Omar Lahlou & mister Patry
time slice : unité de temps dans un systeme mutitache preemptif pdt laquelle la processus à le droit de s'executer.

swap : plus de place dans la ram, mais si un processus neccesite de la ram on a des algo (LRu ou LFU) pour en trouver.
LFU : la zone de swap la plus ancienne est placé sur le disque dur.

préemptif : ordonnaceur preemptif presente l'avantage c une meilleur reactivite du systeme et de son evolution, en stopant un processus par un autre; inconveniant : competition lorsque deux processus accedent aux meme ressources. (remedier par des lock)

I-questions d'echauffement

que pensez-vous de la phrase suivante ?
« L'ordonnanceur est un processus qui se réveille régulièrement et change les processus en cours d'exécution sur les différents processeurs. »

Définition :
La fonction d'ordonnancement gère le partage du processeur entre les différents processus en attente pour s'exécuter, c'est-à-dire entre les différents processus qui sont dans l'état prêt. L'opération d'élection consiste à allouer le processeur à un processus.

oui mais. ne change pas de processeur d'un processus en cours.

Quels sont les points communs et différences entre un appel-système et une interruption au niveau matériel et logiciel ?

Interruption: est un arrêt temporaire de l'exécution normale d'un programme informatique par le microprocesseur afin d'exécuter un autre programme (appelé service d'interruption).

Appel systeme: un **appel système** (en anglais, ***system call***, abrégé en ***syscall***) est une fonction primitive fournie par le noyau d'un système d'exploitation et utilisée par les programmes s'exécutant dans l'espace utilisateur(en d'autres termes, tous les processus distincts du noyau).

Quel est l'intérêt de l'appel-système vfork ? Que pourrait-il se passer si le père reprenait son exécution avant que le fils n'ait fait exec ou exit ?

> vfork - Créer un processus fils et bloquer le père.

/*As for *how* it works (modulo the `printf` issue), `vfork` suspends the parent process until the child terminates, so when `vfork` returns in the parent, the child has already exited.*/

>Le pere ne peut pas reprendre(car bloqué par le noyau) tant que le fils n'a pas fait exec ou exit.

Quels sont les objectifs principaux des systèmes d'exploitation ? On distinguera les intérêts des utilisateurs, des développeurs et ceux des administrateurs.

- Les deux objectifs majeurs d'un système d'exploitation sont:
  - Transformer le matériel en une machine utilisable, c.-à-d. fournir des outils adaptés aux besoins des utilisateurs indépendamment des caractéristiques physiques,
  - Optimiser l'utilisation des ressources (matérielles et logicielles).
- Ces deux objectifs doivent être atteints en garantissant un bon niveau en:
  - Sécurité: intégrité, contrôle des accès, confidentialité...,
  - Fiabilité: satisfaction des utilisateurs même dans des conditions hostiles et imprévues,
  - Performance du système informatique.

Dans un système d'exploitation préemptif, comment garantit-on qu'un processus en cours d'exécution va de temps en temps rendre la main au système et aux autres processus ? En quoi les appels-système interviennent-ils ? (martin)

L'ordonnanceur distribue le temps du processeur entre les différents processus. Dans un système préemptif, à l'inverse d'un système collaboratif, l'ordonnanceur peut interrompre à tout moment une tâche en cours d'exécution pour permettre à une autre tâche de s'exécuter.

/*Dans un système d'exploitation multitâche préemptif, les processus ne sont pas autorisés à prendre un temps non-défini pour s'exécuter dans le processeur. Une quantité de temps définie est attribuée à chaque processus ; si la tâche n'est pas accomplie avant la limite fixée, le processus est renvoyé dans la pile pour laisser place au processus suivant dans la file d'attente, qui est alors exécuté par le processeur. Ce droit de préemption peut tout aussi bien survenir avec des interruptions matérielles.*/

Le noyau est-il la même chose que le système d'exploitation ?

Non puisque certains morceaux de la partie utilisateur (userspace) font quand même partie de l'os

Quelle est la différence entre l'administrateur (root) et le noyau ? Quels privilèges ont-ils ? Qui met en place ces privilèges ?

Noyau -> « maître du système», accès matériel. noyau privilégié matériellement par la processueur.
Root -> utilisateur privilégié, dont les privilèges sont donnés par le noyau. root est privilégié logiciellement par le noyau.

