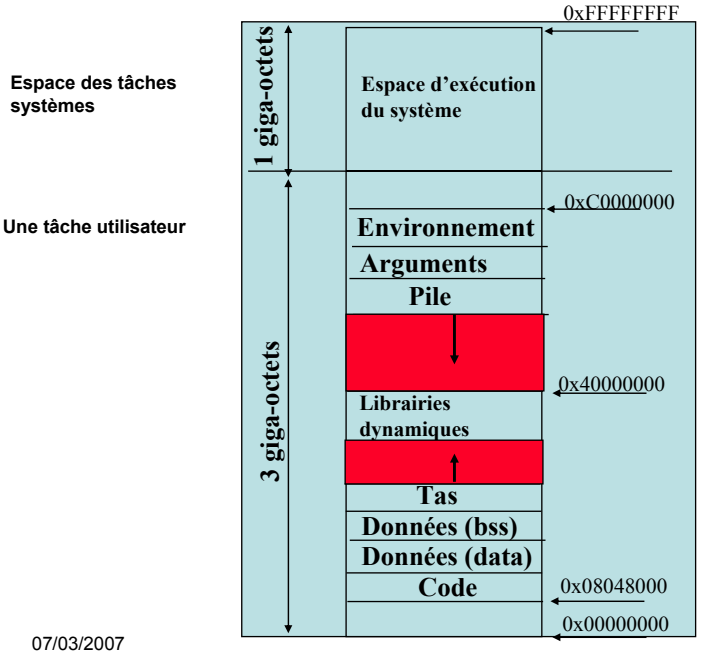
Les tâches

Une tâche est un programme « en cours » d'exécution

- Automate des états (en cours d'exécution)
- Description d'une tâche
- Contrôle d'exécution des tâches

Couches	Nom	Objets impliqués	Exemple d'opérations
13	Shell	Env utilisateur	Langage shell
12	Programme utilisateur	Processus	Fork, quit, kill, suspend, resume
11	Répertoire	Répertoire	Mkdir, remove, attach, detach
10	Périphériques	Imprimante, écran, clavier	Open, close, read, write
9	Système de Gestion de fichiers	Fichiers	Create, destroy, open, close, read, write
8	Communications	Pipes	Create, destroy, open, close, read, write
7	Mémoire virtuelle	Segments, pages	Read, Write, Fetch
6	Mémoire de masse	Blocs de données, canaux d'E/S	Read, Write, Allocate, free
5	Processus, tâches et flots	Tâches, sémaphores, liste de tâches	Suspend, Resume, Wait, Signal
4	Interruptions	Invocation pilote (handler)	Invoke, mask, unmask, retry
3	Procédures	Appel et retour, contexte	Call, return
2	Jeu d'instructions	Pile d'évaluation, gestion des UF, microprogrammes	Load, Store, add, addf, branch, jsr, rts, rti
1	Carte et circuits	Registres, UF, bus …	Clear, activate, transfert …

UNIX



Création de Tâche

- Soumission d'un travail en batch
- Login d'un utilisateur
- Par le SE pour fournir un service
  - Mémoire,
  - Réseau,
  - Périphérique..
- Engendrée par une tâche existante
  - Fork ()
  - Notion de tâche mère et tâche fille
  - Coopération entre tâche

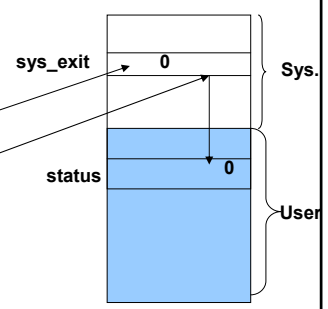
Actions sur création de tâche

```
pid = fork() ;
If (pid < 0 ) { /* le fork a échoué */}
  } else if (pid > 0) {/* code du père */
        } else {  /* code du fils */}
```

- Donner un identificateur unique
- Allouer un espace mémoire
- Initialiser le PCB (Process Control Block)
- Positionner des liens appropriés
  - Ajouter à la liste chaînée des tâches prêtes à être ordonnancées
- Créer un fichier de log

