TD interruptions

Exercice1 :

Type MEP = structure
mode:[utilisateur,système];
masque_dérout : [masqué,démasqué];
masque_autres_it : [masqué,démasqué];
compteur_ordinal : adresse (ou entier);
autres_informations ;
fin;
PILESYSTEME : PILE de MEP;
Savevecteur: MEP; /* structure dans laquelle on sauvegarde le vecteur d'interruption n° 8 correspondant au déroutement */
MEP_dérout: MEP; /* mot d'état utilisateur du déroutement*/
Savecontexte: MEP; /* sauvegarde du MEP du programme principal */
entier : nb; /* pour sauvegarder le nombre de blocs de la mémoire */

Programme principal;

Début

/* Initialisation du mot d'état de la procédure utilisateur de traitement de déroutement */

MEP_dérout->mode := utilisateur; /* mode = utilisateur; car c'est une procédure utilisateur qui traite le déroutement */

MEP_dérout->masque_dérout := masqué; /* voir remarque donnée ci-après */

MEP_dérout->masque_autres_it := masqué; /* Les interruptions externes doivent être masquées au début et à la fin de la procédure de traitement pour faire la sauvegarde et la restauration du contexte (la séquence de sauvegarde et de restauration de contexte doivent s'exécuter pendant une séquence non interruptible), et après on peut les démasquer pendant le traitement ou on peut les laisser masquées pendant toute l'exécution */

MEP_dérout ->compteur_ordinal := Traiter_dérout; (Traiter_dérout est le nom de la procédure de traitement)

/* Sauvegarde du vecteur d'interruption N° 8 du système avant de modifier l'entrée */

```
Lire_vecteur(8, savevecteur);
/*initialisation du vecteur d'interruption N° 8 correspondant au déroutement par le MEP de
  la procédure utilisateur  */
Modifier_vecteur(8, MEP_dérout);
nb=1;  /* on initialise par 1 car la mémoire contient au moins un bloc */
Tantque (vrai) faire
   début
        lire_mem(n*nb) ;  /* accéder au mot d'adresse n*nb */
        nb := nb+1;
   fin;
Etiq : écrire('La taille de la mémoire = ', nb*n, 'octects');
/* Rétablir la procédure normale de traitement des déroutements du système */
Modifier_vecteur(8, savevecteur);
Fin_du_programme_principal ;
```

Remarque :

/* Les interruptions de déroutements doivent rester masquées pendant toute l'exécution de la procédure utilisateur de traitement du déroutement pour éviter d'avoir une boucle infinie dans le cas où la procédure contient une erreur, par exemple une instruction qui va provoquer une division par zéro lors de l'exécution */

Procédure Traiter_dérout ;

```
Début
   Sauvegarder(contexte);
   Si cause = "adresse inexistante"  alors
        Dépiler(PILESYSTEME, Savecontexte);
        savecontexte->compteur_ordinal = Etiq;
        Restaurer(contexte) ;
        Charger_mep(savecontexte);    /* Relancer l'exécution du programme principal */
    Sinon /* Erreur :Rétablir la procédure du système de traitement des déroutements */
         Modifier_vecteur(8, savevecteur);
         Charger_mep(savevecteur);   /* lancer l'exécution de la procédure système pour
                                              traiter l'exception survenue
    Finsi;
```