Comparez les appels-système et les interruptions aux niveaux matériel et logiciel. Quels contextes d'exécution sont mis en jeu ? Qui, quand et comment est programmée la gestion de ces événements ? Doit-on les traiter immédiatement ou peut-on déferrer le traitement à plus tard, et pourquoi ?

Un appel système correspond à une interruption logiciel. Par contre, l'horloge déclanche des interruptions matériel.
En régle générale, une interruption va permettre de passer du mode utilisateur au mode noyau.
Qui : Les programmers de linux et des pilotes.
Quand : Avant que tu ne lance ton système. C'est-à-dire, dans un passé plus ou moins proche. (Sur une debian, c'est dans un passé très lointain. ^^)
Comment : En C ou en assembleur.
On doit les traiter au plus vite. Mais parfois il est necessaire de desactiver les interruptions. En régle générale on les traites immédiatement.

si trop long, on ne requisitionne pas tout le systeme pour gerer une interruption !
Et parfois il faut les déferrer. Si 1Go de données arrive sur ta carte réseau, tu ne va pas traiter les 1Go dans le traitant de l'interruption.
Tu ne fera que le necessaire et tu programmera le reste pour qu'il soit exécuter plus tard par un thread noyau.

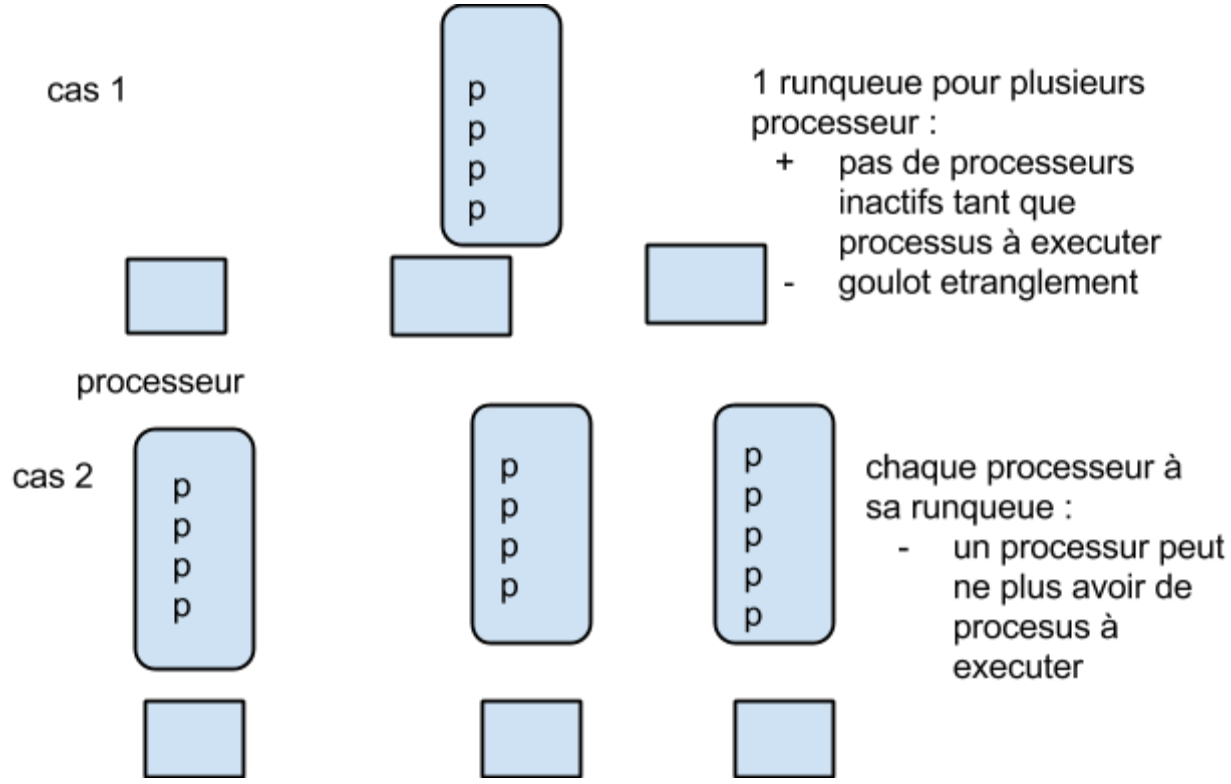
/*nous savons qu'un périphérique peut utiliser une demande d'interruption. Ceci signale au processeur que des données sont en attente (données arrivées sur le port ou données à envoyer)*/

