V. Gestion des ressources.

1. Introduction.

La gestion des ressources d'un système supportant la multiprogrammation demande au système d'exploitation de faire constamment des choix dans son allocation de ressources. La gestion d'un système mono-tâche étant par contre relativement évidente, un seul processus peut disposer de l'ensemble des ressources.

L'introduction de la multiprogrammation impose le partage de toutes les ressources (par définition limitées) entre les différents processus présents en machine. Ces processus peuvent recevoir une priorité qui elle-même peut déprendre des ressources qu'ils mobilisent. De plus, nous avons vu que la multiprogrammation introduisait l'état *Ready* pour un processus qualifiant son état d'attente sur une ressource particulière: l'unité centrale.

Les techniques de partage des ressources varient d'un système d'exploitation à l'autre, elles sont en général concentrées dans un composant du système d'exploitation appelé *scheduler*. Ce scheduler doit mettre en œuvre une politique d'allocation des ressources qui dans les systèmes évolués est fixée par l'administrateur du système.

Parmi les objectifs que le scheduler tente d'atteindre, relevons les principaux:
- s'assurer que l'ensemble des processus reçoivent une part des ressources en fonction de leur importance;
- s'assurer de l'utilisation efficace de l'ensemble des ressources: CPU, mémoire, canaux... :
- répondre rapidement aux utilisateurs interactifs;
- assurer un débit batch maximal.

Certains objectifs sont contradictoires : relevons simplement la difficulté de satisfaire à la fois les utilisateurs interactifs et batch ou assurer un bon temps de réponse alors que les ressources sont utilisées efficacement (CPU 100% par exemple). On comprendra que les systèmes évolués mettent en œuvre une politique d'allocation de ressources qui est définie par un administrateur. Ces politiques doivent être distinguées des mécanismes mis en œuvre au moment de l'allocation qui sont de plus bas niveau, sont plus simples et activés par le composant appelé le dispatcher.

A. Ressources.

Tout élément présent dans un système en quantité limitée doit être considéré comme une ressource. Ces ressources sont allouées ou partagées entre les processus sur décision du système d'exploitation qui

- prévient les inter-blocages au moyen des techniques expliquées précédemment;
- gère les files d'attente pour l'emploi de ces ressources ;
- gère les structures de données décrivant ces ressources et leur état.

Parmi les ressources les plus couramment rencontrées, citons:

- les unités centrales dont l'état est décrit dans les registres de contrôle;
- la mémoire centrale dont l'état est décrit dans les tables de pages/segments;
- les périphériques décrits par un descripteur d'unité (Unit Control Block);
- la ou les mémoires secondaires;
- les fichiers dont l'état est décrit par des catalogues , directory, I-node…

Différents composants du système d'exploitation interviennent dans l'allocation de ces ressources. Le contrôle de l'unité centrale n'est pas accordé par le même composant en charge de l'utilisation d'un fichier partagé.

B. Composants OS

Les systèmes d'exploitation mettant en œuvre des politiques d'allocation des ressources évoluées sont construits de manière modulaire afin de minimiser l'overhead inévitable associé à la politique d'allocation.

Le composant de plus bas niveau mais le plus souvent activé est le *dispatcher*. Sa logique simple consiste à inspecter une ou plusieurs files d'attente et à allouer la ressource au premier client présent dans ces files.

Ce composant est activé soit lors de chaque changement d'état d'un processus soit lors d'une demande d'allocation d'une ressource. La fréquence d'exécution impose la contrainte de simplicité: on ne peut se permettre d'ajuster des priorités ou de réarranger des files d'attente à chaque fois qu'une interruption survient dans un système: plusieurs milliers de fois par secondes.

Un composant de plus haut niveau appelé *scheduler* s'occupe de l'allocation des ressources, gère les priorités et décide de l'introduction de nouveaux processus au sein du système. La figure ci-dessous illustre cette complexité au travers des différents états et transitions de processus dans un système.

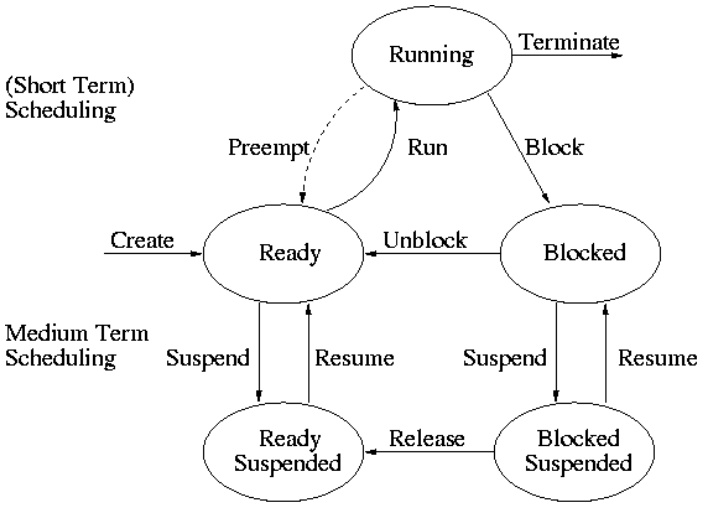

Fig. 56. System Resource Manager Queues.

