Loading and Executing a Child Process

by Kip Irvine. Last update: 12/11/2003

MS-DOS has always taken a fairly straightforward approach to loading and executing programs. From the start, it was designed to be a simple, single-tasking operating system.

Many applications rely on other programs to perform useful services. Suppose you were using a text editor, and you needed to view a calendar. You wouldn't want to have to close the editor to view a calendar, of course, but under MS-DOS, you could not run both programs at the same time. The text editor's author could include code to display calendars, but it would increase the size of the program and probably be a poor imitation of existing calendar programs. The answer to this problem, of course, would be to run a calendar program directly from the text editor.

MS-DOS provides a convenient hook that lets you launch any executable program much in the manner of a subroutine call. The calling program is suspended, the called program executes, and when it finishes, control returns to the calling program.

There are small tradeoffs, of course. First, the calling program, which we will call the *parent* from now on, must resize the default memory allocation assigned to it by MS-DOS when it was loaded. The allocation is far more generous than necessary, because MS-DOS doesn't know how much memory you plan to use during the program's execution.

Second, the parent program must construct a structure called a *parameter block* that contains pointers to its own MS-DOS environment, a command-line string, and other miscellaneous information. The parameter block is passed to the child program when INT 21h is called.

Parameter passing is a little awkward, but easily possible. You can pass a string on the child program's command line, as if the string were typed at the MS-DOS prompt. You can also create a new environment block for the child program., up to 32K in size, and store parameters in the block. The block is an array of null-terminated strings containing system variables such as PATH, PROMPT, and COMPSEC.

When it exits, the child program can optionally return an integer *return code* between 0-255. It does this by passing the integer in AL when it calls INT 21h Function 4Ch to terminate. The return code can be retrieved by calling INT 21h Function 4Dh.

Many programs indirectly execute MS-DOS commands. They do this by loading a second copy of the Command processor (command.com or cmd.exe) and passing it arguments on the command line. We show on this Web site how to customize the Tools menu of TextPad and Visual Studio to assemble and link MASM programs. A typical command line is as follows:

Copyright 2002 Kip R. Irvine. All rights reserved.

```
command.com /C make16.bat AddSub
```

Command.com (called the *command processor*) executes the batch file, passing it the name "AddSub". When the batch file finishes, command.com is removed from memory and control returns to the text editor.

A Simple Example

Let’s take a look at a simple program named Parent that calls a child program named Child1. The parent program has been kept as minimal as possible. It does not use the Irvine16 library because the library’s default stack allocation is rather large. Also, the library functions would increase the size of our program.

Example 1. The Parent.asm Program

```
TITLE (Parent.asm)

.model small
.stack 100h
.286

cr = 0Dh
lf = 0Ah

PARAMBLOCK STRUC
   ofsEnvironSeg WORD  0         ; ptr to environ segment
   pCmdLine      DWORD cmdLine   ; ptr to command tail
   pFcb1         DWORD fcb1      ; ptr to file control block 1
   pFcb2         DWORD fcb2      ; ptr to file control block 2
PARAMBLOCK ENDS

.data
cmdLine BYTE 0,cr          ; command tail

fcb1 BYTE 37 DUP(0)
fcb2 BYTE 37 DUP(0)

pBlock PARAMBLOCK <>

child BYTE "child1.exe",0

.code
main PROC
   mov ax,@data
   mov ds,ax

; Resize the current program's memory allocation. ES must point
; to the PSP when calling Function 4Ah.
```

```
    mov bx,50h              ; required paragraphs (ES:BX)
    mov ah,4Ah              ; resize memory blocks
    int 21h
    jc  quit                ; quit if failed

; Prepare to call the first child program, passing it
; a command line.

    mov ax,SEG pBlock       ; must set up ES
    mov es,ax
    mov dx,OFFSET child     ; DS:DX points to EXE filename
    mov bx,OFFSET pBlock    ; ES:BX points to parameter block
    mov ax,4B00h            ; Function 4Bh, subfunction 00h
    int 21h

quit:
    mov ax,4C00h
    int 21h
main ENDP
END main
```

The PARAMETERBLOCK structure contains only pointers. The **ofsEnvironSeg** field is zero, indicating to MS-DOS that we want to pass a copy of the parent program's environment string to the child program. The **pCmdLine** field points to a blank command tail. The **pFcb1** and **pFcb2** fields are only used by early MS-DOS programs:

```
PARAMBLOCK STRUC
    ofsEnvironSeg WORD  0         ; ptr to environ segment
    pCmdLine      DWORD cmdLine   ; ptr to command tail
    pFcb1         DWORD fcb1      ; ptr to file control block 1
    pFcb2         DWORD fcb2      ; ptr to file control block 2
PARAMBLOCK ENDS
```

