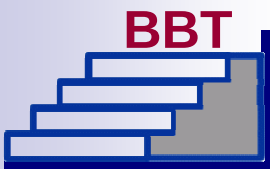


CPP

Unix Process

Process

- Processes are the primitive units for allocation of system resources. Each process has its own address space and (usually) one thread of control.
- Processes are organized hierarchically. Each process has a parent process which explicitly arranged to create it. The processes created by a given parent are called its child processes.

Process descriptor (for linux)

- Process attributes
- Process relationships
- Process memory space
- File management
- Signal management
- Process credentials
- Resource limits
- Scheduling related fields

Process Identification

- Your program should include the header files **unistd.h** and **sys/types.h** to use these functions.
- Data Type: **pid_t**
  - The pid_t data type is a signed integer type which is capable of representing a process ID.
- **pid_t getpid (void);**
  - The getpid function returns the process ID of the current process.
- **pid_t getppid (void);**
  - The getppid function returns the process ID of the parent of the current process.

Creating a Process

- The fork function is the primitive for creating a process. It is declared in the header file unistd.h.

  pid_t fork (void)

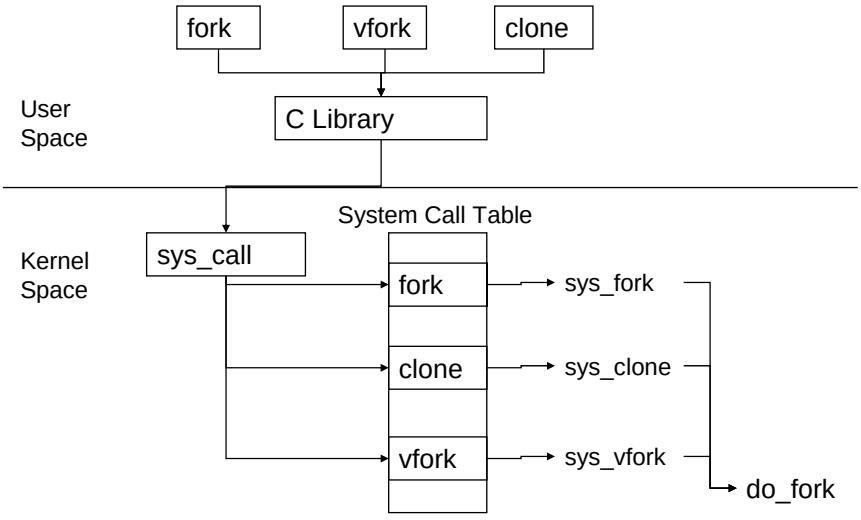
  pid_t vfork (void)

0:child process

-1:error

Other:parent process

Process Creation System Call



The specific attributes of the child process

- The child process has its own unique process ID
- The parent process ID of the child process is the process ID of its parent process
- The child process gets its own copies of the parent process's open file descriptors.
- However, the file position associated with each descriptor is shared by both processes
- The elapsed processor times for the child process are set to zero
- The child doesn't inherit file locks set by the parent process.
- The child doesn't inherit alarms set by the parent process.
- The set of pending signals (see Delivery of Signal) for the child process is cleared.

Excute a program

- int execv (const char *lename, char *const argv[])
- int execl (const char *lename, const char *arg0, ...)
- int execve (const char *lename, char *const argv[], char *const env[])
- execle (const char *lename, const char *arg0, char *const env[], ...)
- execvp (const char *lename, char *const argv[])
- int execlp (const char *lename, const char *arg0, ...)

Executing a File -cont.

- Executing a new process image completely changes the contents of memory, copying only the argument and environment strings to new locations.

Some attributes of the process are unchanged:

- The process ID and the parent process ID.
- Session and process group membership.
- Real user ID and group ID, and supplementary group IDs.
- Pending alarms.
- Current working directory and root directory. See Working Directory. In the GNU system, the root directory is not copied when executing a setuid program; instead the system default root directory is used for the new program.
- File mode creation mask.
- Process signal mask;
- Pending signals;
- Elapsed processor time associated with the process;

Process Completion

- **pid_t waitpid (pid_t pid, int *status-ptr, int options)**
  - The waitpid function is used to request status information from a child process whose process ID is pid.
  - A value of -1 or WAIT_ANY requests status information for any child process;
  - a value of 0 or WAIT_MYPGRP requests information for any child process in the same process group as the calling process;
  - and any other negative value − pgid requests information for any child process whose process group ID is pgid.
  - The options argument is a bit mask, use the WNOHANG flag to indicate that the parent process shouldn't wait; WUNTRACED flag to request status information from stopped processes as well as processes that have terminated.

Process Completion –cont.

- **pid_t wait (int *status-ptr)**
  - This is a simplified version of waitpid, and is used to wait until any one child process terminates.

