Interprocess Communication in Linux

Interprocess communication in Linux can done using shared memory or message passing.

1. Shared Memory

Shared Memory is an efficeint means of passing data between programs. One process creates a shared memory segment that other processes (if permitted) can access. This is done by using *shmget*( ).

```
int shmget(key_t key, size_t size, int shmflg);
```

When a *shmget*( )call succeeds, it returns the shared memory segment ID. This function is also used to get the ID of an existing shared segment (from a process requesting sharing of some existing memory portion).

Once created, a shared segment can be attached to a process address space using s*hmat*( ). *shmat*( ) returns a pointer, *shmaddr*, to the head of the shared segment associated with a valid *shmid*. *shmdt*( ) detaches the shared memory segment located at the address indicated by *shmaddr*.

```
void *shmat(int shmid, const void *shmaddr, int shmflg);

int shmdt(const void *shmaddr);
```

Once attached, a process can read or write to the segment, as allowed by the permission requested in the attach operation.

2. Message Passing (Linux pipes)

A pipe is a direct (in memory) I/O channel between two processes. Often it is used together with the system calls *fork* and exec to make multiple processes cooperate and perform parts of a larger task. A pipe is created using the pipe system call:

#include <unistd.h>

int pipe(int *fildes*[2]);

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array *fildes* is used to return two file descriptors referring to the ends of the pipe. *fildes* [0] refers to the read end of the pipe. *fildes* [1] refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe. read and write system calls are used for I/O through pipes.