ORDONNANCEMENT
DE L'UNITE DE TRAITEMENT

1. Objectifs

L'intérêt de la multiprogrammation est non seulement d'obtenir le meilleur taux d'utilisation du processeur mais aussi le meilleur temps de réponse du système. Le rôle de l'ordonnancement est d'optimiser un certain nombre de critères:

- taux d'utilisation de l'unité centrale
- nombre de programmes utilisateurs traités en moyenne par unité de temps
- temps de traitement moyen pour chaque tâche
- temps de traitement total
- temps de réponse maximum (temps entre la soumission d'une tâche et son accomplissement)

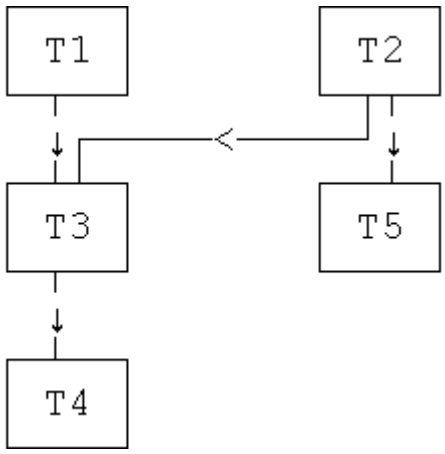
Il s'agit ici d'utiliser l'unité centrale de manière optimale en garantissant le meilleur temps de réponse pour l'ensemble des tâches ainsi qu'un partage équitable.

L'unité centrale peut exécuter les tâches dans un certain ordre connu à priori dans le cas de tâches synchronisées ou dans un ordre à établir dans le cas de tâches indépendantes. Toutes les tâches à effectuer ne sont pas forcément connues en même temps. Chaque tâche T_i est connue du système à un temps t_i. Remarquons que la durée Di d'une tâche Ti n'est pas connue exactement à priori. Lorsqu'on parle de durée d'une tâche, il s'agit d'une durée prévisionnelle, supposée ou estimée.

2. Assignations et diagrammes de Gantt

A un système de tâches correspond une description de l'exécution des tâches par le (ou les) processeurs(s) appelée **assignation**. Une assignation respecte le graphe de précédence si il en existe un. Le **diagramme de Gantt** est utilisé pour représenter une assignation. A partir du tableau des tâches et du graphe de précédence ci-dessous

	Di	ti
T1	1	0
T2	2	0
T3	1	0
T4	1	0
T5	2	0

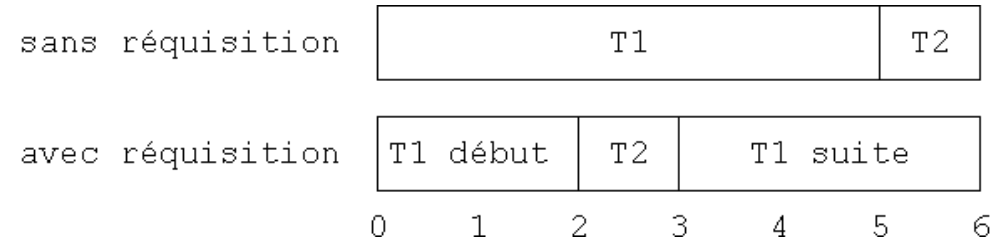


Nous avons le diagramme de Gantt suivant pour 2 processeurs:

Différents algorithmes d'ordonnancement existent. Les algorithmes avec **réquisition** permettent l'interruption d'une tâche pendant son exécution, ceci implique le découpage des tâches en tranche.

Nous nous intéressons au temps moyen d'une tâche. Soient deux tâches T1 et T2 exécutées par un seul processeur, les diagrammes de Gantt sans réquisition et avec réquisition sont les suivants:

	Di	ti
T1	5	0
T2	1	2



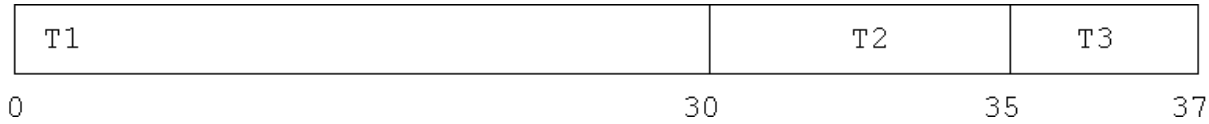
Dans le diagramme sans réquisition, le temps de réponse moyen est [5+(6-2)] / 2 soit **4,5**. Dans le diagramme avec réquisition, il devient [6 + (3-2)] /2 soit **3,5**. La réquisition permet d'exécuter voire de terminer les petites tâches qui surviennent pendant l'exécution de la tâche courante, ceci est particulièrement intéressant lorsque la tâche en cours d'exécution est longue.