Il y a des parties du code du système d'exploitation qui sont portables. D'autres qui sont écrites spécifiquement pour une architecture matérielle. Donner des exemples dans le cadre de la gestion de la mémoire virtuelle et de l'ordonnancement des processus. Qu'est-ce que cela implique sur le langage de programmation utilisé ? (insatisfaisant - martin)

-Gestion de la mémoire virturelle, MMU ou pas
-Ordonnancement des processus, désactiver les interruptions (code assembleur différent d'un processus d'un autre).
→ Le langage de prog doit etre portable.

Quel(s) avantage(s) et inconvénient(s) y a-t-il à avoir une runqueue (file de processus prêts) par processeur dans l'ordonnanceur du système d'exploitation ?

Plusieurs Runqueue -> accès plus rapide, pas de verrouillage à faire, Mais pbm d'équilibrage de la répartition des process.



cas 2 :
- \- equilibrer les queues
- \+ le processus s'execute tjrs sur le meme processuer et pas de goulot d'etranglement

Dans un système d'exploitation préemptif, quel mécanisme permet de garantir que les processus actifs vont de temps en temps être préemptés de force par l'ordonnanceur ?
Comment les appels-système interviennent-ils ? Quelle similarité y a-t-il avec le traitement des signaux ?

/*horloge : permet au noyau de reprendre le contrôle du processeur pour faire ce qu'il

a à faire, comme changer de processus en cours d'exécution. ordonnanceur se sert de l'horloge. */

-Les interruptions dans l'horloge.
-Dans chaque appel systeme, il y a un appel d'ordonnanceur caché.
-Similarité avec le traitement de signaux(a chaque fois qu'on reçoit un signal/interruption, on exécute une fonction

Le noyau connaît les traductions d'adresses virtuelles en adresses physiques. Mais pourquoi doit-il tout de même manipuler des adresses virtuelles lorsqu'il veut accéder à une donnée en mémoire (déréférencer un pointeur) ?
Le noyau doit utiliser des adresses virtuelles car le processeur doit faire la traduction adresse virtuelle vers adresse physique via la MMU.

/* Une **unité de gestion mémoire** (**MMU** pour *memory management unit*) est un composant permettant de contrôler les accès qu'un processeur fait à la mémoire de l'ordinateur dans lequel il est placé.*/

Quel est la différence entre la taille et l'occupation disque d'un fichier ? Dans quel(s) cas sont-ils différents ?

Taille = taille supposée du fichier / Occupation = taille réellement occupée par le fichier sur le disque. Dans le cas où le fichier n'est pas plein (taille>occupation), ou dans le cas où le fichier n'est pas un multiple de la taille d'un secteur (taille<occupation).

Comment les appels-système contribuent-ils à améliorer l'équité en alternant l'exécution des différents processus ? Comment fait-on si aucun appel-système n'est effectué ?

Durant un appel système, le noyau va faire en sorte de changer de processus (valable pour tout les apples systeme ?) . Si la tâche ne fait pas d'appel système, ce sont les interruptions de l'horloge qui vont arrêter la tâche courante. Et qui vont lancer une routine du noyau qui va s'occuper de changer de tâche. (C'est de la préemption.)

Si une page en copy-on-write ne contient que des zéros, que se passe-t-il quand un processus écrit à nouveau un zéro dedans (et donc ne modifie pas son contenu) ? Pourquoi ?

On a un défaut de page car le processus est fainéant il nous crée la page qu'au moment où on veut écrire dedans => défaut de page.

Je présume qu'il ne se passe rien. Tant que le processus ne change pas les données de la page, il ne devrait pas dupliquer la page. (Mais je ne suis pas sûr.)

Comment implémenter un logiciel « preload » qui fait en sorte que les programmes les plus couramment utilisés soient chargés en mémoire (mais pas lancés) au démarrage de la machine (sans provoquer de swap génant) ?

Lire les exécutables à pré-charger (avec des read par exemple). Ainsi leur code sera placé dans le buffer-cache et aura déjà des pages allouées en mémoire centrale.
Donc lors d'un véritable lancement, pas besoin de les charger depuis le disque, ni de leur allouer des pages.

