

System Calls

Jan 22, 2001

`pid_t fork(void) ;`

- 1) Creates child process
- 2) Returns pid of child to parent process,
0 to child process.
- 3) Child process is an exact copy of parent process except different pid and different ppid.
- 4) After fork, parent and child process continue executing their own next statements after fork statement.
- 5) There is no order of execution between parent and child process, order depends on the system scheduler.

6) One way to control parent, child process:

```
pid_t pid;
if( ( pid = fork() ) == 0 )
{ /* code for child process */

}
else
{ /*code for parent process */

}
```

`pid_t getpid(void);` returns process id.

`pid_t getppid(void);` returns pid of parent.

`pid_t getuid(void);` returns user id of the process.

`pid_t getgid(void);` returns group id of the process.

`pid_t geteuid(void);` returns effective user id of the process.

`pid_t gettuid(void);` returns effective group id of the process.

A system call, **exec** overlays the current program with a new code for another executable file and begins execution at its entry point, which means the process begins to execute a new program.

Suffix of exec:

l: argument list

v: argument array
p: automatic pass of PATH for binary search
e: pass environment variables as an argument (no automatic pass of environment variables)

```
Int  execlp ( const    char  *file,  
             const    char  *arg0, ... const char  *argn,  
             char *    NULL );
```

- 1) From the point of view of calling process, exec never returns if succeeds.
- 2) When fails, child returns -1, then keeps executing from the next statement, which means child keeps executing the rest of code as well as its parent.

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

- 1) Calls all the functions registered using atexit fn.
- 2) Calls `_cleanup`, standard i/o library fn to flush and close all the I/O buffers .
- 3) Calls `_exit`, system call.

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

- 1) Flushes and closes all the open file descriptors.
- 2) Notifies parent process the this process is terminating.
- 3) Returns status information of the process to its parent.
- 4) `_exit` is used by a child process when it is necessary not to interfere with I/O buffers shared by parent and child processes.

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

- 1) Waits for a child process to terminate. When the child is not terminated, parent will be blocked until the child is terminated.
- 2) Returns child pid, stores status info at the location pointed by `*stat_loc`. Returns -1 if no child.
- 3) Wait on only one child whichever is terminated first.

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

- 1) wait for the specified child to terminate. The parent will be blocked until the specified child is terminated.

```
# include    <sys/time.h>  
#include    <sys/resources.h>
```

```
int  getrusage( int who,      struct  rusage      *rusage  );
```

1) Returns info about the resources utilized by the current process or all the terminated children.

2) **who** : RUSAGE_SELF or RUSAGE_CHILDREN

```
3) struct  rusage      {  
    struct  timeval    ru_utime;    /* user time used    */  
    struct  timeval    ru_stime;    /* system time used  */
```

....

```
}
```

```
4) struct  timeval      {  
    long   tv_sec;      /* second */  
    long   tv_usec;    /* microsecond */  
};
```