We also declare a structure variable:

```
pBlock PARAMBLOCK <>
```

The child program's executable filename must be a null-terminated string. The file can be in the same directory as the parent program, or it can be specified by a pathname. Following is the filename declaration in our parent program:

```
child BYTE "child1.exe",0
```

If you wanted to put the program in the directory one level higher than the parent's directory, you could supply a relative path:

```
child BYTE "..\child1.exe",0
```

Or, you could also give a complete path specificication including the drive letter:

```
child BYTE "D:\MyFiles\child1.exe",0
```

INT 21h Function 4Ah resizes the memory allocation used by a program. The number of paragraphs shown here is a rough estimate, based on the sizes of segments listed in the program's map file. The function sets the Carry flag if it cannot resize memory:

```
mov bx,50h              ; required paragraphs (ES:BX)
mov ah,4Ah              ; resize memory blocks
int 21h
jc  quit                ; quit if failed
```

Following is a listing of the map file for this program. The stop address of the last segment (stack) is 18Fh, which indicates that the program uses 190h bytes:

```
 Start  Stop   Length Name                   Class
 00000H 00022H 00023H _TEXT                 CODE
 00024H 00088H 00065H _DATA                 DATA
 00090H 0018FH 00100H STACK                 STACK

 Origin   Group
 0002:0   DGROUP

Program entry point at 0000:0000
```

To convert bytes to paragraphs, divide the number of bytes by 10h and add 1. Using the current example, (190h / 10h) + 1 = 20h. At the same time, it's better to overestimate the amount of memory required by the parent program, to be absolutely sure that the child program does not overwrite the parent program's memory space.

> We should keep in mind that our parent and child programs run in Real-address mode, which has none of the boundary protections built into Protected mode!

INT 21h Function 4Bh, subfunction 00h loads and executes a child program. The DS:DX registers must point to the child program's filename, and ES:BX must point to the parameter block:

```
mov ax,SEG pBlock       ; must set up ES
mov es,ax
mov dx,OFFSET child     ; DS:DX points to EXE filename
mov bx,OFFSET pBlock    ; ES:BX points to parameter block
mov ax,4B00h            ; Function 4Bh, subfunction 00h
int 21h
```

If the function fails, the Carry flag is set. This would happen if either there was not enough free memory to run the child program, or the child program's executable file was not found.

The Child Program

The child program is simple bordering on trivial. It just displays a string to let us know it was executed:

```
TITLE  Child1.asm

INCLUDE Irvine16.inc
.data
str1 BYTE "Child program number 1 executing...",0dh,0ah,0
.code
main PROC
   mov ax,@data
   mov ds,ax
   mov dx,OFFSET str1
   call WriteString
   mov ax,4C00h
   int 21h
main ENDP
end MAIN
```

Passing a Command Tail

The next step in our simple program's evolution is to pass a command tail to the child program. The MS-DOS command tail is received as if the child program were run at the command prompt. The following command runs child2.exe, passing it a command tail containing "ABCDEFG":

```
C:\MyProgs> child2 ABCDEFG
```

You can read about the command tail on pages 142 and 485.

Let's add some features to the Parent2 program that were not in the first version. For convenience, we've created a simple macro named **Write** that displays a string. Its one parameter is the name of a string variable:

```
;-----------------------------------------------------------
Write MACRO varName
;
; Display a string on the console
;-----------------------------------------------------------
   pusha
   mov ah,40h              ; write to file/device
   mov bx,1                ; console handle
   mov dx,OFFSET varName
   mov cx,SIZEOF varName
   int 21h
   popa
ENDM
```

The **cmdLine** variable not includes a command tail that will be passed to the child program. Notice that it must begin with a space and must end with 0Dh (the ASCII carriage return character):

```
cmdLine BYTE 8," ABCDEFG",cr
```

As explained on page 485, this layout is exactly the same used by MS-DOS at offset 81h in a program's Program Segment Prefix area.

Next, we initialize the **pCmdLine** field of the PARAMBLOCK structure to the segment and offset of **cmdLine**, the variable that holds the command tail:

```
mov dx,OFFSET child
    mov WORD PTR pBlock.pCmdLine,OFFSET cmdLine
    mov WORD PTR pBlock.pCmdLine+2,SEG cmdLine
    mov bx,OFFSET pBlock
    mov ax,4B00h            ; Execute child process
    int 21h
```

The field is a doubleword, so the WORD PTR operator is used to access it one word at a time.