Dans quel cas (événement et attributs de la page) le système doit-il remplir une page de zéros pour éviter une fuite d'information entre deux processus ? Vaut-il mieux faire le remplissage le plus tôt ou le plus tard possible ?

Lorsque le processus essaye de lire la page. (Mais je ne suis pas sûr.)
Définitivement, le plus tard possible. Le système est fainénant.
Quand on a besoin d'allouer une nouvelle page, le remplissage se fait le plus tard possible (process fainéant)

II-Pagination à la demande

On lance un programme qui alloue 400ko (mapping public anonyme) puis les remplit de zéros. Expliquez pourquoi le lancement de notre programme provoque environ 100 défauts de page. Expliquez ensuite pourquoi ces défauts de page disparaissent si le programme alloue la mémoire mais ne l'utilise pas.

Défaut de page : requête d'accès à une page non chargée (voir page 240)

On veut accéder à 400Ko -> 100pages on veut les manipuler (écriture) donc le système va devoir nous fournir 100 pages -> 100 défauts de page. on veut acceder a la page sans l'avoir chargée en mémoire, c'est un defaut de page.

Tant qu'on n'utilise pas les pages, le système ne les alloue pas, donc il n'y a pas de défaut de pages.

On remarque ensuite que le lancement du processus, même sans allouer de la mémoire ou la modifier, provoque tout de même quelques défauts de page. A quoi sont-ils dûs ? Que peut-on observer si on lance plusieurs fois le processus d'affilée peu après le démarrage de la machine ?

Les défauts de pages toujours présents sont ceux dû à l'allocation de la pile, de l'exécutable ainsi que les données initialisées dans le programme. Après la première lecture, on place les données lues dans un buffer cache, du coup la deuxième exécution ne causera pas de défaut de pages.

On modifie maintenant notre programme pour se dupliquer par fork après avoir alloué les 400ko de mémoire et les avoir remplis de zéros. Le fils créé par fork remplit alors les 400ko avec d'autres valeurs. Quand le fils a terminé, le père remet la mémoire à zéro. Qu'observe-t-on en terme de défauts de page ?

Environ 100 défauts de pages créés par l'allocation des 400Ko par le père. L'écriture faite pas le fils ne causera pas de défaut de page supplémentaire car c'est un mapping public anonyme, donc la mémoire est partagée. Lorsque le père remet les valeurs à 0, il n'y a pas de défaut de page non plus car les pages sont toujours les mêmes.

Pour les questions suivantes, le mapping est maintenant privé et non plus public. Que devient votre réponse à la question précédente ? Expliquez ce qui se passe et l'intérêt du modèle.

Il y a toujours les 100 défauts de pages créés par l'allocation des 400Ko par le père. Comme le mapping est privé, lorsque le fils va s'exécuter il causera 100 défaut de pages en écrivant sur la mémoire. Lorsque le père réécrit, il n'y a pas de défaut créé car les pages sont déjà allouées.

On modifie ensuite le programme pour lancer deux fils se comportant comme ci-dessus. Expliquez les différents états des pages (protection matérielle, compteur de références, ...) et les défauts de pages observés par les trois processus au fur et à mesure de leur exécution.

Au début de l'exécution, le processus père provoque 100 défauts de pages par l'allocation de 400Ko. La protection matérielle est alors RW (Read Write), et le compteur de références est à 1. Lorsque les fils s'exécutent, la protection passe à R et le compteur passe à 3(car deux fils + pere = 3) . Comme la mémoire est privée, ils (fils) allouent chacun 400Ko et créent chacun 100 défauts de pages (200 en tout). Le compteur repasse à 1 , la protection reste à R, jusqu'à ce que le père réécrive sur la mémoire, le système remet la protection à RW.

II. Table de pages(cours: vers page 222)

On suppose disposer d'une architecture 64bits avec 3 niveaux de table de pages matériels et des pages de 16ko (2 14 octets). La table de pages est constituée de tableaux de la taille d'une page. Ces tableaux peuvent être remplis avec des pointeurs vers les sous-niveaux (pointeurs de taille 8 octets) ou avec des PTE (de taille 16 octets)

definition : Une PTE (Page Table Entry) pour chaque cadre virtuel.

