Many of the process initialization details depend on the C runtime environment as well as the calling conventions — one cannot write the code to start a process without understanding the details. For example, on an x86 platform, *create* arranges for arguments to be placed on the runtime stack; on an ARM platform, *create* places some arguments in registers and others on the stack. The code that pushes arguments may be difficult to understand because *create* copies the arguments directly from its own runtime stack onto the stack that it has allocated for the new process. To do so, it finds the address of the arguments on its own stack and moves through the list using pointer arithmetic. The following example shows how *create* forms the stack on an x86 platform.

```
/* create.c - create, newpid */

#include <xinu.h>

local   int newpid();

/*------------------------------------------------------------------------
 *  create  -  Create a process to start running a function on x86
 *------------------------------------------------------------------------
 */
pid32   create(
          void          *funcaddr,      /* Address of the function      */
          uint32        ssize,          /* Stack size in bytes          */
          pri16         priority,       /* Process priority > 0         */
          char          *name,          /* Name (for debugging)         */
          uint32        nargs,          /* Number of args that follow   */
          ...
        )
{
        uint32          savsp, *pushsp;
        intmask         mask;           /* Interrupt mask               */
        pid32           pid;            /* Stores new process id        */
        struct  procent *prptr;         /* Pointer to proc. table entry */
        int32           i;
        uint32          *a;             /* Points to list of args       */
        uint32          *saddr;         /* Stack address                */

        mask = disable();
        if (ssize < MINSTK)
                ssize = MINSTK;
        ssize = (uint32) roundmb(ssize);
        if ( (priority < 1) || ((pid=newpid()) == SYSERR) ||
             ((saddr = (uint32 *)getstk(ssize)) == (uint32 *)SYSERR) ) {
                restore(mask);
                return SYSERR;
        }
```