Transactor Lessons

Walter Cazzola

COLLABORATORS

	TITLE : Transactor Lessons	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Walter Cazzola	February 14, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Transactor Lessons

1.1 AmigaDOS Quinta Lezione

Quinta Lezione: Task e Processi (seconda parte)

eccoci dunque alla quinta puntata di questo corso. Con questa concludiamo, almeno per il momento, il discorso sui task ed i processi. Rivediamo le procedure che la dos.library ci mette a disposizione per manipolare i processi. Al solito, vedremo quelle più importanti:

1) struct MsgPort *CreateProc(STRPTR name, long pri, BPTR segList, long stackSize);

2) struct Process *CreateNewProc(struct TagItem *tags);

3) LONG RunCommand(BPTR seg, long stack, STRPTR paramptr, long paramlen);

4) LONG Execute(STRPTR string, BPTR input, BPTR output);

5) struct Process *FindCliProc(unsigned long num);

6) BPTR LoadSeg(STRPTR name);

7) void UnLoadSeg(BPTR seglist);

8) BPTR NewLoadSeg(STRPTR file, struct TagItem *tags);

(Nota: sia la CreateNewProc che la NewLoadSeg sono presenti nei doppi formati per ricevere le etichette sullo stack o tramite puntatore, come molte altre procedure introdotte con la versione 2.0 del SO).

La funzione Execute, funziona allo stesso modo della shell. Ovvero, fornendo una riga di comando completa, come se stessimo usando una shell, eseguirà il comando impartito. Inoltre, se viene specificato il file di input, questo sarà letto dopo aver interpretato la riga di comando. Se forniamo anche il file di output, le eventuali stampe del programma eseguito saranno scritte in quel file. Se passiamo NULL, verrà usata la shell dal quale il programma che ha chiamato la Execute è stato lanciato. Se il programma è stato avviato da Workbench, verrà aperta una console di output.

La funzione FindCliProc, serve (un pÓ come la FindTask) per ottenere il puntatore al task CLI interessato, quelli visibili con il comando 'status' della shell.

Bene. Adesso, prima di introdurre le altre procedure/funzioni, occorre spiegare le varie XXXLoadSeg, dal quale dipendono. Le funzioni LoadSeg/NewLoadSeg, servono per caricare da disco un modulo prodotto con il linker (eseguibile). I segmenti CODE, DATA e BSS vengono caricati in memoria e concatenati, pronti per essere usati. Nei documenti si accenna agli overlay, ed infatti con queste procedure è possibile gestirli.

La procedura NewLoadSeg, differisce dalla LoadSeg, per il fatto di avere un parametro in più e di essere stata (a detta dei documenti) migliorata. Il secondo parametro, che sarebbero poi una lista di tag, è inutile attualmente perchè non ci sono tag definiti.

Per liberare una lista di segmenti caricati con le LoadSeg/NewLoadSeg, è sufficente chiamare la UnLoadSeg con il puntatore restituito da queste funzioni.

Rimangono quindi, le funzioni CreateProc/CreateNewProc e RunCommand. Tutte e tre, servono per creare un nuovo processo tranne l'ultima, che in pratica funziona come una execl() di unix sostituendo il codice del chiamante con quello precedentemente caricato tramite LoadSeg/NewLoadSeg.

Vediamole una per volta.

struct MsgPort *CreateProc(STRPTR name,long pri,BPTR segList,long stackSize);

Questa funzione, dati in input nome logico del programma (ad esempio Prova), dimensione dello stack, priorità richiesta e lista dei segmenti, creerà un nuovo processo allocando le risorse necessarie ed inserendolo nella lista dello scheduler.

Al termine del programma, la seglist passata non verrà liberata. Sarà compito del processo padre o del processo figlio liberarla. Siccome questa procedura è ormai vecchiotta, i documenti consigliano caldamente di usare al suo posto la:

struct Process *CreateNewProc(struct TagItem *tags);

che come vedete ha un aspetto molto più 'pulito' .

Notare, che mentre la prima restituisce il puntatore alla porta messaggi del task, questa restituisce direttamente il puntatore al processo creato. Questa funzione, prende dal processo padre, il maggior numero di informazioni possibili nel caso queste non siano passate.