C. Nouveaux processus.

Le scheduler assure la création de nouveaux processus si les ressources couramment allouées permettent la création de ce processus. Ainsi dans un système batch, les jobs seront mis en attente dans des files d'attente correspondant à certaines classes d'exécution, ces classes d'exécution pouvant elles-mêmes correspondre à l'emploi de ressources particulières (imprimantes, armoires à bandes/K7…). Un job n'entrera pas en activité si le scheduler juge que les ressources qu'il désire s'allouer sont indisponibles. De même, un processus utilisateur ne sera pas créé par le scheduler dans un système interactif s'il s'avère que les ressources sont insuffisantes pour l'accepter: nombre maximum d'utilisateurs, mémoire disponible…

D. Priorités.

L'ordre d'activation d'un processus dépend de sa position dans une file d'attente et de l'importance de cette file d'attente vis à vis des autres files d'attentes. Le scheduler assigne les priorités aux processus (dispatching priority) en fonction de différents algorithmes que nous détaillons plus loin.

E. Politique d'allocation des ressources.

Ces politiques ont deux objectifs :

- éviter les blocages dans le système ou carrément le blocage du système ;
- assurer un usage équilibré de ces ressources c.à.d. empêcher qu'un processus ne s'attribue le monopole d'emploi d'une ressource.

F. Workload Manager.

Enfin, dans le cas de systèmes distribués (que nous définirons pour l'instant comme des systèmes d'exploitation coopérant à la réalisation d'un ou de plusieurs travaux de nature similaire), on trouve un composant chargé d'assurer la distribution équitable du travail, ce composant est intitulé workload manager.

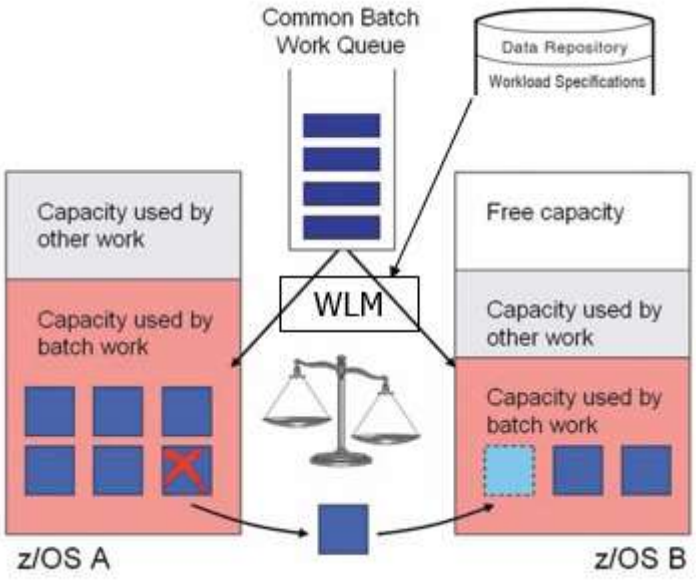

Fig. 57. Principe du Workload Manager.

Le workload manager doit être considéré comme un scheduler de plus haut niveau permettant la prise en charge d'un processus au sein d'un système d'exploitation en fonction de l'activité de ce dernier. Son objectif principal est de distribuer la charge de travail de manière équitable entre les différents systèmes d'exploitation qu'il gère en fonction du degré d'occupation de ces derniers.

2. Les algorithmes de scheduling.

Nous allons passer en revue dans ce chapitre les algorithmes de scheduling et en premier lieu les algorithmes les plus simples. Ces premiers algorithmes supposent que le ou les processeurs constituent la ressource principale. Rappelons-nous pour comprendre l'évolution des algorithmes de scheduling que les premiers systèmes d'exploitation supportaient la présence simultanée en mémoire de plusieurs processus de type batch (deck de cartes) et que les systèmes interactifs et de gestion de base de données n'ont été introduits que plus tardivement entraînant une inflation dans la complexité de la gestion des ressources, complexité liée à la diversité des charges de travail.

A. Scheduling simple.

a) Absence de scheduling.

Dans les premiers systèmes, le scheduling consistait à introduire un deck de cartes dans un lecteur. Le scheduler lit alors simplement les cartes, alloue éventuellement les unités, charge le programme en mémoire et lui passe le contrôle. Il n'existe donc pas de partage de ressources, le scheduling est uniquement provoqué par un événement externe : une touche sur le *card reader*.

b) Le premier job le plus court.

