

Write three programs. The first program will be the parent process the other two will be child processes. The parent process will start two child processes (*fork* system call), and then wait for a signal from child number 1 (*pause* system call.) After receiving a signal from child 1 (*signal* system call), the parent process will send a signal to the second child, and wait for the termination of the first child (*waitpid* system call). Once the first child terminates, the parent process will wait for the second child to terminate, and then will exit.

The first child will change programs (*execv* system call) and then will busy wait for a while (for loop, count to 1,000,000,000). Then it will signal the parent process (*getppid* system call, and *kill* system call). After signaling the parent, the first child will terminate.

The second child will also change programs and will then wait for a signal from the parent (*pause* system call.) After receiving a signal from the parent (*signal* system call), the second child process will terminate.

Each process will print messages on *stdout* to show what it is doing. An example of the printout of the parent process is given below. The two child processes should do something similar. In general, parent and child messages will be interspersed, and may even be garbled.

```
Parent: Forking child 1
Parent: Forking child 2
Parent: Waiting for signal
Parent: Signal Received
Parent: Waiting for child 1 termination
Parent: Waiting for child 2 termination
Parent: Exiting
```

Here are the system calls you will need. Most of these require you to add *#include* statements to your program. Use the UNIX *man* command to find out what they are. Use signal SIGUSR1 to signal back and forth between parent and child. The correct form of the *man* command for system calls is given below. If you leave off the 2, you may get the documentation for an identically-named shell command instead of what you want.

## man 2 kill

```
pid_t getppid( )
pid_t fork( )
char * INull = NULL;
int execv(ProgramPath,INull)
pid_t waitpid(pid_t,NULL,NULL)
sighandler_t signal(SIGUSR1,Handler) -----> void *Handler(int x)
int pause( )
int kill(pid_t, SIGUSR1)
```