Process Completion Status

- Mcros are defined in the header file sys/wait.h.
- **int WIFEXITED (int status)**
  - This macro returns a nonzero value if the child process terminated normally with exit or _exit.
- **int WEXITSTATUS (int status)**
  - If WIFEXITED is true of status, this macro returns the low-order 8 bits of the exit status value from the child process.
- **int WIFSIGNALED (int status)**
  - This macro returns a nonzero value if the child process terminated because it received a signal that was not handled.
- **int WTERMSIG (int status)**
  - If WIFSIGNALED is true of status, this macro returns the signal number of the signal that terminated the child process.

Process Completion Status -cont

- **int WCOREDUMP (int status**)
  - This macro returns a nonzero value if the child process terminated and produced a core dump.
- **int WIFSTOPPED (int status)**
  - This macro returns a nonzero value if the child process is stopped.
- **int WSTOPSIG (int status)**
  - If WIFSTOPPED is true of status, this macro returns the signal number of the signal that caused the child process to stop.

system function

- **int system (const char * command)** → stdlib.h

Job control

- Job control refers to the protocol for allowing a user to move between multiple process groups (or jobs) within a single login session.

Process group

- A single command may run just one process—but often one command uses several processes.
- The processes belonging to a single command are called a **process group or job**. This is so that you can operate on all of them at once.

Session

- A **session** is a larger group of processes. Normally all the processes that stem from a single login belong to the same session.

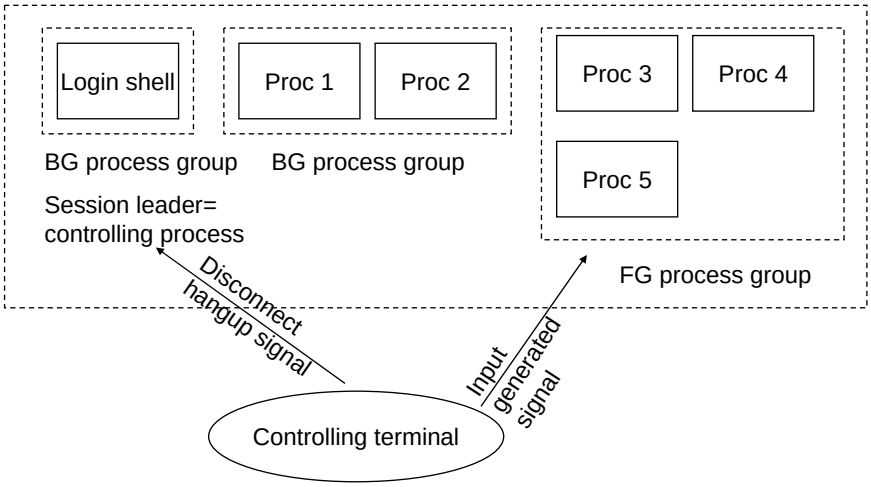
Controlling Terminal

- A session can have a single controlling terminal. This is usually the terminal device on which we log in.
- The session leader that establishes the connection to the controlling terminal is called the controlling process.
- The process groups within a session can be divided into a single foreground process group and one or more background process groups.

Controlling Terminal –cont.

- If a session has a controlling terminal, it has a single foreground process group, and all other process groups in the session are background process groups.
- Whenever we type the terminal's interrupt key, this causes the interrupt signal be sent to all processes in the foreground process group.
- If a modem (or network) disconnect is detected by the terminal interface, the hang-up signal is sent to the controlling process (the session leader).

Process groups and sessions showing controlling terminal



Orphaned Process Groups

- the parent of every member is either itself a member of the group or is not a member of the group's session.
- when the parent terminates, POSIX.1 requires that every process in the newly orphaned process group that is stopped be sent the hang-up signal (SIGHUP) followed by the continue signal (SIGCONT).

Process Group Functions

- **pid_t setsid (void)**
  - The setsid function creates a new session. The calling process becomes the session leader, and is put in a new process group whose process group ID is the same as the process ID of that process.
- **pid_t getsid (pid_t pid)**
- **pid_t getpgrp (void)**

setpgid

- **int setpgid (pid_t pid, pid_t pgid)**
  - The setpgid function puts the process pid into the process group pgid. As a special case, either pid or pgid can be zero to indicate the process ID of the calling process.
  - A process can set the process group ID of only itself or any of its children.
  - it can't change the process group ID of one of its children after that child has called one of the exec functions.

Daemon process

- Daemons are processes that live for a long time.
  - They are often started when the system is bootstrapped and terminate only when the system is shut down.

Coding Rules

- call umask to set the file mode creation mask to 0.
- fork(), exit the parent
- call setsid() to create a new session
- chdir to root, so that unmount can take place
- Unneeded file descriptors should be closed.
- close the main descriptors STDIN, STDOUT, STDERR