3. Algorithmes sans réquisition

3.1. ORDONNANCEMENT DANS L'ORDRE D'ARRIVEE (FIFO)

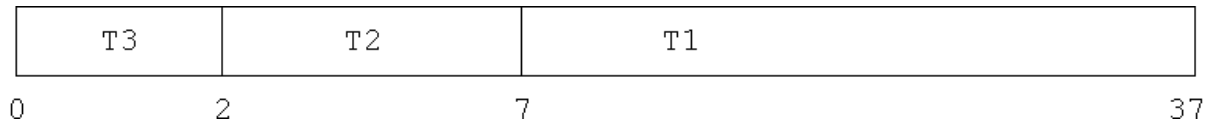
A un instant donné, la tâche à activer est la tâche la plus vieille dans la file d'attente. Cet algorithme permet de prendre en compte naturellement toutes les tâches à activer les unes après les autres.

	Di	ti
T1	30	0
T2	5	ε
T3	2	2ε



Temps moyen = [30 + 35 + 37] /3 = **34**

Ici les 3 tâches arrivent presque en même temps et dans un ordre défavorable. La tâche la plus longue est exécutée en premier et les 2 autres tâches attendent longtemps. Il est évident que l'ordre suivant aurait permis de ramener le temps moyen à [2 + 7 + 37] / 3 soit **15,3**:

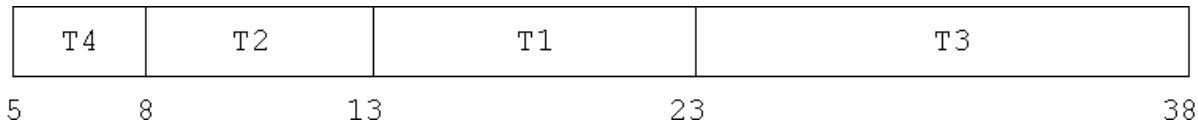


3.2. PLUS COURT TEMPS D'EXECUTION (PCTE)

Cet algorithme permet d'éviter de démarrer la tâche la plus longue en premier. Les tâches sont ordonnées par temps d'exécution croissant: la tâche ayant le plus court d'exécution d'abord.

	Di	ti
T1	10	0
T2	5	2
T3	15	3
T4	3	4

A l'instant t=5, les tâches T1 à T4 sont en attente. L'assignation produite par l'algorithme PCTE est

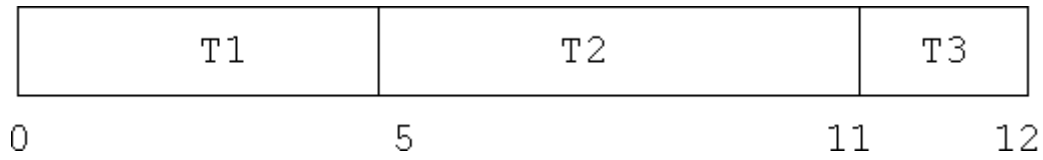


Le temps moyen est [(8-4) + (13-2) + (23-0) + (38-3)] / 4 soit **18,25**. L'instant ti au cours duquel le système prend connaissance d'une tâche intervient dans le calcul du temps de réponse moyen.

Lorsque toutes les tâches sont connues, cet algorithme donne le meilleur temps moyen. Cette situation n'est toutefois pas réaliste car des tâches à exécuter peuvent se présenter au cours de l'assignation. Exemple :

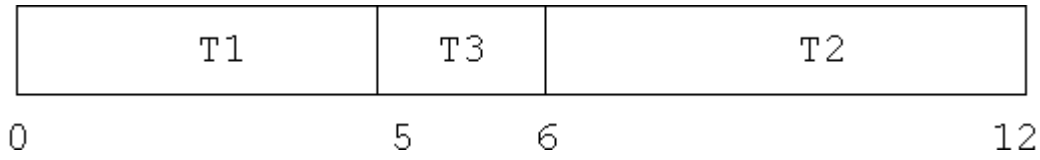
	Di	ti
T1	5	0
T2	6	0
T3	1	5

A l'instant t=0, seules les tâches T1 et T2 sont connues. L'assignation produite par l'algorithme PCTE (t=0 puis t=11) donne :



Temps moyen = [(5-0) +(11-0) +(12-5)] / 3 = **7,66.**

Une réévaluation de l'ordonnancement à la fin d'exécution de la tâche courante (T1 au temps t=5) donnerait :



Temps moyen = [5 +1 +12] / 3 = **6**

Cet algorithme n'est pas utilisé dans la pratique. En effet si des tâches de durées prévisionnelles inférieures à 6 unités se présentent régulièrement, elles empêcheront la tâche T2 de s'exécuter en passant devant elle. La tâche T2 risque de ne jamais passer.