When you run the Parent2 program, it executes Child2, and the latter displays the command tail. A message displays when control returns to the parent program:

```
ABCDEFG
...back in parent program
```

Example 2. The Parent2 Program

```
Title (Parent2.asm)

.model small
.stack 100h
.286

cr = 0Dh
lf = 0Ah

;-----------------------------------------------------------
Write MACRO varName
;
; Display a string on the console
;-----------------------------------------------------------
    pusha
    mov ah,40h              ; write to file/device
    mov bx,1                ; console handle
    mov dx,OFFSET varName
    mov cx,SIZEOF varName
```

```
    int 21h
    popa
ENDM

PARAMBLOCK STRUC
    ofsEnvironSeg WORD  0         ; ptr to environ segment
    pCmdLine      DWORD 0         ; ptr to command tail
                  DWORD fcb1      ; ptr to file control block 1
                  DWORD fcb2      ; ptr to file control block 2
PARAMBLOCK ENDS

.data
str1 BYTE "Cannot resize memory block",cr,lf,0
str2 BYTE "Error trying to execute child process",cr,lf,0
str3 BYTE "...back in parent program",cr,lf,0
cmdLine BYTE 8," ABCDEFG",cr
fcb1 BYTE 37 DUP(0)
fcb2 BYTE 37 DUP(0)
pBlock PARAMBLOCK <>
child BYTE "child2.exe",0

.code
main PROC
    mov ax,@data
    mov ds,ax

; Shrink this program’s memory usage.
    mov bx,50h              ; required paragraphs (ES:BX)
    mov ah,4Ah              ; resize memory block
    int 21h
    jc  CannotResize

; Set ES to the current program's data segment.
    mov ax,@data
    mov es,ax

; Prepare to call the child program. Set the pCmdLine
; field of the parameter block to the segment/offset of the
; command tail.
    mov dx,OFFSET child
    mov WORD PTR pBlock.pCmdLine,OFFSET cmdLine
    mov WORD PTR pBlock.pCmdLine+2,SEG cmdLine
    mov bx,OFFSET pBlock
    mov ax,4B00h            ; Execute child process
    int 21h
    jc  CannotExecute
    Write str3              ; "Back in parent program"
```

```
; Get the child program's exit type and return code.
   mov ah,4Dh
   int 21h                ; AH = exit type, AL = return code
   jmp quit

; Error messages ---------------------------------
CannotExecute:
   Write str2
   jmp  quit

CannotResize:
   Write str1
   jmp  quit

; End of program ---------------------------------
quit:
   mov ax,4C00h
   int 21h
main ENDP
END main
```

Example 3. The Child2 Program

The Child2 program uses a loop to retrieve and display each character in the command tail passed from the parent program. Note that ES already points to the Program Segment Prefix when the program begins to execute:

```
TITLE  Child2.asm

.model small
.stack 100h
.code
main PROC
   mov ax,@data
   mov ds,ax

; Display the command tail.
   mov cx,0
   mov cl,es:[80h]        ; get length byte
   mov si,82h             ; first nonblank byte

L1:mov dl,es:[si]         ; get byte from PSP
   mov ah,2               ; INT 21h Function 2
   int 21h                ; display on console
   inc si
   loop L1                ; continue until CX = 0

   mov dl,0dh             ; output carriage return
```

```
    int 21h
    mov dl,0ah              ; output line feed
    int 21h

    mov ax,4C02h            ; return value = 02
    int 21h
main ENDP
END main
```

You may recall that the segment override (ES:) is required when accessing data located somewhere other than the DATA segment. You can read more about segment overrides on page 589:

```
mov dl,es:[si]
```

The Child2 program also uses INT 21h Function 4Ch to pass a return code back to the parent program. It's up to the Parent2 program to call Function 4Dh if it wants to retrieve the return code:

```
mov ah,4Dh
int 21h
```

The value returned in AH is called the *exit type*, and the value in AL is the *return code*. You can read more about this function on page 655. The Child2 program passes back 02 as its return code, so the values of AH and AL in the Parent2 program are 00 and 02, respectively.