La file d'attente des jobs batch est ordonnée suivant les durées d'exploitation requises pour chacun d'eux. L'algorithme n'est évidemment valable que pour les environnements batch où l'on peut obtenir une estimation du temps d'exécution dans la description du job. Le but est de minimiser le temps d'exécution des jobs courts. Une version améliorée permet l'interruption d'un processus en cours au profit d'un autre plus court, ce dernier exerce un droit de préemption.

c) Round_robin.

Développé dans le but de répondre rapidement à une requête de courte durée, lorsque les temps d'exécution sont inconnus. Chaque processus présent dans le système reçoit un quota déterminé de temps de service (time slice) avant d'être reclassé en fin de file d'attente.

Les processus qui exigent de longues périodes d'exploitation tourneront plusieurs fois dans la file d'attente avant d'être achevés, les processus qui demandent moins de temps que le quota seront achevés dès le premier tour. Chaque processus est l'égal de son voisin, aucune priorité relative n'est attribuée. Le choix du quota doit faire l'objet d'un compromis entre le temps passé en changement de contexte et le temps d'attente d'un processus pour obtenir le contrôle du processeur.

B. Scheduling complexes.

Chaque processus se voit assigner une priorité, le dispatcher doit alors choisir le premier processus placé dans la file d'attente la plus prioritaire. On définit une file d'attente par priorité, le scheduler place alors les processus dans la file d'attente correspondant à leur priorité.

Les priorités peuvent être assignées de manière fixe ou être adaptées dynamiquement. Des techniques telles que l'*aging* peuvent être appliquées, elles consistent à augmenter la priorité d'un processus au fur et à mesure de son temps d'attente dans la file d'attente.

Les critères qui régissent l'attribution des priorités peuvent être les suivants:
- les processus qui détiennent un grand nombre de ressources peuvent obtenir une haute priorité afin de pouvoir s'achever rapidement et libérer les ressources;
- les processus attachés au système d'exploitation doivent recevoir les priorités proportionnelles à l'urgence des fonctions réalisées;
- les gestionnaires d'unités périphériques doivent recevoir une haute priorité. En général, plus l'unité est rapide plus la priorité de son gestionnaire pourra être rapide.

Les algorithmes complémentaires et de plus haut niveau tiennent compte de l'occupation mémoire, de l'activité de pagination... afin de décider de la présence de processus en mémoire centrale, on parle alors de swapping.

a) Scheduling Unix

Unix est un système en temps partagé, l'algorithme de scheduling est conçu pour offrir un temps de réponse correct aux processus interactifs.

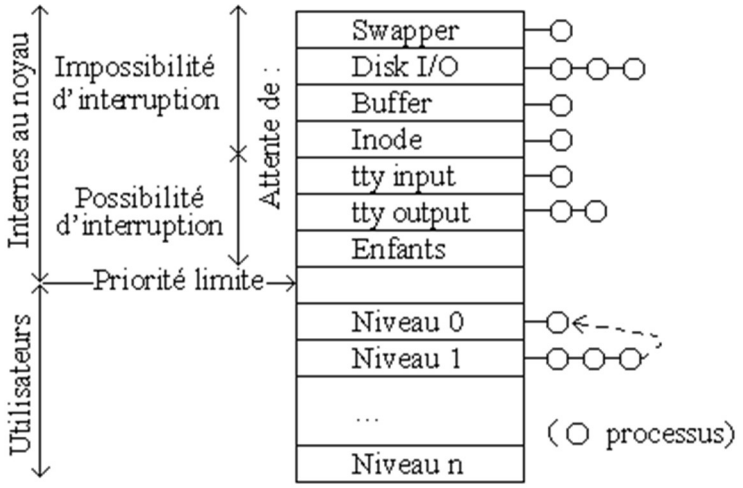

Fig. 58. Gestion des priorités en Unix.

Un dispatcher sélectionne les processus dans des files d'attente associées chacune à une priorité. Un algorithme de scheduling de plus haut niveau arrange les transferts entre mémoire et disque (swapping) afin que chaque processus ait une chance de recevoir le contrôle du processeur.

Les processus s'exécutant en mode utilisateur possèdent une priorité de valeur positive, les processus s'exécutant en mode superviseur (Kernel) possèdent une priorité de valeur négative[8], seuls les processus présents en mémoire et dans l'état ready to run sont placés en file d'attente.

Le premier processus de la file d'attente possède la priorité la plus élevée. Ce processus peut disposer du processeur durant un temps correspondant au maximum à un quantum. La consommation CPU d'un processus utilisateur est ajoutée à sa priorité à la fin du quantum, augmentant la valeur qui lui est assignée et donc diminuant sa priorité. Périodiquement (toutes les secondes), la consommation en CPU est recalculée afin d'atténuer l'impact de cette technique.