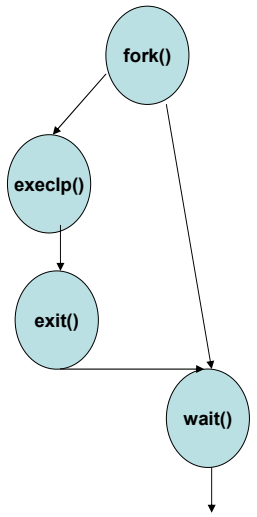
Le fork

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#define N 50

int main(void)
{
      pid_t tab_pid[N] ; /* tableau des pid des fils */
      int status, i ;
      for (i = 0 ; i < N; i++) {
            if ( ! (tab_pid[i] = fork() ) ) { /* le code des fils */
                  printf (''hello ; je suis %d\n '', getpid() ) ;
                  sleep(1) ;        /* attente de 1s. */
                  exit(0) ; /* les fils terminent */
            }
      }
/* le père attend la terminaison */
      for (i=0 ; i < N ; i++ ) {
            waitpid (tab_pid[i], &status, 0) ;
      }
return 0 ;
}
```



Le fork

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main(void)
{
      pid_t tab_pid ; /* identificateur de la tâche*/
            pid = fork() ;
            if (pid<0) { /* erreur */
            fprintf(stderr, '' fork en erreur'');
            exit (-1) ;
      }
      else if (pid == 0) {/* code du fils */
            execlp (''/bin/ls '', ''ls'', NULL) ;
      }
      else { /* le code du père */
      /* il attend que le fils termine */
      wait (NULL) ;
      printf( '' c'est fini'') ;
      exit(0) ;
      }
}
```



Terminaison de tâche

- Batch émet une instruction Halt
- L'utilisateur fait un log out
- Une application est quittée (exit)
- Détection d'erreur ou de faute

Causes

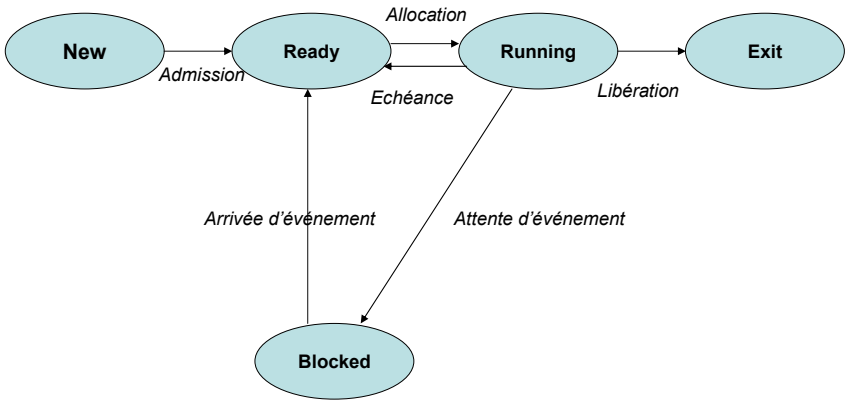
- Nominal
- Limite de temps
- Mémoire non disponible
- Violation d'adressage
- Violation de droit (protection de fichiers)

Causes (suite)

- Erreur arithmétique, parité mémoire…
- Watch dog (attente limitée sur un événement)
- Erreur d'E/S
- Instruction invalide
- Tentative d'exécution d'instruction privilégiée
- Utilisation incorrecte de données
- Détection d'un deadlock
- La tâche mère termine ce qui entraîne la fin de la tâche fille
- Explicite par une tâche mère

Un modèle de tâche à 5 états

- Running
- Ready
- Blocked
- New
  - Créée mais non chargée en mémoire
- Exit
  - Retirée de l'ensemble des tâches exécutables

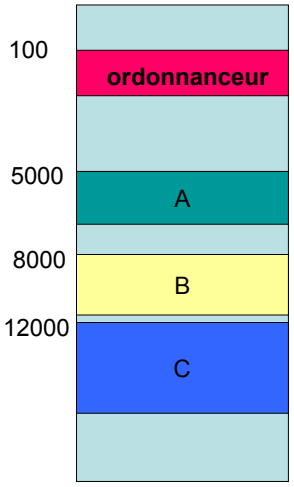


Mode d'exécution

- Mode utilisateur
  - Pour les programmes utilisateur
  - Privilège minimum
- Mode noyau
  - Privilège maximum
  - Contrôle total du processeur, des registres et de la mémoire
  - Réservé à la programmation système