Exercice 2 :

```
type MEP  =  structure
     mode:[utilisateur,système];
     masque_timer : [masqué,démasqué];
     masque_cptnul : [masqué,démasqué];
     masque_autres_it : [masqué,démasqué];
     masque_dérout :[masqué, autorisé] ;
    compteur_ordinal : adresse (ou entier);
    fin;
Hms = structure
     h : entier;
     m : entier;
     s : entier;
  fin;
 vecteurs_it:tableau[0..n-1] de MEP;
 PILESYSTEME : PILE de MEP;
 Savevecteur: MEP; /* Structure dans laquelle on sauvegarde le vecteur d'IT du système
                                correspondant à l'interruption compteur nul */
 MEP_cptnul: MEP; /* mot d'état utilisateur de l'it compteur nul */
 Savemep, M: MEP;
 Nbr_sonneries : entier; /* compteur de sonneries */
 Const  Nbr_sonneries_total = 14 ;  /* le nombre total de sonneries à déclencher par jour */
 Heure : Hms ;
  /* Initialisation du tableau contenant les valeurs du compteur du TIMER : temps des
sonneries en secondes */
 Valeur_Cptimer : tableau[0 .. Nbr_sonneries_total -1] d'entier
    ={  (08*60+30-1)*60, /* 8h30: début de la 1ière séance */
          (09*60+30)*60, /* 9h30: fin de la 1ière séance */
          (09*60+35-1)*60, /* 9h35: début de la 2ème séance */
          (10*60+35)*60, /* 10h35: fin de la 2ème séance */
          (10*60+40-1)*60, /* 10h40: début de la 3ème séance */
          (11*60+40)*60, /* 11h40: fin de la 3ème séance */
          (13*60+00-1)*60, /* 13h00: début de la 4ème séance */
         (14*60+00)*60, /* 14h00: fin de la 4ème séance */
```

(14*60+05-1)*60, /* 14h05: début de la 5ème séance */

(15*60+05)*60, /* 14h05: fin de la 5ème séance */

(15*60+10-1)*60, /* 15h10: début de la 6ème séance */

(16*60+10)*60, /* 16h10: fin de la 6ème séance */

(16*60+15-1)*60, /* 16h15: début de la 7ème séance */

(17*60+15)*60} ; /* 17h15: fin de la 7ème séance */

}

Procédure Initialisation() ; /* Procédure exécutée une seule fois au démarrage du système */

T,k :entier ;

Début

Nbr_sonneries:=0;

/* Initialisation du vecteur d'it utilisateur de l'interruption compteur nul CPTNUL*/

M->mode := utilisateur;

M->masque_timer := masqué; /* on peut aussi laisser l'it du timer démasquée pendant le traitement de la procédure CPTNUL cela ne poserai aucun problème mais ça sera inutile et ralentira l'exécution de la procédure inutilement */

M->masque_cptnul := masqué;

M->masque_dérout := autorisé; /* il faut autoriser le traitement des éventuelles erreurs qui peuvent survenir pendant l'exécution de la procédure CPTNUL */

M->masque_autres_it := masqué;

M->compteur_ordinal := CPTNUL; /*CPTNUL est le nom de la Procédure utilisateur de traitement de l'it cptnul */

/* Sauvegarde du vecteur d'interruption système avant de modifier l'entrée */

lire_vecteur(21, savevecteur) ;

/* ranger le mot d'état M dans le vecteur d'interruption N°21 */

modifier_vecteur(21, M) ;

/* Initialisation du compteur du timer : calculer la valeur initiale de variable CPTIMER */