Rappelez rapidement ce qui est stocké dans chaque niveau de la table de pages. Pourquoi utilise-t-on en général des tableaux de la taille d'une page ?

/*définition : Un mécanisme de traduction assure la conversion des adresses virtuelles en adresses physiques, en consultant une *table des pages* (*page table* en anglais) pour connaître le numéro du cadre qui contient la page recherchée. L'adresse physique obtenue est le couple (numéro de cadre déplacement).*/

1er niveau : références vers les autres tables.
niveau final : les infos sur les droits

On utilise des tableaux de la taille d'une page pour la simplicité de gestion des pages. Pas + grand car difficulté de gestion des pages pas côte à côte, et pas + petit sinon pertes de données.

Combien de pointeurs ou PTE peut-on mettre par page ? En déduire le nombre de bits utilisés pour chaque niveau de la table de pages. Combien de bits sont nécessaires pour décrire un décalage dans une page ?

pointeurs de taille 8 = 2^3octets -> 2^11 pointeurs par table→ 2^14/2^3
PTE de taille 16 = 2^4octets -> 2^10 PTE par table
Premier niveau -> 11bits, pointeurs normaux -->Pointeurs
Deuxième niveau -> 11bits, pointeurs normaux -->Pointeurs
Troisième niveau -> 10bits, PTE -->que pte
Pour décrire un décalage de 2^14octets, il faut 14bits.

En déduire la taille maximale des adresses virtuelles manipulées sur cette architecture. Comment relier ce résultat au fait que l'architecture est présumée « 64bits » ? Que fait-on des bits restants ?

On a besoin de 46bits(bits niveau 1 + bits niveau 2 + bits niveau 3+ bits décalage= 11+11+10+14), l'adresse devra faire 46bits et l'architecture est de 64bits, donc on n'utilise pas le plein potentiel du système 64bits. Les bits restants sont mis à 0.

Quel espace mémoire est nécessaire pour stocker la table des pages d'un processus qui utilise l'intégralité de son espace d'adressage ? Et s'il n'utilise que 100 octets ?

(1 + 2^11 + 2^11.*2^11) *2^14 =
2^14 + 2^14*2^11 + 2^14*2^11*2^11,

Si il n'utilise que 100octets, on aura besoin que de 2^14*3(3 = nbre de niveau de pages).

IV. Synchronisation passive

a. Cas possibles: processus p lit la liste pendant qu'elle est modifiée:
- soit p ne prend pas en compte les modif alors qu'il faudrait
- soit p prend en compte les modifs alors qu'il ne faudrait pas

On utilise un verrou: lorsqu'un processus R/W la liste, aucun autre processus ne peux R/W.

V. Déduplication dans systèmes de fichiers

Si le système de fichiers ne supporte pas la déduplication automatique, que peuvent faire l'utilisateur ou l'administrateur pour ne pas dupliquer le contenu de plusieurs fichiers entièrement identiques ? Quelles sont les limites de ce modèle, au niveau de l'application, du contenu et du système de fichiers ?

Si le système ne supporte pas la déduplication automatique, les utilisateurs peuvent créer des liens, ou raccourcis, vers les fichiers. Les limites de ce modèle sont que les fichiers doivent être entièrement identiques, et que c'est géré par l'utilisateur.

concretemnt : **lien symbolique** (ou **symlink**)

En supposant avoir en mémoire tous les hash de tous les blocs disque, quels appels-système doivent être modifiés pour éviter la duplication des blocs et comment ?

Il faut que write recupère le bloc dans lequel on écrit, applique les modifications au bloc puis hashe le bloc. On parcourt ensuite toute la liste des hashs pour voir si un hash correspond à celui que l'on vient de créer. Si le hash n'existe pas, on l'ajoute à la table de hashage et on prend un nouveau bloc dans le disque pour écrire ces données. Si le hash existe déjà, on va modifier les références sur le disque pour que les fichiers partagent le même bloc.

Si le processeur sait hasher un bloc à la moitié du débit mémoire maximum, combien de temps cela prend-il environ ? En reprenant votre réponse à la question précédente, précisez la probabilité que chacun des cas possibles se produise et discutez des surcoûts et gains en temps.