Exemple d'un système multitâche

Traces d'exécution



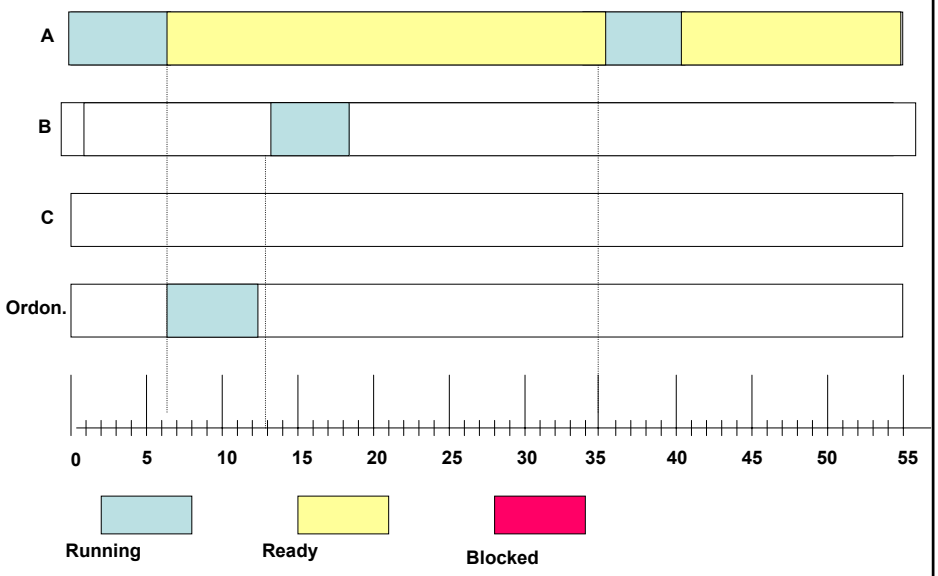
Tâche A	Tâche B	Tâche C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5010		12200
5011		12201
5012		12202
5013		12204
5014		12004
5015		12005
5016		12006
5017		12007
		12008

CO **8001**

CYCLES	ADRESSES		CYCLES	ADRESSES
1	5000		27	12200
2	5001	***Ordonnanceur***	28	12201
3	5002		29	100
4	5003		30	101
5	5010		31	102
6	5011		32	103
7	100	Time out	33	104
8	101		34	105
9	102		35	5012
10	103		36	5013
11	104		37	5014
12	105		38	5015
13	8000		39	5016
14	8001		40	5017
15	8002		41	100
16	8003		42	101
17	100	Demande E/S	43	102
18	101		44	103
19	102		45	104
20	103		46	105
21	104		47	12202
22	105		48	12004
23	12000		49	12005
24	12001		50	12006
25	12002		51	12007
26	12003		52	12008

A quel cycle se situe la machine représentée par l'image précédente ?

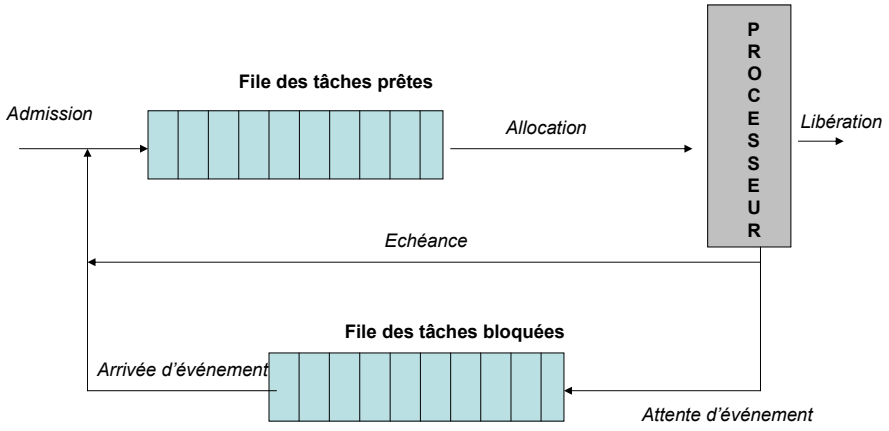
Cartographiez les états des tâches A, B et C de l'exemple précédent



