

# NANO NOYAU MULTITÂCHE

**U**n système multitâche permet l'exécution apparemment simultanée de plusieurs tâches distinctes. Prenons par exemple le cas d'une caisse enregistreuse. Celle-ci doit gérer plusieurs modules : un premier module scrutera le clavier et, à l'appui d'une touche, effectuera les opérations nécessaires. Un deuxième module s'occupera de l'impression des tickets. Un troisième module gèrera une liaison série qui permettra l'échange avec un ordinateur central de données concernant l'état actuel de la caisse, les nouveaux codes associés aux articles, etc. Un quatrième interrogera le centre carte bancaire pour avoir l'autorisation de débit. Ces quatre modules doivent opérer indépendamment les uns des autres. La frappe sur le clavier doit pouvoir être faite pendant l'impression des tickets, un nouveau client peut être traité pendant l'interrogation du centre carte bleu pour le client précédent...

Avec un système multitâche, la programmation de chaque tâche se fait indépendamment des autres tâches. Tout se passe comme si on disposait d'autant de processeurs qu'il y a de tâches. Sans système multitâche, la programmation de ce type d'application demande beaucoup d'astuce de la part du programmeur, et ce souvent au détriment de la lisibilité et de la maintenance du programme. Il existe deux types principaux de systèmes multitâches. Le multitâche coopératif et le multitâche préemptif. Dans un système multitâche coopératif (type windows 3.1), le changement de tâche est à l'initiative de chaque tâche. Il se fera soit de manière explicite en appelant la fonction qui commute les tâches, soit de manière implicite, lors de l'appel d'une fonction système qui elle-même demandera la commutation des tâches. Dans un système multitâche préemptif, la commutation d'une tâche à une autre se fait de manière autoritaire après l'écoulement d'un délai donné. Par exemple toute les millisecondes, la tâche active sera interrompue et une nouvelle tâche sera sélectionnée et reprendra son activité avec le même environnement que celui dans lequel elle avait été interrompue précédemment. En fait les

```

; Micro noyau multitâche assembleur 51
; Le micro noyau permet un multitâche simple coopératif.
; Quatre tâches au maximum peuvent être définies. Chaque tâche utilise
; une banque de registre distincte des autres tâches pour sauver ces propres
; registres R0..7. Les autres registres (PSW, ACC, B, DPTR) sont sauves dans
; le stack. Chaque tâche dispose de son propre stack qui est défini à
; l'initialisation des tâches.
RELOAD EQU 8192 ; période entre 2 interruption
_ZSEC EQU 244 ; 200000/RELOAD : env 2 secondes
_T_ZSEC EQU 61 ; 500000/RELOAD : env 1/2 seconde
_T0MSEC EQU 1 ; 10000/RELOAD : env 10 msec

NB_TSK EQU 3 ; nombre de tâches
DSEG AT NB_TSK*8 ; réserve la place des registres R[0..7]

; VARIABLES ASSOCIEES A LA GESTION DU TEMPS
ITACC: DS 1
ITPSW: DS 1 ; sauvegarde de PSW pendant l'it
TIME: DS 4 ; temps système (incrémenté dans it)

; VARIABLES ASSOCIEES A LA GESTION MULTITACHE
STACKS: DS NB_TSK ; sauvegarde des SP
; Réserve la place pour le stack des différentes tâches (excepté la tâche 0)
; Il faut réserver 7 octets +
; 2 octets pour l'interruption temps +
; 2 * n octets pour les niveaux d'appel de sous programmes +
; n octets poussés au niveau le plus profond dans le stack
STACK1: DS 7+2 ; juste la place nécessaire
STACK2: DS 7+2+2 ; un niveau d'appel en plus de TSKSWI
STACK: DS 1 ; début de la pile système
CSEG AT 0
JMP START ; début du programme
CSEG AT 0BH
JMP TIMER0 ; interruption timer0 (temps)

;----- PRIMITIVES DE GESTION MULTITACHE -----
; Initialise une tâche. R0 pointe sur le stack de la nouvelle tâche, A le
; numéro de la tâche, DPH:DPH pointe sur la procédure associée à la tâche.
; Il faut réserver au moins 9 octets dans le stack d'une tâche :
; - 5 octets pour la sauvegarde des registres PSW, ACC, B, DPL, DPH
; - 2 octets pour l'appel à TSKSWI (si TSKSWI est appelé directement)
; - 2 octets pour l'interruption temps (si elle est utilisée)
; A ces 9 octets, il faut ajouter tous les octets nécessaires pour les appels
; des sous programmes et les sauvegardes temporaires dans le stack.
; La tâche 0 (la première tâche) ne doit pas être initialisée puisque elle
; réside dans le stack «natif». Le stack de la tâche 0 est automatiquement
; initialisé au premier TSKSWI.
TSKINI: MOV @R0,DPL ; adresse de la tâche dans le stack
INC R0
MOV @R0,DPH
INC R0
MOV R1,A ; sauve le numero de la tâche
RR A
SWAP A ; numéro dans R51:RS0
MOV @R0,A ; sauve le futur PSW
MOV A,R0
ADD A,#4 ; ACC pointe sur le futur DPH
XCH A,R1 ; restaure le numéro de la tâche
ADD A,#STACKS
XCH A,R1
MOV @R1,A ; sauve le stack de la tâche
RET

; Commute de la tâche courante à la tâche suivante. Le numéro de la tâche
; correspond au numéro de la banque utilisée. Celui-ci est présent dans le
; registre PSW, bits R51:RS0. Les registres qui ne sont pas sauves dans les
; banques de registres sont sauves dans le propre stack de la tâche avant
; commutation. Le pointeur de pile de la tâche est stocké dans le tableau
; STACKS comprenant les pointeurs des différentes tâches.
; Appel compris : 42 cycles
TSKSWI: PUSH PSW ; sauve les registres critiques
PUSH ACC
PUSH B
PUSH DPL
PUSH DPH
; sauve R0 dans B pour économiser le stack
MOV B,R0
; prend le numéro de la tâche courante
MOV A,PSW
RL A
SWAP A
ANL A,#00000011B ; sauve le stack de la tâche courante
ADD A,#STACKS ; tableau des pointeurs de pile
MOV R0,A
MOV @R0,SP ; sauve le stack courant
; calcule le numéro de la tâche suivante
INC R0
CJNE R0,#STACKS+NB_TSK,TSKSWO ; tâche suivante
MOV R0,#STACKS ; compare au nombre max de tâche
; prend le nouveau stack
TSKSWO: MOV SP,@R0
; restaure R0 dans sa banque d'origine
MOV R0,B
; restaure les registres de la nouvelle tâche
POP DPH ; et restaure les registres critiques
POP DPL
POP B
POP ACC
POP PSW ; restaure la banque (R0..7)
RET

;----- PRIMITIVES DE GESTION DE TEMPS -----
; Le temps système TIME+0:TIME+1:TIME+2:TIME+3 est incrémenté à chaque
; interruption. Le PSW n'est pas sauve dans le stack pour éviter de
; d'encombrer le stack de CHAQUE tâche. TIME reboucle au bout de 407 jours.
TIMER0: MOV ITPSW,PSW ; sauve le PSW (habituellement PUSH)
MOV IITACC,A
CLR A
INC TIME+3 ; incrémente le temps système
CJNE A,TIME+3,TIME00 ; TIME[0..3]
INC TIME+2
CJNE A,TIME+2,TIME00