Les processus peuvent exécuter un appel système appelé "nice" leur permettant d'ajuster leur priorité en faveur des autres processus présents en mémoire. Seul l'administrateur du système peut attribuer des valeurs "nice" négatives.

La priorité d'un processus utilisateur est calculée comme suit:

$$Priority = (CPU/2) + Start\ priority + nice.$$

La valeur du CPU est calculée à partir des interruptions générées par l'horloge du système, sa valeur est recalculée toute les secondes (divisée par 2) afin d'atténuer l'effet permanent qu'aurait la consommation CPU sur la priorité d'un processus.

Les processus en mode superviseur (Kernel) ont une valeur "priority" inférieure à la valeur seuil, plus l'algorithme est de bas niveau plus la valeur est basse, donc plus la priorité du processus est élevée. Cette technique est adoptée afin de libérer le plus rapidement possible les ressources les plus critiques car le plus souvent sollicitées.

Le scheduler ne choisit que des processus présents en mémoire et dans l'état ready to run. Un processus appartenant au système Unix (kernel) appelé swapper organise le transfert des processus entre la mémoire secondaire et la mémoire centrale (swap-in et swap-out). Les objectifs de ce processus sont de libérer de la mémoire centrale et de donner une chance égale à tous les processus utilisateurs d'être schédulés.

b) Scheduling MVS.

MVS est un système d'exploitation général, il est à la base un système batch ayant évolué vers le support du temps partagé et du transactionnel. Les ressources doivent donc être partagées entre des processus qui ont des caractéristiques différentes: la charge de travail (workload) est mixte. L'algorithme de scheduling est relativement élaboré et met en œuvre une politique définie par l'administrateur du système et inscrite dans des paramètres (IPS/ICS)[9].

---

[8] ou en dessous d'une certaine valeur seuil. S'il s'agit de valeurs négatives, 0 est la valeur seuil.

[9] ICS: catégorisation du travail (Installation Control Specifications)

- IPS: assignation de priorités (Installation Performance Specifications)

Les processus dans un système MVS sont rassemblés en trois grandes catégories:

- Sous-systèmes : processus remplissant une mission de base, ils sont donc prioritaires. Ils peuvent être assimilés aux daemons que l'on trouve dans un système Unix.
- Batch : processus traitant des lots en background. Ces processus sollicitent fortement les unités centrales, les sous-systèmes I/O ou la mémoire.
- On-line : processus interactif qui demandent un temps de réponse constant et court.

Le gestionnaire du système ajuste l'utilisation des ressources grâce à différentes fonctions mises en œuvre par le scheduler (SRM, pour Server Resource Manager):

- ajustement du *maximum multiprogramming level* (MPL), détermine combien de processus peuvent être prêts à être *dispatchés*. Donc présents en *central storage* et dans l'état *ready*. Si l'utilisation du CPU monte ou si la pagination augmente, le scheduler réduit le MPL.
- *Logical swapping*, pour autant que suffisamment de mémoire réelle soit présente, les processus dans l'état *blocked* restent en mémoire afin de favoriser les temps de réponse.
- *Working set management* : déterminer le nombre minimal de pages constituant le *working set* d'un processus, utilisé en relation avec l'activité globale du système. Si le système pagine et que le *working set* d'un processus entraînerait en cas d'activation une dégradation de la situation, le scheduler diminue la priorité du processus.

Les priorités sont assignées suivant l'importance du processus, elles peuvent être fixées ou varier en fonction:

- de la consommation en ressources : aging équivalent à celui présent UNIX mais pas uniquement basé sur l'utilisation des unités centrales,
- de l'activité en entrées/sorties. *Mean time to wait* : la priorité augmente en fonction de l'activité I/O.

i. Notion de Service Unit

La notion de service unit (SU) permet de caractériser la consommation de ressources d'un processus. Le scheduler va mesurer le taux de service c'est à dire la rapidité à laquelle les service units sont consommées par un processus.

Chaque emploi d'une ressource (secondes CPU, nombre I/O et pages mémoire) est transformé en un nombre d'unités de service globales. Les poids associés à chaque ressource sont choisis afin de refléter l'importance ou la rareté de cette ressource. Cette unité permet également d'établir une indépendance vis à vis des modifications de configuration. En effet, un processus nécessitant 1 seconde de travail d'une unité centrale capable d'exécuter 50 millions d'instructions par seconde ne prendra plus que 500 millisecondes sur un processeur deux fois plus rapide ; le travail réalisé est pourtant identique, les unités de services consommées par le processus resteront identiques, seul le débit du processeur change: le nombre de service units délivrées par seconde.