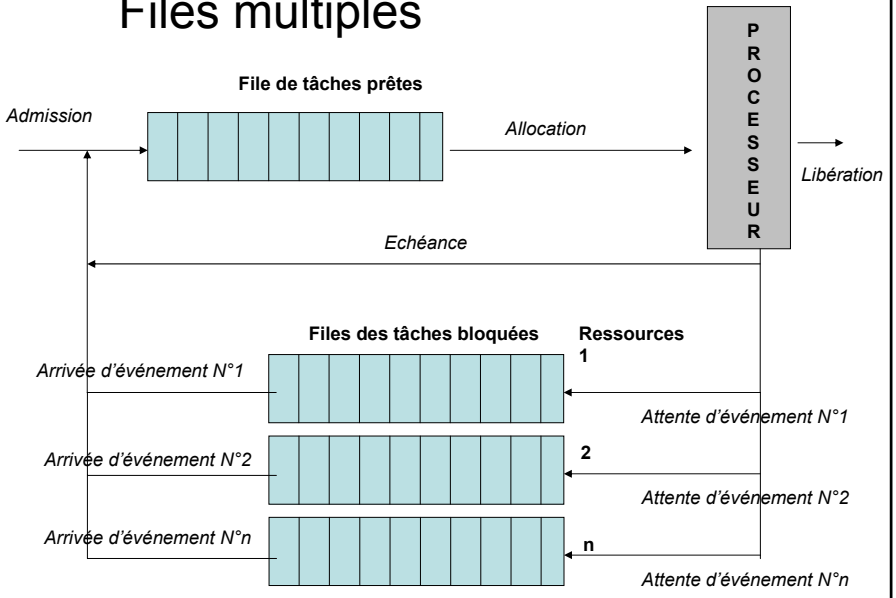
Ordonnancement

- Plusieurs niveaux :
  - Long_term : ordonnancement des tâches non chargées
  - Short term : ordonnancement des tâches présentes dans l'UC (processeur et mémoire centrale).
- Plusieurs types :
  - Préemptif : une tâche allouée peut-être suspendue selon des conditions extérieures à son exécution
  - Non préemptif : une tâche allouée s'exécute jusqu'à ce qu'elle termine ou qu'elle demande une E/S.

Ordonnancement par le modèle des files d'attente



Files multiples



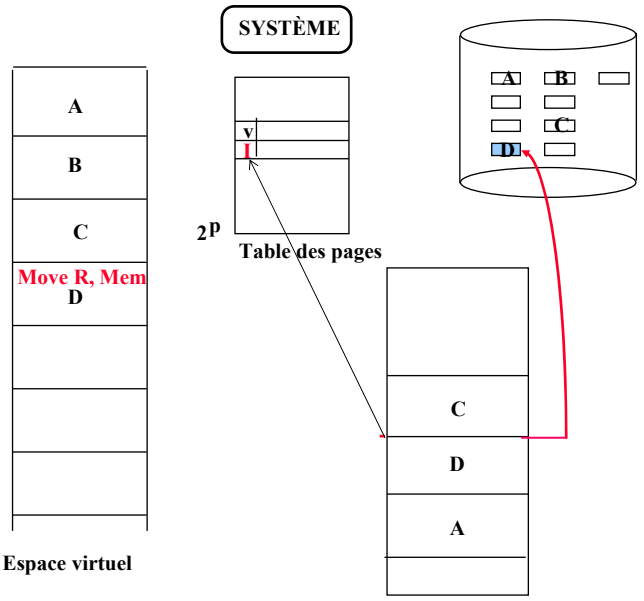
Tâches suspendues

- Le swapping (« va_et_vient »)
  - L'UC étant plus rapide que les E/S, toutes les tâches sont en attente sur une E/S
  - L'UC est donc oisive la plupart du temps
  - La mémoire doit être étendue pour que le système accepte davantage de tâches :

Swapping

Tâches suspendues (2)

- Swapping
  - Transfert de tout ou partie d'une tâche de la mémoire sur le disque (swap out)
  - Rechargement pour activation (swap in)
  - Changement de l'état *Blocked* à *Suspended*
  - Création de la file des tâches suspendues : tâches qui ont provisoirement été « jetées » de la mémoire centrale