Si le système stocke effectivement les hash en mémoire, que se passe-t-il après un reboot ? Si on envisage de stocker des hashs sur le disque, où serait-il judicieux de les stocker ? Réfléchissez à quand les hashs et les autres éléments du système de fichiers sont lus pour expliquer pourquoi votre solution est intéressante.

Après un reboot on perd tous les hashs, car c'est stocké dans la mémoire. Pour accéder efficacement aux hashs, il serait judicieux de les placer soit au début du disque, soit après les inœuds.

VI. Entrées-sorties

On considère un tube dans lequel un processus écrit pendant qu'un autre lit. Le système d'exploitation utilise un tampon mémoire intermédiaire comme intermédiaire.

S'agit-il d'entrées-sorties logiques ou physiques ? Pourquoi ? Comparez avec d'autres mécanismes de communication inter-processus.

logique car n'affecte aucun appels aux périphériques. Exemple: mmap (lire ou ecrire dans un fichier projeté, généré une I/0 physique)

Comment les données sont-elles transférées concrètement ? Quels accès mémoire dans quels espaces d'adressage sont nécessaires ? Dans quels contextes d'exécution se déroulent-ils ?

quand on écrit dans le tube, le noyau met ce contenu dans la tampon qui est dans la mémoire noyau, en attendant qu'il soit consommé. (ou lu).

Dans un 1er temps, le père est en R et ecrit dans l'espace du noyau (W) et donc dans un 2eme temps, le fils est en W et lit dans l'espace du noyau (R).

=> on est dans l'espace du noyau car on effectue les appels systemes Read/Write qui s'éxécute en mode privilégié.

Imaginez un mécanisme permettant de ne pas utiliser de tampon mémoire intermédiaire et donc de copier directement depuis la mémoire de l'écrivain dans celle du lecteur

On fait un appel à write bloquant tant qu'il n'y a pas de lecteurs et dès que le 2nd processus essaie de lire dans le tube, on copie d'un processus à un autre les données. (puisque c'est ds le noyau)

V. Entrées-sorties (bis)
exo à chier

On considère le sous-système gérant les émissions réseau dans un système d'exploitation. On suppose disposer d'une carte réseau capable d'accéder à la mémoire centrale par DMA à 100Mo/s et d'envoyer sur le câble réseau à la même vitesse

Rappeler rapidement ce qu'est un DMA et comment la carte va l'utiliser pour émettre des données vers le réseau. Comment le pilote peut-il indiquer à la carte quand démarrer l'émission et où se trouvent les données ?

>Direct Memory Access
>***Objectif : Décharger le processeur des transferts de données***
->Le processeur peut faire autre chose
->Le transfert des données peut être plus rapide

>Des périphériques ont régulièrement besoin d'"emprunter de la mémoire" au système afin de s'en servir comme zone de **tampon** (en anglais *buffer*), c'est-à-dire une zone de stockage temporaire permettant d'enregistrer rapidement des données en entrée ou en sortie.

Un canal d'accès direct à la mémoire, appelé **DMA** a ainsi été défini pour y remédier. Cette méthode permet à un périphérique d'emprunter des canaux spéciaux qui lui donnent un accès direct à la mémoire, *sans faire intervenir le microprocesseu*r, afin de le décharger de ces tâches.

II. Anomalie de Belady

On considère un programme qui manipule 5 pages de mémoire virtuelle dans l'ordre suivant: 3 2 1 0 3 2 4 3 2 1 0 4. Ces 5 pages virtuelles sont initialement stockées sur le disque. Elles vont pouvoir être chargées (individuellement) dans les pages de mémoire physique lors d'un accès par le programme, puis éventuellement renvoyées (individuellement) vers le disque plus tard si nécessaire.

L'algorithme FIFO possède l'anomalie de Belady

A la suite d'un défaut de page, le système d'exploitation doit ramener en mémoire la page manquante à partir du disque. S'il n'y a pas de cadres libres en mémoire, il doit retirer une page de la mémoire (la moins récemment utilisee) pour la remplacer par celle demandée. Si la page à retirer a été modifiée depuis son chargement en mémoire, il faut la réécrire sur le disque. Quelle est la page à retirer de manière à minimiser le nombre de défauts de page.