Executing a Batch File

A program that executes a child process can easily be made to execute a batch file. To do this, you must execute the command processor shell program. In Windows 95/98, the program is named command.com; in Windows 2000 and XP, you can execute either command.com or cmd.exe (preferred). You must supply a complete path name when specifying the filename, as in the following example:

```
child BYTE "c:\windows\system32\cmd.exe",0
```

The batch file's name is passed as a command-line parameter, preceded by the /C option that tells the command processor to return as soon as the batch file ends. The filename must contain the complete execution path, including drive letter. For example,to run the sayHello.bat file located in the c:\temp directory, the command line is defined as follows:

```
cmdLine BYTE 0," /C c:\temp\sayHello.bat",cr
cmdLineSize = (SIZEOF cmdLine) - 1
```

The cmdLineSize constant conveniently calculates the number of bytes in the command line, so we execute a statement later in the program that initializes the first byte of cmdLine to its own size:

```
mov cmdLine,cmdLineSize
```

Example 4 shows a complete program that runs the sayHello.bat file. The latter is a simple file

that displays a one-line message:

```
@ECHO The sayHello.bat file says HELLO.
```

When we run the program, the following lines display in the console window:

```
The sayHello.bat file says HELLO.
...back in calling program
```

Example 4. The runBat.asm Program

```
TITLE runBat.asm

.model small
.stack 100h
.286

cr = 0Dh
lf = 0Ah

;-----------------------------------------------------------
Write MACRO varName
;
; Display a string on the console
;-----------------------------------------------------------
   pusha
   mov ah,40h             ; write to file/device
   mov bx,1               ; console handle
   mov dx,OFFSET varName
   mov cx,SIZEOF varName
   int 21h
   popa
ENDM

PARAMBLOCK STRUC
   ofsEnvironSeg WORD  0  ; ptr to environ segment
   pCmdLine      DWORD 0  ; ptr to command tail
                 DWORD fcb1; ptr to file control block 1
                 DWORD fcb2; ptr to file control block 2
PARAMBLOCK ENDS

.data
str1 BYTE "Cannot resize memory block",cr,lf,0
str2 BYTE "Error trying to execute batch file",cr,lf,0
str3 BYTE "...back in calling program",cr,lf,0

Comment @
 The batch filename must include a complete path. The leading /C
 tells the command processor to close as soon as the batch file
finishes.
```

```
@
cmdLine BYTE 0," /C c:\temp\sayHello.bat",cr
cmdLineSize = (SIZEOF cmdLine) - 1

fcb1 BYTE 37 DUP(0)
fcb2 BYTE 37 DUP(0)

pBlock PARAMBLOCK <>

; Complete path of MS-DOS command processor:
child BYTE "d:\windows\system32\cmd.exe",0

.code
main PROC
   mov ax,@data
   mov ds,ax

Comment @
 Shrink the current program's memory allocation to make room
 for the command processor. ES already points to the current
 environment segment block.The number of required paragraphs
 is a rough estimate, based on inspection of this program's MAP
file.
@
   mov bx,50h             ; required paragraphs (ES:BX)
   mov ah,4Ah             ; resize memory block
   int 21h
   jc  CannotResize

; Set ES to the current program's data segment. This is
; required when passing ES:BX to Function 4B00h.

   mov ax,@data
   mov es,ax

; Prepare to call the child program. Set the pCmdLine
; field of the parameter block to the segment/offset of the
; command line string.

   mov cmdLine,cmdLineSize

   mov dx,OFFSET child
   mov WORD PTR pBlock.pCmdLine,OFFSET cmdLine
   mov WORD PTR pBlock.pCmdLine+2,SEG cmdLine
   mov bx,OFFSET pBlock
   mov ax,4B00h           ; Execute child process
   int 21h
   jc  CannotExecute
```

```
    Write str3              ; "Back in parent program"

; Get the child program's exit type and return code.
    mov ah,4Dh
    int 21h
; AH = exit type (0 if terminated by Function 00h or 4Ch)
; AL = return code passed by child program
; Display these using CodeView (AH=00, AL=02)
    jmp quit

; Error messages ---------------------------------
CannotExecute:
    Write str2
    jmp  quit

CannotResize:
    Write str1
    jmp  quit

; End of program ---------------------------------
quit:
    mov ax,4C00h
    int 21h
main ENDP
END main
```

Suggestions for Further Study:

1. Read Ray Duncan's book, *Advanced MS-DOS*, 2nd Edition. It's out of print, but used copies are often available at online bookstores. It only covers through MS-DOS 3.0, but the explanations are very well done.

2. Write a menu-style program that lists a number of short tasks. Each task should be carried out by a separate executable program. Use a table lookup (array of program names) to select and execute each of the external programs.

3. Write a program that executes a batch file, using the method shown earlier in this article.