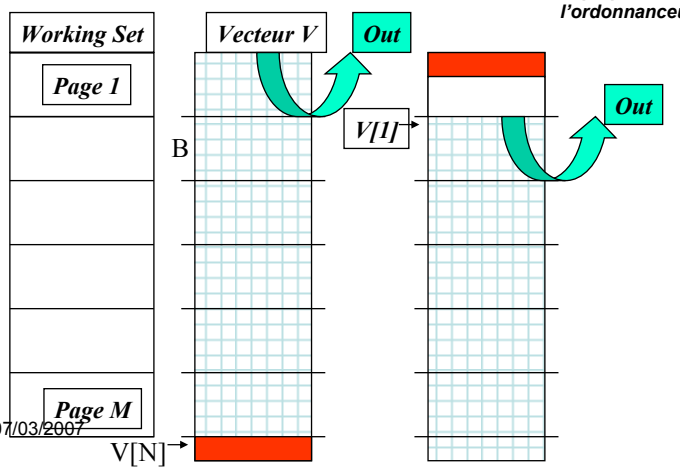


Un exemple : gestion FIFO des pages mémoire

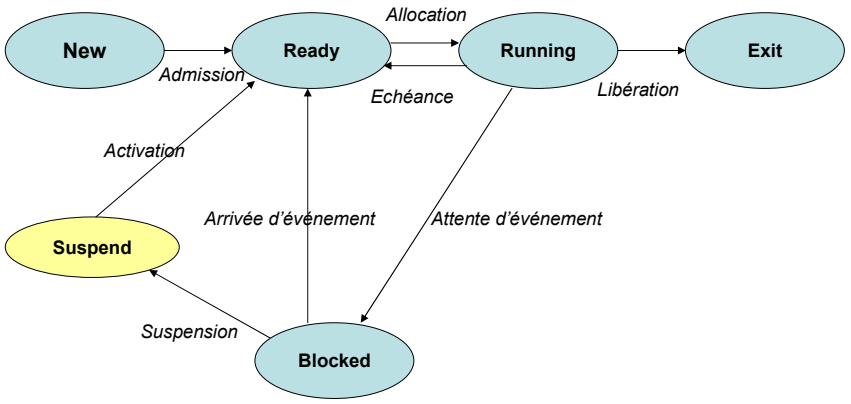
```
Real V[N] ;
        for j = 1 to P
                for i =1 to N
                V[i] = f(i,j) ;
```

*Soit N = M*B + 1*
V[1..N-1] en mémoire

Expliquez ce qui se passe dans l'ordonnanceur du système

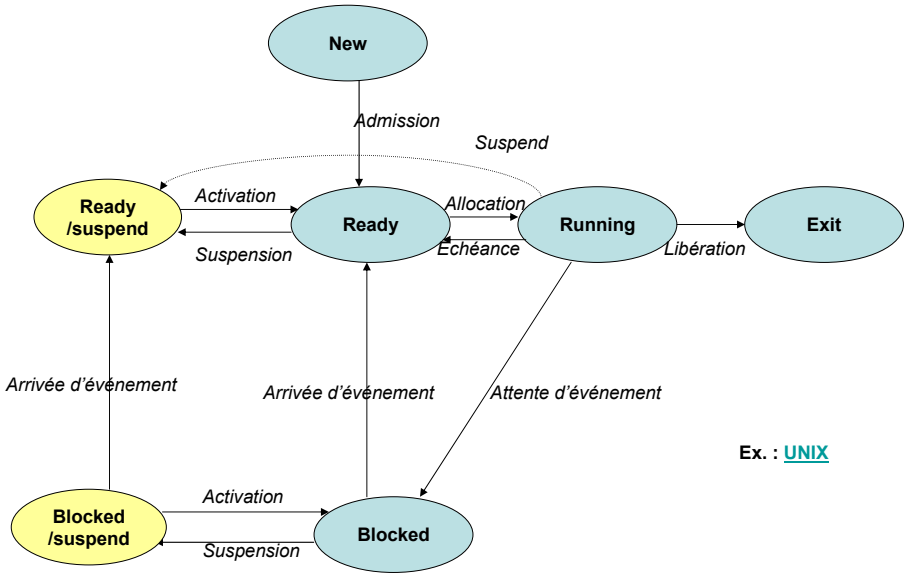


Nouveau modèle à 6 états



Causes d'une suspension

- Swapping
  - L'O/S doit libérer de la mémoire pour activer une tâche prête
- Requête interactive
  - L'utilisateur demande explicitement la suspension (debug)
- Échéance
  - La tâche a atteint une échéance d'exécution fixée par l'OS ou par l'utilisateur
  - La tâche doit s'exécuter périodiquement
- Requête de la tâche mère
  - Synchronisation explicite demandée par la mère
- Divers
  - Détection d'erreur, arrêt d'une tâche moins prioritaire