Dans l'algorithme FIFO les tâches les plus longues pénalisent le système quand elles passent en premier. Dans l'algorithme PCTE, les tâches les plus brèves sont favorisées et les tâches les plus longues ont un temps de réponse très mauvais.

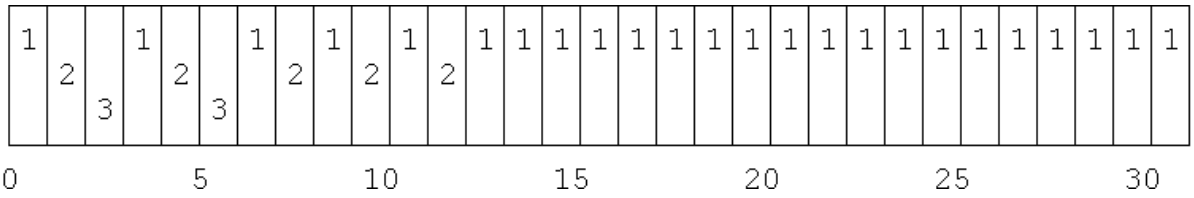
4. Algorithmes avec réquisition

Afin de rendre plus équitable le partage du temps CPU, on exécute les tâches à tour de rôle. Le laps de temps minimal réservé pour l'exécution partielle d'une tâche s'appelle le **quantum**. Il s'agit d'une unité de temps assez petite pour permettre le découpage des tâches. Sa durée est plus grande que le temps de commutation des tâches. Une nouvelle tâche entre en exécution lorsque la tâche en cours a consommé son quantum de temps (interruption) ou bien se termine (fin de tâche).

4.1. TOURNIQUET (ROUND ROBIN)

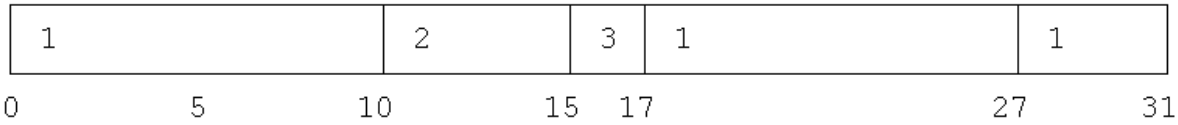
La technique du tourniquet établit une liste des tâches à exécuter puis exécute à tour de rôle partiellement les tâches les unes après les autres selon cet ordre. Exemple :

	Di	ti
T1	24	0
T2	5	ε
T3	2	2ε



Temps moyen est [31+12+6] /3 = **16,33**

Un quantum petit donne un meilleur temps de réponse. Il ne doit pas être trop petit, car le temps de commutation des tâches n'est plus négligeable et pénalisera fortement le fonctionnement du système. Avec un quantum élevé (par exemple 10 unités), le temps de réponse est moins bon: [31+15+17] /3 = **21**.

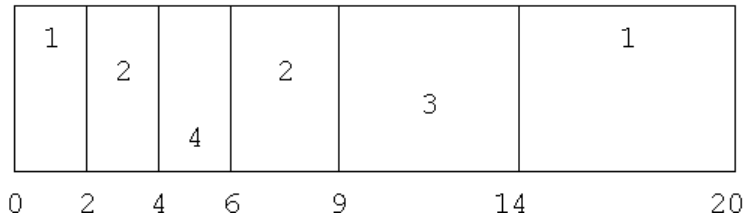


Cet algorithme très équitable permet à toutes les tâches de se terminer en un temps fini.

4.2. PLUS COURT TEMPS D'EXECUTION RESTANT (PCTER)

Cet algorithme généralise l'algorithme PCTE. A la fin de chaque quantum (supposé égale à 1), la tâche ayant le plus petit temps d'exécution restant est candidate au CPU. Exemple:

	Di	ti
T1	8	0
T2	5	2
T3	5	3
T4	2	4



La même remarque que pour l'algorithme PCTE s'applique: les tâches les plus longues sont défavorisées et risquent de ne jamais se terminer.