Il minimo di tag che DEVONO essere passati sono NP_SegList o NP_Entry. NP_SegList è il puntatore alla lista di segmenti caricati con la LoadSeg NP_Entry è il puntatore alla routine da chiamare. Nel file dos/dostags.h sono definite le varie tags da usare. Vediamone alcune:

#define NP_Seglist (NP_Dummy + 1)

/* seglist of code to run for the process */

#define NP_FreeSeglist (NP_Dummy + 2)

/* free seglist on exit - only valid for */

/* for NP_Seglist. Default is TRUE. */

#define NP_Entry (NP_Dummy + 3)

/* entry point to run - mutually exclusive */

/* with NP_Seglist! */

#define NP_Input (NP_Dummy + 4)

/* filehandle - default is Open("NIL:"...) */

#define NP_Output (NP_Dummy + 5)

/* filehandle - default is Open("NIL:"...) */

#define NP_CloseInput (NP_Dummy + 6)

/* close input filehandle on exit */

/* default TRUE */

#define NP_CloseOutput (NP_Dummy + 7)

/* close output filehandle on exit */

/* default TRUE */

#define NP_Error (NP_Dummy + 8)

/* filehandle - default is Open("NIL:"...) */

#define NP_CloseError (NP_Dummy + 9)

/* close error filehandle on exit */

/* default TRUE */

#define NP_CurrentDir (NP_Dummy + 10)

/* lock - default is parent's current dir */

#define NP_StackSize (NP_Dummy + 11)

/* stacksize for process - default 4000 */

#define NP_Name (NP_Dummy + 12)

/* name for process - default "New Process" */

#define NP_Priority (NP_Dummy + 13)

/* priority - default same as parent */

Da quì si può già vedere, che è molto comodo usare la funzione CreateNewProc per creare i propri processi figli.

Ultima funzione è la:

LONG RunCommand(BPTR seg,long stack,STRPTR paramptr,long paramlen);

che esegue un comando dal proprio processo o cli.

La differenza che passa tra questa e la Execute è che mentre la Execute crea un nuovo processo per eseguire il comando richiesto (come un doppio click su un'icona per intenderci), la RunCommand riutilizza il processo chiamante.

Ecco di seguito la struttura Process, che come abbiamo imparato le lezioni precedenti, include come primo elemento una struttura Task (ecco perchè i processi sono un superset di task).

Molti dei campi di questa struttura vengono inizializzati dal SO e conviene quasi sempre lasciarli così come sono se non utilizzando le relative procedure propense alla modifica di un determinato valore.

struct Process {

struct Task pr_Task;

struct MsgPort pr_MsgPort; /* This is BPTR address from DOS functions */

WORD pr_Pad; /* Remaining variables on 4 byte boundaries */

BPTR pr_SegList; /* Array of seg lists used by this process */

LONG pr_StackSize; /* Size of process stack in bytes */

APTR pr_GlobVec; /* Global vector for this process (BCPL) */

LONG pr_TaskNum; /* CLI task number of zero if not a CLI */

BPTR pr_StackBase; /* Ptr to high memory end of process stack */

LONG pr_Result2; /* Value of secondary result from last call */

BPTR pr_CurrentDir; /* Lock associated with current directory */

BPTR pr_CIS; /* Current CLI Input Stream */

BPTR pr_COS; /* Current CLI Output Stream */

APTR pr_ConsoleTask; /* Console handler process for current window */

APTR pr_FileSystemTask; /* File handler process for current drive */

BPTR pr_CLI; /* pointer to CommandLineInterface */

APTR pr_ReturnAddr; /* pointer to previous stack frame */

APTR pr_PktWait; /* Function to be called when awaiting msg */

APTR pr_WindowPtr; /* Window for error printing */

/* following definitions are new with 2.0 */

BPTR pr_HomeDir; /* Home directory of executing program */

LONG pr_Flags; /* flags telling dos about process */

void (*pr_ExitCode)(); /* code to call on exit of program or NULL */

LONG pr_ExitData; /* Passed as an argument to pr_ExitCode. */

UBYTE *pr_Arguments; /* Arguments passed to the process at start */

struct MinList pr_LocalVars;/* Local environment variables */

ULONG pr_ShellPrivate; /* for the use of the current shell */

BPTR pr_CES; /* Error stream - if NULL, use pr_COS */

}; /* Process */

Bene, la prossima lezione, continueremo a vedere le procedure della dos.library. Inizieremo a vedere le funzioni relative ai file ed ai cassetti.

Lezione Precedente Indice AmigaDOS

Indice Corsi Lezione Successiva