Rôle du système d'exploitation

- Contrôler les événements subis par l'UC
- Ordonnancer et activer les tâches pour leur exécution par l'UC
- Allouer des ressources aux tâches
- Répondre aux requêtes soumises par l'utilisateur
- Gérer l'utilisation des ressources par les tâches

Comment ?

Structures de données du SE : tables et files

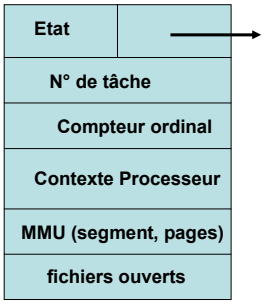
- Tables des tâches (Process Control Block)
  - Contient les attributs d'une tâche
- Tables mémoire
  - Allocation MC aux tâches
  - Allocation mémoire secondaire
  - Attributs de protection pour accéder à des régions partagées
  - Segments et pages
- Tables des E/S
  - Un contrôleur est disponible ou en cours d'utilisation
  - Une opération est : en cours, initialisée, terminée
  - Adresse mémoire source ou destination

- Tables de gestion des fichiers
  - Association adresse logique - nom de fichier
  - Adressage en mémoire secondaire
  - Etat courant (ouvert, fermé, en L/E)
  - Attributs (protection, dates…)

Ces tables sont gérées en association avec le SGF

Contenu du PCB : Identification et état

- Identification
  - De la tâche, de son créateur, de l'utilisateur.
- Information de contexte
  - CO, CC, état, TLB, MMU, pointeurs de pile




```
struct task_struct
        { pid_t pid ;           /* identificateur de tâche*/
        pid_t ppid;             /* identificateur du parent */
        long state ;            /* état de la tâche */
        struct thred_info *thread_info /* information d'état du proc */
        unsigned int time_slice ; /* information pour l'ordonnanceur*/
        struct files_struct  *files ; /* liste des fichiers ouverts */
        struct mm_struct *mm ;  /* espace d'adressage de cette tâche */
        .
        .
        .
        }
```


Contenu du PCB : contrôle de la tâche

- Ordonnancement et information d'état
  - Les informations nécessaires à l'OS pour réaliser l'ordonnancement :
    - Etat de la tâche (prête, suspendue, en exécution….)
    - Priorité
    - Paramètres de l'algorithme d'ordonnancement : temps d'attente de la tâche, temps d'exécution lors de la dernière activation…
    - Evénement : identité de l'événement sur lequel il doit être réveillé.
- Les structures de données de chaînage des tâches
    - File, anneau, liste…

- Communication inter-tâches
  - Flags, signaux, messages échangés lors de la communication entre deux tâches
- Droits d'accès
  - Adresses mémoires accessibles
  - Type d'instructions exécutables
- La gestion mémoire
  - Tables des segments, table des pages
- Propriétés et utilisation des ressources
  - Liste des fichiers ouverts
  - Histoire d'utilisation du processeur et des autres ressources

Fonctions basiques d'un noyau de SE

- Gestion des tâches
  - Création
  - Ordonnancement et allocation
  - Commutation
  - Synchronisation et communication
  - Gestion des PCB
- Gestion de la mémoire
  - Allocation d'espace
  - Swapping
  - Gestion des pages et des segments

- Gestion des E/S
  - Gestion des buffers
  - Allocation des canaux d'E/S et des périphériques aux tâches
- Fonction de services
  - Gestion des interruptions
  - Comptabilité
  - Mesures
  - Surveillance …