Les service units peuvent être employées dans le cadre d'une politique d'accounting (facturation des ressources informatiques aux utilisateurs). Imaginons qu'une machine bi-processeur d'une puissance unitaire (par CPU) de 500 MIPS soit capable de délivrer

3000 SU/sec. 1000 SU sont facturées 1 EUR. Un processeur plus rapide d'un facteur 2 est installé, il est capable de délivrer 6000 SU/Sec. Si nous avions facturé la seconde CPU, soit l'utilisateur aurait vu une réduction directe de ses coûts, alors que les coûts d'exploitation ne sont pas fondamentalement modifiés, soit nous aurions dû modifier nos calculs de facturation ; l'emploi des services units rend l'installation transparente en adaptant le poids (et donc le coût) de la seconde CPU dans la Service Unit.

ii. Assignation de caractéristiques

L'administrateur du système va placer les différents processus dans des groupes afin d'assigner les priorités, l'algorithme de variation de ces priorités et d'ajuster le niveau de multiprogrammation.

Les sous-systèmes tels que TP monitor, gestionnaire de communications, gestionnaire de base de données, gestionnaire de lock, gestionnaire de mémoire… ont un niveau de multiprogrammation sans limite. Aucun de ces processus ne sera mis en dehors de la mémoire (swapping), le working set ne subira aucune limitation. Leur priorité est fixe et élevée, ces processus rendent des services communs aux processus utilisateurs.

Les processus batch sont récurrents, leur durée connue et leur soumission est planifiée. Le niveau de multiprogrammation est limité, les processus dont le working set est important peuvent subir des swaps en cas de contraintes fortes sur la mémoire centrale. La priorité est basse et variable suivant deux algorithmes *Mean-to-Wait* croissante en fonction de l'activité I/O, et *aging* décroissante en fonction du nombre d'unités de service consommées.

Les processus on-line sont démarrés en réponse à une commande tapée par un utilisateur, le temps de réponse doit être court mais les commandes lourdes doivent être pénalisées. Le niveau de multiprogrammation est limité mais doit faire l'objet d'un suivi de la part de l'administrateur du système (nous reviendrons dans le chapitre suivant sur ce sujet). Le *logical swapping* peut être mis en œuvre si la mémoire centrale est peu sollicitée. La priorité est fixe pour les commandes légères mais est gérée par un algorithme *Mean-to-Wait* au fur et à mesure de la consommation d'unités de service.

La figure 59 illustre les différentes transitions subies par un processus. Comme en Unix, seul un processus présent en In Queue (Ready) pourra être choisi par le dispatcher suivant une priorité. Les transitions d'états sont soit dues à la mise en état d'attente du processus par lui-même soit dues à des décisions prises par le SRM: improve paging, Unilateral swap, Exchange swap...

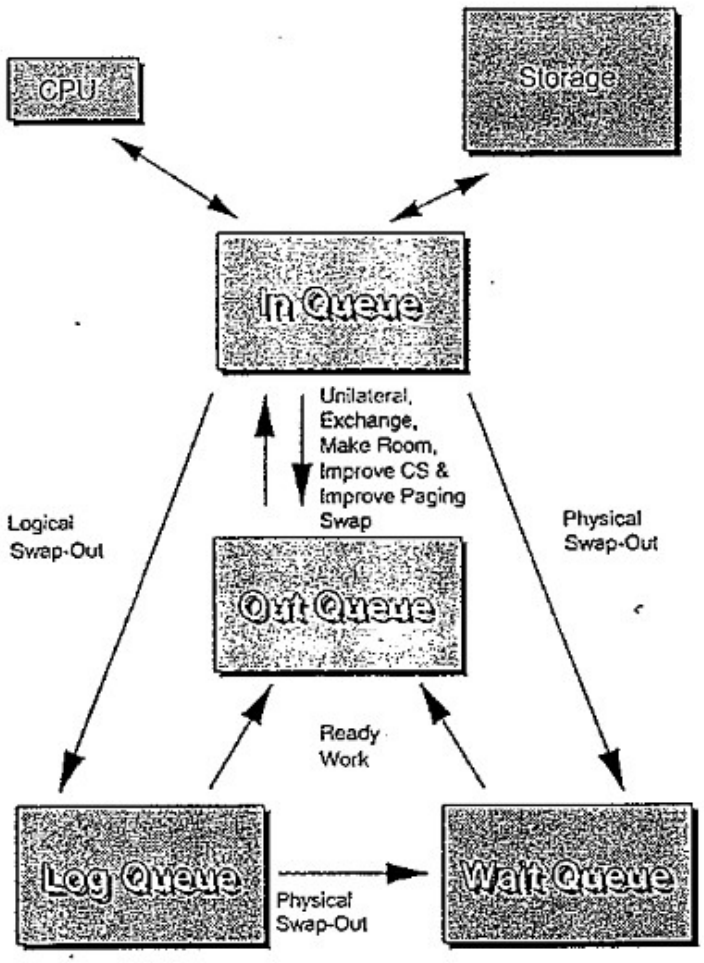

Fig. 59. Transition d'état des processus en MVS.