4.3. ALGORITHMES AVEC PRIORITE

Un système de priorité différent que ceux présentés ci-dessus (FIFO, tourniquet, PCTER, etc) peut être mis en place en fonction de différents critères d'éligibilité au CPU: occupation mémoire centrale, temps CPU consommé, temps horloge écoulé, temps d'exécution restant, etc. Les priorités sont dynamiques c'est à dire recalculées à la fin de chaque quantum.

Si les priorités étaient statiques (calculées une fois pour toutes), alors pareillement aux algorithmes PCTE ou PCTER, les tâches les moins prioritaires vont laisser passer devant elles les tâches les plus prioritaires et risquent de ne jamais se terminer.

5. Caractéristiques et classification d'un travail

Un découpage des tâches dans différentes classes de travaux permet de garantir l'exécution d'ensembles de tâches de nature différentes: temps demandé élevé ou non, grande occupation mémoire ou non, etc.

Différents critères permettent de séparer les travaux en classes d'exploitation:

- nature de l'application (mise au point, codes numériques, travaux de gestion, temps réel, etc)
- taux prévisionnel d'utilisation des ressources (temps CPU, mémoire centrale, périphériques non partageables, etc)
- priorité de planification des travaux.

L'ordre de traitement des travaux n'est pas forcément celui dans lequel les travaux sont déposés au centre de calcul ou dans la file d'attente. Le responsable du centre de calcul peut définir des trains particuliers (trains de nuit par exemple) pour les travaux de longues durées de façon à réduire le temps de réponse de l'interactif ou des autres trains de travaux. Exemple:

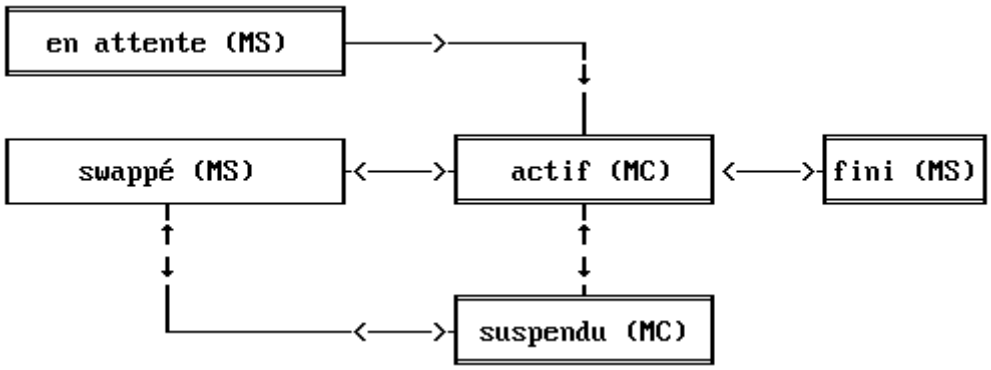
Classes	Nb Max	Tps CPU Max	Mém. Max	Etat
Petits_jobs	10	300s	4 Mo	ON
Grands_jobs	2	3600s	16 Mo	OFF le jour
Interactifs	10	infini (*)	4 Mo	ON

(*) en général chaque commande interactive (process) a une limitation en temps CPU et en occupation mémoire.

APPLICATION - Programme SE_BATCH

Le programme SE_BATCH permet la simulation d'un traitement Batch. Les caractéristiques connues d'un job sont le temps présumé d'exécution et l'occupation de la mémoire centrale. Chaque job consomme exactement le temps demandé et occupe exactement le maximum de mémoire demandé. Dans la réalité, il en va tout autrement. Un job peut finir avant pour de multiples raisons et la taille de la mémoire occupée peut varier en cours d'exécution.

La mémoire centrale simulée est composée de 5 cases mémoires. Les jobs soumis occupent 1 ou 2 cases contiguës. Un job demandant 3 cases ou plus est rejeté. Une limitation du temps d'exécution a été fixé à 500 unités CPU. Les états possibles d'un job sont:



Un job est *swappé* pour laisser sa place en mémoire centrale à un autre job. Ceci permet d'exécuter plus de jobs en parallèle. Dans le programme SE_BATCH, le "*swapping*" n'est pas pris en compte.