Pid = fork()	Crée un processus fils : Retour 0 pour le fils, PID du fils retourné vers le père
E = waitpid(pid, &etat,opts)	Attends un fils pour terminer
E= execve(nom, argv, envp)	Remplace l'image exécutable d'une tâche
Exit (etat)	Termine l'exécution et renvoie l'état
E = sigaction(sig,&act,&oact)	Spécifie l'action à entreprendre pour un signal
E = kill(pid, sig)	Envoie un signal à un processus
Pause()	Suspend l'appelant jusqu'au prochain signal

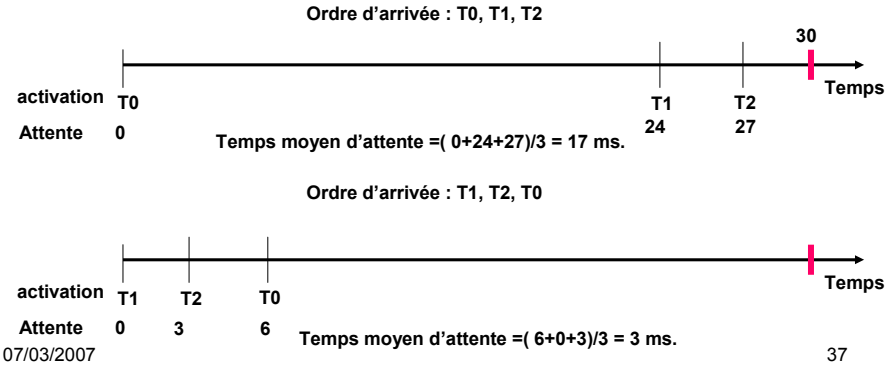
Politique d'ordonnancement des tâches

- L'ordonnanceur doit assurer :
  - L'équité, /*chacun à droit à la ressource*/
  - L'efficacité, /* le processeur doit être saturé*/
  - Le temps de réponse, /* interactivité*/
  - Le temps d'exécution, /* au max de la perf pour chacun*/
  - Le rendement. /* favoriser le débit d'exécution de prgmes*/
- Conclusion :
  - « à l'impossible nul n'est tenu ! »
  - Le temps est géré par un « timer » interrogé périodiquement par le SE pour gérer les tâches.

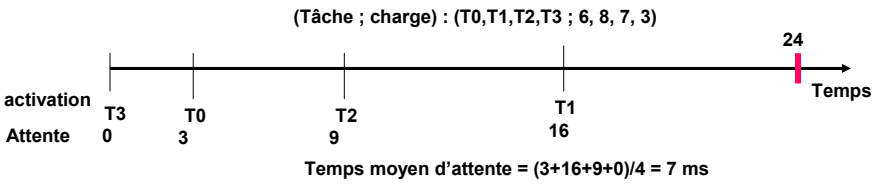
Type d'ordonnancement

- Les tâches sollicitent essentiellement soit :
  - Les ressources dans l'UC : tâches de calcul
  - Les E/S : tâches interactives
- Ordonnancement préemptif
  - Une tâche en cours peut être suspendue (arrivée à échéance, trop de mémoire consommée …)
- Ordonnancement non préemptif
  - Une tâche s'exécute jusqu'à sa terminaison et peut bloquer toutes les autres (généralement réservé à des tâches déterminantes pour le système)

- Algo1 : premier arrivé, premier servi
  - Réalisé par une file FIFO
  - Engendre un fort déséquilibre du temps moyen d'attente de toutes les tâches selon l'ordre d'arrivée
  - Ex soit : (T0, T1, T2) = (24, 3, 3) temps d'occupation en ms de l'UC.



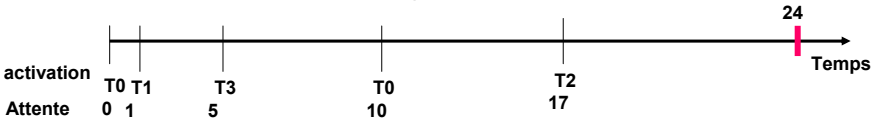
- Algo2 : le plus court suivant non préemptif
  - Ordonne la file selon la charge induite
  - Nécessite de connaître ou prédire la charge d'exécution



- Algo3 : le plus court suivant préemptif
  - Ordonne la file selon la charge induite
  - Préempte la tâche en cours si la charge induite par le suivant est plus faible

(Tâche ; instant d'arrivée) : (T0,T1,T2,T3 ; 0, 1, 2, 3)

(Tâche ; charge) : (T0,T1,T2,T3 ; 8, 4, 9, 5)