```





```

TIME00: INC QJNE TIME+1
        INC A,TIME+1,TIME00
        MOV PSW,ITPSW ; restaure le PSW
        RETI

; Prend le temps système
GTIME: CLR ETO ; inhibe l'it timer
        MOV R4,TIME+0 ; prend la valeur du temps
        MOV R5,TIME+1
        MOV R6,TIME+2
        MOV R7,TIME+3
        SETB ETO ; revalide le timer
        RET

; Additionne R[4..7] au temps système
ADDTIM: CLR ETO ; inhibe les it timer
        MOV A,TIME+3 ; additionne R[4..7]
        ADD A,R7,A ; au temps système
        MOV R7,A
        MOV A,TIME+2
        ADDC A,R6
        MOV R6,A
        MOV A,TIME+1
        ADDC A,R5
        MOV R5,A
        MOV A,TIME+0
        SETB ETO
        ADDC A,R4
        MOV R4,A
        RET

; Compare R[4..7] au temps système
CMPPTIM: CLR C
        MOV A,R7 ; compare le temps «cible» au temps
        CLR ETO ; système
        SUBB A,TIME+3
        MOV A,R6
        SUBB A,TIME+2
        MOV A,R5
        SUBB A,TIME+1
        MOV A,R4
        SUBB A,TIME+0
        SETB ETO
        RET

; Attend R4:R5:R6:R7.
; La valeur R[4..7] est additionnée au temps système puis est comparée à
; celui-ci jusqu'à ce que la différence soit négative.
WAIT: CALL ADDTIM ; additionne R[4..7] au temps système
; temps «cible» dans R[4..7]
WAIT0: CALL TSKSWI ; commute les tâches
        CALL CMPPTIM ; compare le temps cible au système
        JNB ACC,7,WAIT0 ; cible < système, attend
        RET

; Le Timer0 est programme en compteur 13 bits. Il génère une interruption tous
; les 8192 cycles (env 8 ms)
INITM0: ANL TMOD,#0FOH ; compteur 13 bits, horloge interne
        SETB TR0 ; compteur actif (TCON.4)
        SETB ETO ; valide l'interruption timer 0
        RET ; (si EA == 1)

----- PROGRAMME PRINCIPAL
START: MOV SP,#STACK-1 ; initialise le stack (tache 0)
        CALL INITM0 ; initialise le timer 0
; initialise les différentes tâches
        MOV A,#1 ; tâche 1
        MOV R0,#STACK1 ; pointeur sur le bas du STACK 1
        MOV DPTR,#TASK1 ; adresse de la première tâche
        CALL TSKINI ; initialise la tâche
        MOV A,#2 ; tâche 2
        MOV R0,#STACK2 ; pointeur sur le bas du STACK 1
        MOV DPTR,#TASK2 ; adresse de la première tâche
        CALL TSKINI ; initialise la tâche
        SETB EA ; autorise les interruptions activées

----- TACHE 0
; La tâche 0 fait clignoter P1.0 à 1 Hz si le bit P1.2 est à 1
TASK0: JB P1,2,TASK00 ; si doit clignoter ...
        CALL TSKSWI ; sinon commute sur la tâche 2
        AJMP TASK0 ; et boucle
TASK00: MOV C,P1,0 ; lit l'état du port
        CPL C ; inverse l'état
        MOV P1,0,C ; et re-stocke
        MOV R4,#0 ; charge la valeur de 1/2 seconde
        MOV R5,#0
        MOV R6,#HIGH_1_2SEC
        MOV R7,#LOW_1_2SEC
        CALL WAIT ; attend 1/2 seconde
        AJMP TASK0 ; et boucle

----- TACHE 1
; La tâche 1 fait clignoter P1.3 à la vitesse maximum
TASK1: MOV C,P1,3
        CPL C
        MOV P1,3,C
        CALL TSKSWI ; commute sur la tâche 1
        AJMP TASK1

----- TACHE 2
; La tâche 2 fait flasher P1.1 toute les 2 secondes
; Tous les 4 flash, inverse l'état de P1.2 qui conditionne la tâche 0
TASK2: MOV R0,#4
        MOV C,P1,2
        CPL C
        MOV P1,2,C
        SETB P1,1 ; met le P1.1 à 1
        MOV R4,#0
        MOV R5,#0
        MOV R6,#HIGH_10MSEC
        MOV R7,#LOW_10MSEC ; pendant 10 MSEC
        CALL WAIT
        CLR P1,1 ; puis remet à 0
        MOV R4,#0
        MOV R5,#0
        MOV R6,#HIGH_2SEC
        MOV R7,#LOW_2SEC ; pendant 2 SEC
        CALL WAIT
        DJNZ R0,TASK20 ; flashe 4 fois
        AJMP TASK2 ; et inverse l'état de cligne
    
```

