error :- no error

```
pid_t getppid(void);
```

exit :- header file :- stdlib.h

Return type :- void

Parameter :- status

It ends a process and return a value to its parent

Status is an integer b/w 0-255. This number is returned to parent via wait() as the exit status of the process

```
void exit(int status);
```

_exit :- header file :- unistd.h

Return type :- void

Parameter :- status

exit clean up the standard I/O streams (tempfile()) before calling _exit();
_exit insted of exit() will bypass this cleanup
It ends a process and return a value to its parent

Status is an integer b/w 0-255. This number is returned to parent via wait() as the _exit status of the process

```
void _exit(int status);
```

fork :- header file :- unistd.h

Return type :- pid_t

Parameter :- void

Return value :- Successful return 0 to the child process and shall return the process ID of the child process to the parent process

Error :- -1 if Error :- no resources, insufficient memory and CHILD_MAX is exceeded.

```
pid_t fork(void);
```

wait :- header file :- sys/wait.h

Return type :- pid_t

Parameter :- *stat_loc (Status from child)

It return the exit status multiplied by 256 right 8 bit (divided by 256) to obtain the correct value

if exit	status	is	1	:-	0000001 0000000
"	"	"	"	2	:- 0000010 0000000
"	"	"	"	-1	:- 255
"	"	"	"	-2	:- 254

Error :- -1 (EINTR)

1. The process has no child to wait for
2. stat_loc points to an invalid addd

Info return by wait()

1. Ended by exit() :- 2nd lowest byte is set to the argument by exit()
   Lowest byte is set to zeros
2. Ended by signal :- Lowest byte is set to signal no that ended the process 2nd lowest byte is set to zeros

```
pid_t wait(int *stat_loc);
```

waitpid :-

header file :- sys/wait.h

Return type :- pid_t

parameter :- pid, *stat_loc, options

If pid is 0, status is required for any child process whose group ID is equal to that of the calling process

If pid is >0 It specify the process ID of a single child process whose status is requested

If pid is -1, status is required for any child process. then and option val is 0 waitpid is equivalent to wait()

```
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

pause :- header file :- unistd.h

Return type :- int

Parameter :- void

It suspends the execution of the calling process until it receives a signal

If the action is to terminate the process, pause() shall not return

If the action is to execute a signal-catching function (SIGCONT) <signal.h>, pause() shall return after the signal-catching function returns.

Error :- -1 for no successful completion

It fail if a signal is caught by the calling process and control is returned from the signal-catching function

```
int pause(void);
```

kill :- header file :- signal.h

Return type :- int

Parameter :- pid, signal

It send signal to a process or a group of processes specified by pid
If signal is 0 (the null signal), error checking is performed but no signal is actually send
The null signal can be used to check the validity of pid

if pid > 0 signal send to the process whose process ID is equal to pid
pid = 0 signal send to all process whose process group ID is equal to the process group ID of sender, and for which the permission to send a signal
pid = -1 signal send to all process for which the process has permission to send that signal
pid < -1 signal send to all process whose process ID is equal to the absolute value of pid and for which the process has permission to send a signal

If Kill() fails, no signal will be send, and return -1 and send error

Success it return 0 and send signal to any of the process specified by pid

Error :- EINVAL :- The value of the signal argument is an invalid or unsupported signal no
EPERM :- The process does not have permission to send the signal to any receiving process
ESRCH :- No process or group of process can be found corresponding to that specified by pid

int Kill (pid_t pid, int sig);

Sleep :- header file :- unistd.h

Return type :- unsigned

Parameter :- unsigned int

In single threaded programs sleep() may make use of SIGALRM
In multi threaded programs sleep() shall not make use of SIGALRM

Return value :- after time elapsed :- 0

Error :- no error

```
unsigned sleep(unsigned seconds);
```

abort :- header file :- stdlib.h

Return type :- void

Parameters :- void

It cause abnormal process termination to occur unless the signal SIGABRT is being caught and the signal handler does not return

Error :- No error

```
void abort(void);
```

pthread_create :- header file :- pthread.h

Return value :- int

Parameter :- thread, attr, fun^n, parameter of fun^n

The pthread_create() create a new thread with attribute specified by attr, within a process. If attr is NULL, the default attributes shall be used. If the attributes specified by attr are modified later the thread's attributes shall not be affacted. Upon successful completion It store the ID of created thread in the location referenced by thread.

Return value :- success :- 0
otherwise errorno

Error :- EAGAIN :- lack of resources, thread are more than limit
EPERM :- The caller does not have appropriate privileges to set the required scheduling parameters or scheduling policy
EINTR :- attr invalid

int pthread_create(pthread_t* restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine)(void*), void *restrict arg);

pthread_join :- header file :- pthread.h

Return value :- int

Parameter :- thread, **value_ptr

It shall suspend execution of the calling thread until the target thread terminates.
On success it call with a non-NULL value_ptr argument the value passed to pthread_exit() by terminating thread shall be made available in the location referenced by

value_ptr.

Return value :- Success :- 0

Error :- EDEADLK :- A deadlock was detected
ESRCH :- Thread not found
EINVAL :- no reference to joinable thread

int pthread_join(pthread_t thread, void ** value_ptr);

pthread_exit :- header file :- pthread.h

Return value :- void

Parameter :- void * value_ptr

An implicit call to pthread_exit() is made when a thread other than the thread in which main() was first invoked returns from the start routine that was used to create it. The fun return value shall serve as the thread's exit status.

The behaviour of pthread_exit() is undefined if called from a cancellation cleanup handler or destructor fun that was invoked as a result of either an implicit or explicit call to pthread_exit()

The process shall exit with an exit status 0 after the last thread has been terminated The behaviour shall be as if the implementation called exit() with 0 argument at thread termination time

Return value :- It can't return to its caller

Error :- No error

```
void pthread_exit (void *value_ptr);
```

pthread_self :- header file :- pthread.h

Return value :- pthread_t

Parameter :- void

Return value :- It always be successful and no return value is reserved to indicate an error

Error :- No error

```
pthread_t pthread_self (void);
```

pthread_mutex_init :- header file :- pthread.h

Return value :- int

Parameter :- pthread_mutex_t, Const pthread_mutexattr_t