Déroulez l'exécution du programme lorsque la mémoire virtuelle charge les pages en mémoire physique quand elles sont réellement nécessaires, et évince (swappe) les pages sur le disque selon l'algorithme FIFO. A chaque accès mémoire, dites quelle page virtuelle va éventuellement être chargée dans quelle page physique. Combien de défauts de pages vont se produire ?

****FIFO sur 5 pages physiques***

3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	3	3	3 -> **2**	2	2	2	2	**0**	0
	2	2	2	2	2	2 ->**4**	4	4	4	4	4

		1	1	1	1	1	1->3	3	3	3	3
			0	0	0	0	0	2	2	2	2
				3	3	3	3	3	1	1	1

noir => remplissage de la fifo (algo fifo)

vert => le moins récent dégage

11 defaut de page sur 3 pages physique

*fifo sur 3 pages physique

3	2	1	0	3	2	4	3	2	1	0	4
3	3	3	**0**	0	0	**4**	4	4	**1**	1	1
	2	2	2	**3**	3	3	3	3	3	**0**	0
		1	1	1	**2**	2	2	2	2	2	**4**

=> 10 defaut de page

On considère maintenant une machine avec 4 pages physiques au lieu de 3. Déroulez à nouveau l'exécution du programme. Que constate-t-on ? Qu'en pensez-vous ?

à refaire….plus ya de page physique plus le nombre de defaut de page est grand ! (erreur ex1 aussi)

Que dire de l'efficacité de l'algorithme FIFO ? Pourquoi LRU est-il souvent meilleur ? Qu'en est-il de l'algorithme optimal ?

>FIFO pas efficace

>LRU (Least Recently Used) : garde les pages récentes

>Non il n'y a pas d'algo optimal

III.Concurrence et synchronisation

Quelle est la différence entre de la concurrence logique et de la concurrence physique ? Comment les machines multiprocesseurs et/ou multicoeurs influent-elles sur ces deux notions ?

- ➢ concurence logique : plusieurs tâche qui s'éxécutent sur le même processeur et qui avance d'une manière uniforme
- ➢ concurence physique : plusieurs taches s'éxécutant en même temps sur 2 coeur #.

Si pas de multicoeur, ya pas de concurence physique ! il n'y a que de la concurence logique.

Sur une machine monoprocesseur monocoeur, quel événement matériel peut causer de la concurrence ? Cette concurrence est-elle logique ou physique ? Donner un exemple illustré avec une carte réseau. Comment peut-on s'en prémunir ?

Sur une machine monoprocesseur et monocoeur, les interruptions de l'horloge peut peuvent causer de la concurrnence. C'est une concurrence logique.

Exemple : 1er processus lit un paquet réseau et pendant qu'il le déplace vers la mémoire noyau, il reçoit une interruption de l'hrloge. A ce moment, l'ordonnaceur met un autre processus sur le processeur, et ce processus à aussi besoin d'une lecture de donnée réseau.

On peut s'en prémunir grâce aux verrous.

Quelles sont les points communs et différences entre la synchronisation en espace utilisateur et en espace noyau ? Comment les événements matériels influent-ils sur l'espace noyau et utilisateur ? Qu'est-ce qu'un développeur peut faire dans un espace et pas dans l'autre ?

- ➢ points communs : accès aux mêmes ressources , exclusion mutuelle, deadlock / famine
- ➢ différences : s'il y a un pb dans le noyau, c'est plus grave qu'en espace utilisateur; (ex: deadlock), le noyau maitrise la preemption, et les interruption !

Influence des événements :

espace utilisateur : peut pas empecher les événement.

espace noyau : peut empecher les événements en désactivant les interruptions.

Dans l'espace noyau, il faut éviter les verrous (attente passive) et le cout de la synchro est critique.

III. Sémaphores

On s'intéresse à une structure sémaphore et aux fonctions Post() et Wait() qui incrémentent/relâchent et décrémentent/prennent (attente passive) respectivement le compteur correspondant. Le système d'exploitation cible étant multiprocesseur avec ordonnanceur préemptif, on fera attention au verrouillage et aux interruptions.

En vous inspirant du TD2, expliquez comment implémenter cette structure et ces fonctions (en pseudocode).

4.b du TD2

Discutez l'équité de votre implémentation selon la façon dont vous gérez file d'attente des processus endormis. Donnez un exemple concret de problème pouvant se poser

Pb : un seul processus prend toujours la sémaphore