Processi e thread 7 Thread Windows

giovedì 16 ottobre 2025 16:34

in windows i thread sono caratterizzati da u nid numerico, un contesto (da sistemare dopo) , priorità base del thread , priorità dinamica del thread, affinità di processore , tempo di esecuzione e stato di uscita. Per i servizi : creazione , apertura (maniglia tornata), richiesta/modifica gestione thread , terminazione di thread, letture/modifica di contesto e servizi per bloccarli : **non esistono servizi sulla memoria virtuale** . Vediamole in dettaglio :

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES
                    lpThreadAttributes,
                    SIZE_T dwStackSize,
                    LPTHREAD_START_ROUTINE lpStartAddress,
                    LPvoid lpParameter,
                    DWORD dwCreationFlags,
                    LPDWORD lpThreadId)
```

Descrizione
- invoca l'attivazione di un nuovo thread

Restituzione
- un handle al nuovo thread in caso di successo, NULL in caso di fallimento.

In dettaglio : il primo parametro rappresenta un puntatore alla tabella che gestisce info sulla sicurezza, il secondo rappresenta la taglia dello stack, la terza è un puntatore ad indirizzo alla routine nella quale quel thread deve vivere (zona testo dove il thread esegue le attività), il quarto è un puntatore generico alla funzione nella quale vivrà il nuovo thread , il quinto sono dei flag e l'ultimo è un puntatore all'area di memoria dove vogliamo il codice numerico. Quindi questa syscall ha un doppio ritorno : sia il codice numerico sia una maniglia. Vediamo ora come <u>completare solo il thread</u> senza completare l'intero processo :

```
VOID ExitThread(DWORD dwExitCode)
```

Descrizione
- invoca la terminazione di un thread

Parametri
- dwExitCode specifica il valore di uscita del thread terminato

Restituzione
- non ritorna in caso di successo

Per catturare invece il valore di uscita di un thread :

```
int GetExitCodeThread(
        HANDLE hThread,
        LPDWORD lpExitCode
        )
```

Descrizione
- richiede lo stato di terminazione di un thread

Parametri
- hThread: handle al processo
- lpExitCode: puntatore all'area dove viene scritto il codice di uscita

Restituzione
- 0 in caso di fallimento

Vediamo un esempio :

```c
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>


DWORD ThreadFiglio();


int main(int argc, char *argv[]) {

        HANDLE hThread;
        DWORD hid;
        DWORD exit_code;

        hThread = CreateThread(NULL,
                0,
                (LPTHREAD_START_ROUTINE)ThreadFiglio,
                NULL,
                NORMAL_PRIORITY_CLASS,
                &hid);



        if (hThread == NULL)  printf("Thread creation failure\n");
        else {
                WaitForSingleObject(hThread, INFINITE);
                GetExitCodeThread(hThread, &exit_code);
                printf("child thread exited with code %d\n", exit_code);
                fflush(stdout);
        }
        printf("parent thread - give me an integer to make me exit:");
        fflush(stdout);

        scanf("%d", &exit_code);
}



DWORD ThreadFiglio()
{

        int x;
        printf("child thread - give me an integer to make me exit:");
        fflush(stdout);
        scanf("%d", &x);
        ExitThread(x);
}
```

Il cui output è il seguente: appena la eseguo il processo genera un thread figlio il quale aspetta un intero per terminare: finchè non termina il padre è in wait. Quindi quando il thread figlio finisce l'esecuzione (passando un intero), il controllo torna al padre che si risveglia e se gli passo un intero finisce tutto. Ma cosa succede se passiamo ad esempio una stringa a posto dell'intero ? Si ha inconsistenza dei dati in quanto la libreria chiamata è la stessa. Vediamo ora un altro esempio : scrivere una stringa in un area di memoria, fatta attraverso più thread:

```c
int main(int argc, char** argv){
        HANDLE hThread;
        DWORD hid;
        int i;
        char *old_buffer;
        DWORD exit_code;
        char buffer[MAX_CHAR];
        int notfirst = 0;
        while (1) {
                printf("Inserisci stringa: ");
                fflush(stdout);
                scanf("%s", buffer);
                if (strcmp(buffer, "quit") == 0) break;
                /*string duplication
                        crea nuova area duplicando la stringa allocando memoria
                        ritirna il puntatore
                */
                old_buffer = strdup(buffer);
                if (notfirst) {
                        WaitForSingleObject(hThread, INFINITE);
                        GetExitCodeThread(hThread, &exit_code);
                }
                else notfirst = 1;
                //passo indirizzo di memoria dell'area duplicata
                hThread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)OrderThread,(LPVOID)old_buffer,NORMAL_PRIORITY_CLASS,&hid);
                if (hThread == NULL) {
                        printf("cannot create thread\n");
                        ExitProcess(1);
                }
        }
        if (notfirst) {
                WaitForSingleObject(hThread, INFINITE);
                GetExitCodeThread(hThread, &exit_code);
        }
        for (i = 0; i< string_number; i++) printf("String %d: %s\n", i, string_array[i]);
        ExitProcess(0);
}
```

```c
/*****************************************************************
this program allows sorting strings coming form the standar input
it uses one thread for input and one thread at a time for sorting
*****************************************************************/

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CHAR        1000
#define MAX_STRING      1000

int string_number = 0;
char *string_array[MAX_STRING];

void OrderThread(char *new_string) {
        DWORD exit_code;
        int i, j;
        //printf("Ordering thread active (inserting: %s)\n",new_string);
        for (i = 0; i < string_number; i++) {
                if (strcmp(new_string, string_array[i]) <= 0) {
                        for (j = 0; j<(string_number - i); j++) {
                                string_array[string_number - j] = string_array[string_number - j - 1];
                        }
                        break;
                }
        }
        string_array[i] = new_string;
        string_number++;
        exit_code = 0;
        ExitThread(exit_code);
}
```

Il cui output è un riordinamento delle stringhe passate dal main thread, appena passo in input la stringa quit.

Ultimo esempio usando la thread local storage (tls) :

```c
#include <windows.h>
#include <stdio.h>

#define EXPOSE

#ifdef EXPOSE
int * v[2];
#endif
//variabile di definizione nel thread
__declspec(thread) int  variable;

void ChildThread(char * me)
{
        printf("Child thread %d active - variable is %d\n", (int)me,variable);
        fflush(stdout);
#ifdef EXPOSE
        variable = (int)me + 33;
        v[(int)me] = &variable;
#endif
        Sleep(3000);
        ExitThread((DWORD)me);
}

int main(int argc, char** argv){
        //array di maniglie
        HANDLE hThread[2];
        int i;
        DWORD hid;
        variable = 1;
        for (i = 0; i < 2; i++){
                hThread[i] = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)ChildThread,(LPVOID)i,NORMAL_PRIORITY_CLASS,&hid);
                if (hThread[i] == NULL) {
                        printf("cannot create thread %d\n", i);
                        ExitProcess(-1);
                }
        }

#ifdef EXPOSE
        Sleep(1000);
        for (i = 0; i < 2; i++){
                printf("thread %d - variable value is %d\n", i, *v[i]);
        }
#endif
        for (i = 0; i < 2; i++){
                WaitForSingleObject(hThread[i], INFINITE);
        }

}
```

Il quale output è il seguente : creo due thread i quali assegnano il loro valore a questa variabile.