Ces politiques se traduisent comme nous l'avons indiqué précédemment par des paramètres (IPS/ICS).

3. Gestion des systèmes loosely coupled.

Ces systèmes sont constitués de différentes images d'un même système d'exploitation travaillant à la réalisation d'un ensemble de tâches en collaboration lâche (loosely). Ils peuvent se présenter sous forme de

- Pool de processeurs
- Distribué
- Hybride

A. Type d'algorithme

Les algorithmes de gestion des ressources possèdent alors les caractéristiques suivantes:

- Déterministe (conditions connues) vs Heuristique
- Centralisé (un processus en charge de la gestion des *workload*s) vs Distribué
- Optimal (essayer de trouver le meilleur) vs Suboptimal
- Local (influencé par des conditions locales) vs Global
- Initié via émission (mission de l'état *overloaded*) vs réception

a) Algorithme déterministe

Cet algorithme est basé sur un graphe représentant le flux de données entre processus.

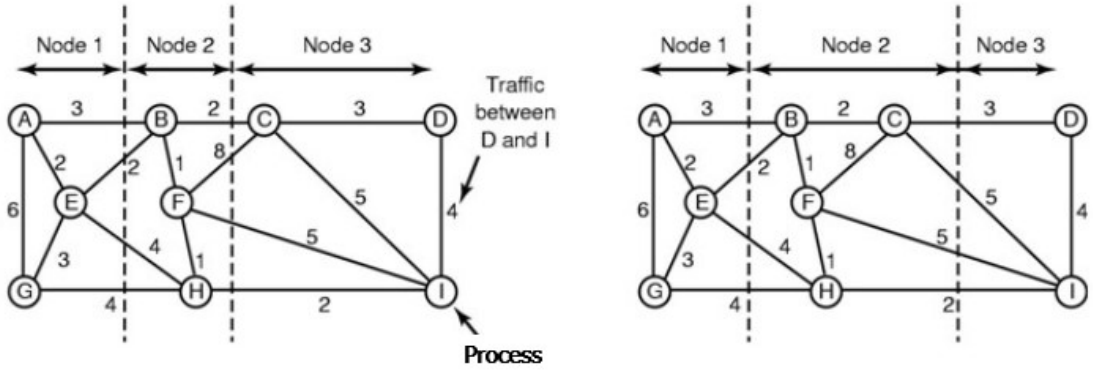

Fig. 60. Allocation des processus dans un système Loosely- coupled

Le système peut être représenté comme un graphe avec des poids. Chaque nœud représente un processeur, chaque arc représente le flux de messages entre deux processeurs. L'objectif de l'algorithme de scheduling est de partitionner ce graphe en sous-graphe suivant certaines contraintes liées à l'utilisation des processeurs et de la mémoire.

Tous les processus d'un même sous-graphe seront assignés à un seul processeur, les échanges de messages entre processus d'un même sous-graphe peuvent être ignorés. L'objectif est d'aboutir à une découpe en sous-graphes qui respecte les contraintes et aboutit à un trafic inter-processeur minimal.

b) Algorithme heuristique décentralisé Hiérarchique

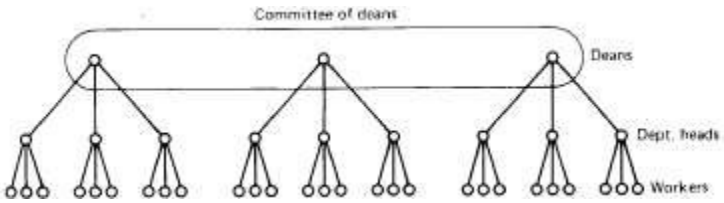

Fig. 61. Algorithme hiérarchique (heuristique décentralisé).

Dans des grands systèmes, un algorithme déterministe centralisé peut devenir un bottleneck. On lui préfère alors un algorithme hiérarchique décentralisé. Si le coordinateur d'un niveau estime ne pas pouvoir répondre à une requête (M processus), il remonte la requête au niveau supérieur.

c) Message passing

Le scheduling de processus au sein d'un système distribué doit faire appel à des techniques dites de co-scheduling lorsque des processus coopérants s'exécutent sur des processeurs différents. Normalement, les processus s'exécutent de manière indépendante au sein d'un système d'exploitation, ces processus sont alors gérés suivant les techniques de scheduling mises en œuvre dans un système classique. Des problèmes peuvent survenir lorsque des processus coopérants s'exécutant sur des systèmes différents doivent s'échanger des messages et que leur exécution est désynchronisée faute d'une politique de scheduling adéquate.