Heure=temps() ; /* Récupérer l'heure de démarrage du système */

T :=(Heure->h*60+Heure->m)*60+Heure->s ; /* convertir l'heure de démarrage du système en secondes*/

```
k :=0 ;
Tantque ((T > Valeur_Cptimer[k]) et (k < Nbr_sonneries_total )) faire
    k :=k+1
finfaire;

Si (k < Nbr_sonneries_total ) alors /* cas où l'heure de démarrage du système <= 17h15
*/
   Si T = Valeur_Cptimer[k]  alors  /* le chargement du système coïncide avec l'heure
                                       de la sonnerie*/
        Sonner() ;
        Nbr_sonneries : = (k+1) Mod Nbr_sonneries_total;  /* mettre à jour le pointeur
                                                            de la prochaine sonnerie */
    Sinon
        Nbr_sonneries : = k ; /* mettre à jour le pointeur de la prochaine sonnerie */
   Finsi ;
/* initialisation du compteur du Timer CPTIMER*/
   Si (Nbr_sonneries < > 0 ) alors
      CPTIMER:=Valeur_Cptimer[Nbr_sonneries] - T; /* système chargé avant 17h15 */
   Sinon
        Si (k<>0) alors   /* système chargé à 17h15 */
          CPTIMER:=(24*3600 - T) + Valeur_cpt[0] ;
        Sinon /* cas où le système démarre après minuit et avant 8h29 par exemple le
                 système est démarrer à 1h00 du matin */
          CPTIMER:= Valeur_Cptimer[Nbr_sonneries] – T;
       Fsi
  Sinon /* système chargé après 17h15 et avant minuit */
     Nbr_sonneries:=0;
     CPTIMER:=(24*3600 - T) + Valeur_cpt[0] ;
finsi ;
Fin de la procédure initialisation;
```

```
Procédure CPTNUL; /* procédure de traitement utilisateur de l'interruption compteur nul */
  Début
     /* procédure exécutée à chaque passage à zéro de  la variable CPTIMER*/
      Sauvegarder(contexte) ;   /* sauvegarde du contexte du processus interrompu */
      Sonner() ;      /* Déclencher la sonnerie */
      Nbr_sonneries := Nbr_sonneries +1;  /* incrémenter le nombre de sonneries : le pointeur
    Nbr_sonnerie pointera l'heure de la prochaine sonnerie dans le tableau Valeur_Cptimer  */
  /* initialiser le compteur du TIMER : calculer la valeur initiale de la variable CPTIMER */
    Si ( Nbr_sonneries < Nbr_sonneries_total )   alors
          CPTIMER:=Valeur_Cptimer[Nbr_sonneries] - Valeur_Cptimer[Nbr_sonneries - 1];
     Sinon   /* fin de journée */
        Nbr_sonneries :=0 ;  /* Réinitialiser le pointeur de sonneries */
        CPTIMER:=(24*3600 - Valeur_Cptimer[Nbr_sonneries_total - 1]) + Valeur_cpt[0];
              /* 6h45+8h29 ➔  durée entre 17h15 et 8h29 */
      Finsi ;
      dépiler(PILESYSTEME, Savemep); /* déplier le MEP du programme interrompu*/
      Restaurer(contexte) ;   /* restaurer le contexte  */
      Charger_mep(savemep); /* Charger le MEP du processeur avec le MEP du programme
                interrompu contenu dans savemep  pour reprendre l'exécution du programme */
      Fin de la procédure CPTNUL;
```

=-

```
Programme principal ;
    Début
       Initialisation();    /* Appeler la procédure d'initialisation */
       Tantque (vrai) faire    /*  Boucle infinie  */
            Attendre() ;        /* Mettre en attente le programme pendant une durée de temps,
        FinTq                                  utiliser la fonction sleep( ) du C par exemple  */
     Fin du programme principal;
```

<u>Remarque</u> : Le programme principal doit rester en exécution de manière permanente afin de maintenir la procédure CPTNUL de traitement de l'it cptnul en mémoire centrale car si le programme principal s'arrête, il sera supprimé de la mémoire par le système, et par conséquent la procédure CPTNUL sera supprimée également de la mémoire, ce qui va entraîner l'arrêt du système de sonneries.

==========

Exercice3

a) Masques **RM** du programme P et les différentes routines de traitement des interruptions :

Il y a deux solutions possibles :

1ière solution : **Masqué le niveau i** et tous les niveaux inférieurs au niveau i (un programme de niveau i ne sera pas interrompu pendant l'exécution par un programme de même niveau ou de niveau inférieur).

2ième solution : **Démaqué le niveau i** et masqué tous les niveaux inférieurs au niveau i (un programme de niveau i peut être interrompu pendant l'exécution par un programme de même niveau mais ne sera jamais interrompu par un programme de niveau inférieur).

	RM en binaire	RM en binaire
P	11**0**1**0**110	11010111
N1	1111110**0**	1111111**0**
N2	11111**0**00	11111**1**00
N3	1111**0**000	1111**1**000
N4	111**0**0000	111**1**0000
N5	11**0**00000	11**1**00000
N6	1**0**000000	1**1**000000
N7	***0***0000000	***0***0000000

b) Diagramme:

Remarque : Dans ce schéma le digramme est décalé de 10unités (le tracé commence à 0 et non pas a 10)