systèmes multitâches préemptifs sont généralement beaucoup plus complexes. On peut associer à chaque tâche une priorité qui déterminera si une tâche peut ou non interrompre une autre tâche. D'autres mécanismes plus complexes permettent de s'assurer que des tâches ne seront pas complètement bloquées. D'autres encore gèrent les accès simultanés par plusieurs tâches à des ressources communes.

Le nano noyau multitâche permet un multitâche de type coopératif et l'exécution de quatre tâches indépendantes. Chaque tâche a à sa disposition l'ensemble des registres, exactement comme si elle exploitait seule le microprocesseur. Bien évidemment, lors de la commutation des tâches (appel à TSKSWI), le contenu des registres du microprocesseur doit être sauvegardé, et la tâche qui va être activée doit avoir le contenu de ses registres restauré. Les registres R0..R7 sont sauvegardés dans l'une des quatre banques de registres. La banque de registre est sélectionnée par les bits RS0 et RS1 du registre d'état PSW. Les registres restants (ACC, B, PSW, DPL, DPH) sont sauvegardés dans la pile de la tâche courante. Enfin le pointeur de pile (SP) de chaque tâche est stocké dans le tableau STACKS. La taille de la pile associée à chaque tâche doit être déterminée avec attention. Elle doit être au minimum suffisante pour permettre la mémorisation des registres courants et des adresses de retour de l'appel des sous-programmes et des interruptions. Les piles système associées aux tâches résident dans la mémoire interne du microprocesseur. L'initialisation des tâches activées se fait en appelant le sous-programme TSKINI. A l'appel de TSKINI, l'accumulateur contient le numéro de la tâche, R0 pointe sur le stack associé à la tâche, et DPTR contient l'adresse de la première instruction de la tâche. La tâche zéro utilise la pile courante et ne doit pas être initialisée avec TSKINI. Le listing d'exemple utilise le noyau multitâche pour faire «clignoter» des lignes de ports à des vitesses différentes. La commutation des tâches se fait de manière explicite pour la tâche 1 par l'appel à TSKSWI, et de manière implicite pour la tâche 0 et la tâche 2, les deux autres tâches par l'appel à WAIT qui appelle lui-même TSKSWI. D'autres routines pourront être créés qui elles-mêmes permettront un appel implicite à TSKSWI. Ce sera le cas par exemple d'une routine qui attend la frappe d'une touche, ou celle qui attend un caractère sur l'interface série. Enfin, le programme devra être écrit en tenant compte du fait que chaque tâche utilise une banque données différente. Attention donc aux accès aux registres R0..R7 directement en mémoire. On préférera la séquence MOV A,R0 suivi de MOV RI,A à l'instruction MOV RI,A R0. Le listing est disponible sur le serveur ERP.

END