Le schéma se simplifie donc. Au début un job est donc en MS et attend de rentrer en exécution (en MC). Il peut être suspendu par une intervention de l'opérateur (groupe d'exploitation) et ceci a pour conséquence de privilégier les autres jobs en compétition. La possibilité de tuer (*kill*) un job n'a pas été prise en compte.

Lorsque le programme SE_BATCH met un job en MS (mémoire secondaire simulée) il n'y a pas forcément une image de ce job simulé dans la vraie mémoire secondaire. On a choisi dans le modèle présent d'avoir en MS réelle les jobs en attente (fichiers .JOB) ainsi que les jobs finis (fichiers .LST).

L'arrêt du programme SE_BATCH est équivalent à l'arrêt de l'exploitation simulée. En principe tous les jobs doit être sauvés en mémoire secondaire simulée et donc ici forcément en MS réelle. Si la possibilité de "*swapping*" avait été programmé, on aurait choisi de mettre l'image (*core*) du job *swappé* en MS vraie de telle manière qu'un arrêt suivi d'une nouvelle exécution du simulateur SE_BATCH puisse reprendre ces jobs *swappés* là où ils étaient (relecture de fichiers .SWP). Dans la version actuelle, l'arrêt puis la reprise du programme SE_BATCH reprend les jobs à leur début (en principe et dans la réalité tous les jobs ne s'y prêtent pas à moins d'avoir été prévus pour car il faut prendre en compte les E/S et des mises à jour des fichiers).

L'algorithme d'ordonnancement est de type "tourniquet". Un quantum de temps d'une unité CPU est donné tour à tour à chaque job actif en mémoire centrale simulée. Afin d'avoir un comportement qui se rapproche de la réalité dans laquelle: un job consomme du CPU, fait des E/S ou bien de manière plus générale fait consommer du CPU au système, on a choisi un modèle probabiliste dans lequel le job consomme soit CPU "user" soit du CPU 'système".

Le programme SE_BATCH présente à l'écran une image des jobs en exécution. Cette image évolue pour rendre compte de l'évolution des données et du compteur ordinal.

Bien que le "*swapping*" n'a pas été pris en compte, l'image des jobs en cours est sauvée puis restaurée à chaque appel de l'interpréteur de commande DOS.

Le fichier de commande QSUB.BAT permet de mettre un job dans la file d'attente gérée par ordre alphabétique. Le script QSUB_ALL.BAT génère un ensemble de jobs tests.

Un compactage de la mémoire centrale a été programmé. Lorsque le compactage survient le simulateur passe en mode manuel et permet de voir étape par étape l'évolution du compactage.

Imaginer un système de priorité prenant en compte efficacement ce type d'exploitation.

Procédures utilisées

TxCURSOR	Cache et fait apparaître le cursor écran
TxUpCASE	Conversion en majuscules d'une chaîne
GetCMDE	Lecture d'un caractère
GestionBREAK:	Gestion du break
AFF_TIME	Retourne la date et l'heure courante
EPILOGUE	Epilogue systématique du simulateur
Change_FIC	Change le suffixe d'un fichier
CASE_LIBRE	Teste si cases libres, retourne l'adresse MC simulée correspondante
CASE_LIBRE_COUNT	Nombre de cases libres au total
CASE_CLEAR	Efface une zone MC simulée
CASE_SAUVE	Sauve l'image d'une zone MC simulée
CASE_RESTO	Restauration de l'image sauvée
CASE_CRUNCH	Compactage de la MC simulée
PROCESS_LIBRE	Teste si entrée libre dans la table des jobs
PROCESS_INIT	Démarrage d'un job, attribution d'une zone MC
PROCESS_SAUVE	Sauvegarde de toutes les images de jobs
PROCESS_RESTO	Restauration des images
PROCESS_CONFLIT	Détermine si CPU user ou CPU système
PROCESS_WRITE	Mise à jour de l'image de la MC simulée
PROCESS_RUN	Exécution d'un job pour une unité CPU
PROCESS_SWITCH	Changement en bascule de l'état actif et suspendu
PROCESS_ISOK	Retourne vrai si un fichier .JOB n'est pas connu du simulateur ou bien qu'il est connu "en attente" et que de la MC simulée est disponible.
PROCESS_AJOUT	Ajout d'un JOB en attente, lecture des caractéristiques avant INIT
PROCESS_TRI	Tri de la table des processus
PROCESS_MAJ	Si place disponible en MC : lecture des fichiers en attente NOM.JOB et ajout éventuel en MC simulé
PROCESS_LISTE	Affichage de l'état des processus à l'écran