Un exemple d'algorithme tenant compte de cette problématique est illustré au moyen de la figure ci-dessous. Les colonnes du premier tableau représentent les processus assignés à 2 processeurs. A chaque quota de temps, le processeur active un processus, une rangée du tableau correspond à chaque quota de temps. L'algorithme de co-scheduling tente d'harmoniser le choix d'une time slice pour les processus coopérants. Imaginons que A doive communiquer avec D, suivant l'illustration si A et D ne sont pas schédulés en même temps, des délais d'attente égaux à 2 time slices seront observés lors des communications entre A et D.

Time slice	Processor 0	Processor 1
	A	C
	B	D
	A	C
	B	D
	A	C
	B	D

Processor 0	1	2	3	4	5	6	7
X				X			
		X			X		
	X			X		X	
X					X		
	X		X				X
		X		X			

Fig. 62. Arrangement des time-slices en fonction du trafic.

Le second tableau illustre le principe du co-scheduling, les rangées correspondent ici aussi à des quanta de temps (time slice), chaque colonne correspond à un processeur.

L'idée de base de cet algorithme consiste à synchroniser les time slices sur les différents processeurs et à arranger l'exécution de processus communicants entre eux au sein de la même time slice. Une étude du comportement des processus est nécessaire afin de déterminer des groupes de processus à rassembler au sein d'une même time slice.

d) Partitionnement

Le partitionnement est une technique qui consiste à partager une configuration (unités centrales, mémoire, canaux,..) entre plusieurs partitions ou domaines au sein desquels une instance du système d'exploitation s'exécute.

Chacun des systèmes d'exploitation est activé en fonction de son importance relative. Notons toutefois qu'il est important d'assigner un ou plusieurs processeurs à un système d'exploitation afin d'obtenir de bonnes performances. Le partitionnement de la mémoire est fixé mais peut être modifié au moyen de commandes opérateurs.

4. Instrumentation et suivi de la capacité.

Nous allons aborder dans ce paragraphe la problématique du suivi de la capacité (Capacity Planning) et des performances des systèmes.

A. Systèmes équilibrés.

Nous devons, avant d'entamer cette discussion, revenir sur une notion importante: l'équilibre d'un système. Ce concept se base sur la relation entre les différentes ressources d'un système comme illustrées à la figure ci-dessous.

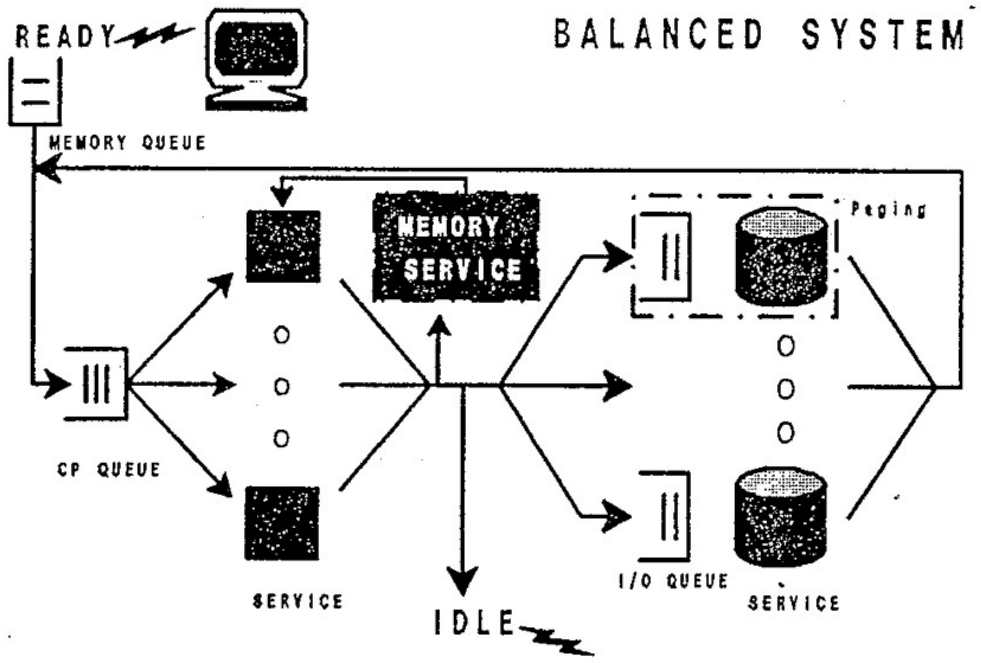

Fig. 63. Modélisation d'un système.