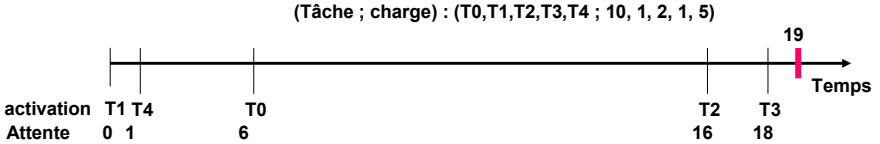
Temps moyen d'attente = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 6,5 ms

Exercice : calculer le temps moyen d'attente avec un algorithme non préemptif.

- Algo4 : le plus prioritaire
  - Chaque tâche a une priorité et les tâches de plus grandes priorités (valeurs décroissantes) préemptent les autres. (algo préemptif ou non préemptif)
  - La priorité d'une tâche diminue au cours du temps pour éviter un possible blocage infini.

(Tâche ; priorité) : (T0,T1,T2,T3,T4 ; 3, 1, 4, 5, 2)

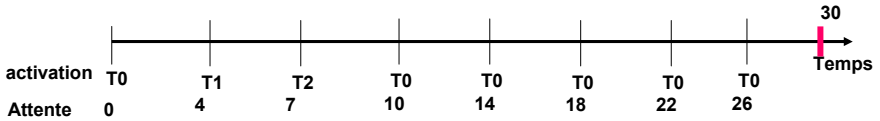
(Tâche ; charge) : (T0,T1,T2,T3,T4 ; 10, 1, 2, 1, 5)



Temps moyen d'attente = [0+1+6+16+18]/4 = 8,2 ms

- Algo5 : Le tourniquet
  - Dédié à des systèmes temps partagé
  - Liste (FIFO) des processus prêts avec leur quantum de temps : suspendue à l'expiration ou sur une E/S.
  - Difficulté de gérer la valeur du quantum

(Tâche ; charge) : (T0, T1, T2 ; 24, 3, 3) et quantum de 4 ms.

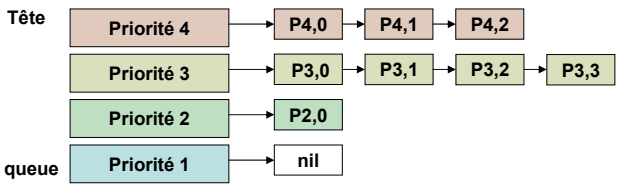


Temps moyen d'attente =(17)/3 = 5,66 ms.

Autre possibilité : Tourniquet des tâches exécutables par niveau de priorité

- Ordonnancement multi-niveau par priorité
  - La file des tâches prêtes est partitionnée en plusieurs files selon la priorité associée à chaque entrée.

File des tâches prêtes



Commutation de tâches

- Interruptions
  - Horloge (time out)
  - E/S
  - Défaut mémoire
  - Erreur (division par 0, parité…)
- Appel superviseur
  - Instructions SVC

Coût de commutation de tâches : une dizaine de µs jusqu'à plusieurs mms si commutation sur disque.

Changement d'état d'une tâche

- Sauvegarde du contexte processeur
- Mise à jour du PCB
- Transférer le PCB dans la file appropriée des tâches
- Sélection d'une autre tâche pour exécution
- Mise à jour du PCB de la nouvelle tâche
- Mise à jour des structures de données de la MMU
- Restore le contexte de la tâche élue

Exemple : Windows XP

- Algo à priorité, ordonnancement préemptif
- L'ordonnancement est réalisé par module du noyau, le « dispatcher ».
- Le « dispatcher » utilise une file à 32 niveaux de priorité, classés en deux groupes :
  - 1 à 15 groupe standard variable (0, réservé à la gestion mémoire)
  - 16 à 31 groupe temps-réel

Les priorités de Windows XP

Classe prioritaire — Classe variable

Priorité dans la classe	Temps-réel	Haut	+ Normal	Normal	- Normal	oisif
Temps -critique	31	15	15	15	15	15
haute	26	15	12	10	8	6
Au dessus de normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
En dessous de normal	23	12	9	7	5	3
basse	22	11	8	6	4	2
inactif	16	1	1	1	1	1

Priorité de base — Classe par défaut