Le modèle repris dans cette figure fait abstraction des aspects propre au réseau, mentionnons toutefois que le temps de réponse constitue le seul et unique facteur caractérisant les performances d'un système aux yeux d'un utilisateur. Ce temps de réponse doit être décomposé en un temps de réponse système et un temps de réponse réseau (in et out).

Suivant les installations, l'un ou l'autre composant domine le temps de réponse perçu par l'utilisateur. Notons également qu'un utilisateur est peu sensible à des variations de temps de réponse de l'ordre de 1/10 de secondes. A titre d'exemple, un environnement de type transactionnel desservant des utilisateurs répartis sur l'ensemble de la Belgique (WAN) offre un temps de réponse moyen de 1.3 secondes, le temps de réponse système comptant pour 0.3 secondes. Ces proportions seront peut-être inversées dans le cas d'un utilisateur d'un PC attaché à un LAN interrogeant une base de données relationnelle (SQLServer, Oracle…)

Les performances du système sont déterminées par le flux de travail au travers des composants qui le constitue. La loi des flux forcés implique que le travail doit pouvoir

s'écouler au sein du système comme s'écoule un fluide au travers d'un ensemble de conduites.

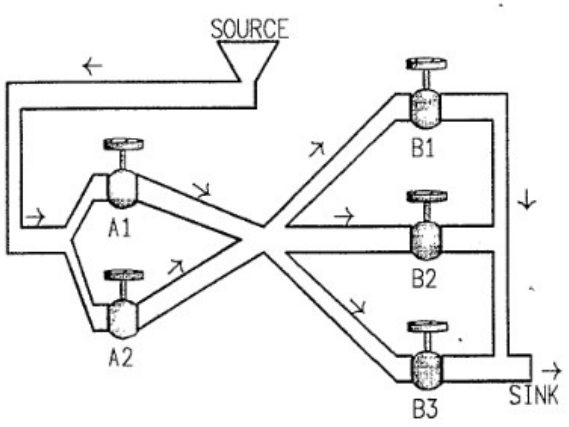

Fig. 64. Système balancé.

La loi des flux forcés implique que le flux traversant les robinets A est égal au flux traversant les robinets B. Si ce flux n'est pas égal, le système est déséquilibré.

B. Capacity Planning

Un gestionnaire de systèmes doit avoir à tout moment une idée du comportement de son système afin de pouvoir répondre aux questions fondamentales:

- Le système est-il toujours équilibré?
- Le système sera-t-il toujours équilibré à l'avenir?

L'équilibre actuel du système conditionne la satisfaction des utilisateurs soit en terme de temps de réponse soit en terme de débit batch. Un système balancé garantit une rentabilité correcte de l'installation.

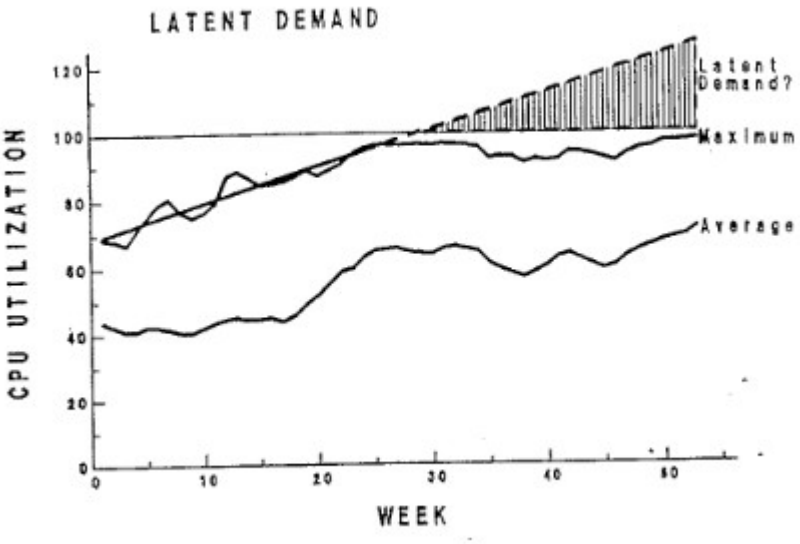

Fig. 65. Apparition de demande latente.

Le capacity planning permet de suivre les tendances d'évolution du système: le nombre d'utilisateurs interactifs augmente-t-il, le nombre de transactions augmente-t-il...? Ces évolutions servent de base de décision de nouveaux investissements. En conséquence, les évolutions à 3 ou 6 mois doivent pouvoir être prévues car la mise en œuvre d'une nouvelle configuration peut prendre du temps. La figure précédente illustre le concept de demande latente, la capacité d'un système peut être nettement sous-dimensionnée par rapport à la charge de travail soumise, toute augmentation de capacité va être consommée dès qu'elle est mise à disposition.