IF-3-SYS : Systèmes d'exploitation 19 juin 2025

NOM Prénom :

Consignes

- L'examen dure 1h30. Prenez le temps de lire le sujet en entier (21 questions sur 8 pages)
- Les réponses seront à inscrire sur le sujet. Commencez par écrire votre nom ci-dessus.
- Écrivez lisiblement et surtout sans ratures. Utilisez un brouillon (vraiment).
- Documents et appareils interdits, sauf une feuille A4 recto-verso manuscrite.
- Pour le binaire et l'hexadécimal, aidez-vous des tableaux page 8.

1 Questions de cours

Sauf mention contraire, chaque question est indépendante. Dans les questions avec plusieurs réponses, les erreurs sont décomptées : ne répondez pas au hasard.

Noyau, processus, appels système

Question 1 Pour chacun des huit éléments de la liste ci-dessous, entourez V si cet élément est typiquement implémenté comme un processus, et entourez F sinon.

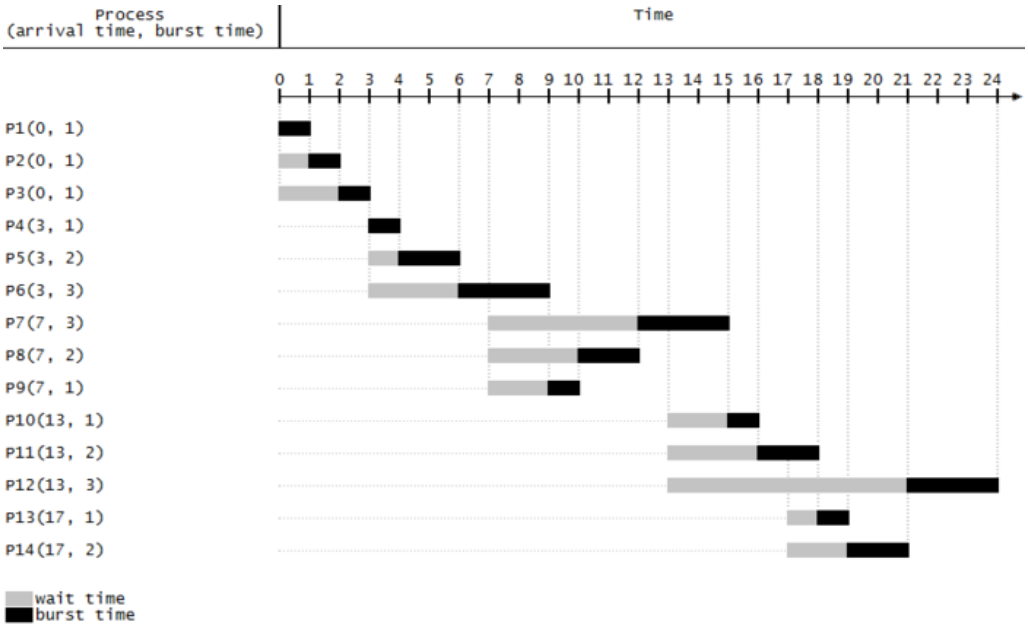
V	F	appel système	V	F	ordonnanceur
V	F	compilateur	V	F	pilote de périphérique
V	F	cycle de von neumann	V	F	shell
V	F	noyau	V	F	système de fichiers

Question 2 Complétez la fonction ci-dessous de telle sorte que l'exécution de `chaine(N)` crée N nouveaux processus, avec une structure de parenté en forme de chaîne. Autrement dit, on veut que chaque processus ait, au maximum, un seul enfant.

```
void chaine(unsigned int N)
{









}
```

Ordonnancement

Question 3 Le diagramme ci-dessous illustre l'exécution d'un ensemble de tâches P1 à P14 sur une machine monoprocesseur : le temps d'attente est représenté en gris, et le temps passé sur le CPU est représenté en noir. Chaque tâche est caractérisée par son instant d'arrivée et par sa durée d'exécution (indiqués entre les parenthèses). Mais quelle est la stratégie appliquée par l'ordonnanceur ? Pour chaque proposition ci-dessous, indiquez si elle vous paraît compatible avec le chronogramme.



V F FCFS
V F SJF
V F SRTF
V F RR

Allocation dynamique

Question 4 Les affirmations ci-dessous se rapportent à la stratégie d'allocation dynamique *first-fit*. Pour chacune d'entre elles, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.

V F Cette stratégie n'a typiquement pas besoin de considérer l'intégralité de la freelist.
V F Cette stratégie n'est jamais victime du problème de fragmentation du tas.
V F Cette stratégie nécessite d'initialiser le tas avec des blocs libres de tailles bien choisies.
V F Cette stratégie cherche à exploiter en priorité les blocs libres les plus petits.

Question 5 Qu'affiche le code ci-dessous ?

```
int x = ~0 ;
printf("%d\n",x) ;
```

$\longrightarrow$ ☐

Concurrence et synchronisation

Question 6 Pour chacune des affirmations ci-dessous, entourez V si elle est correcte, ou entourez F si elle est fausse ou absurde.

V F Une fois qu'un thread est rentré en section critique (en verrouillant un mutex) il devient interdit pour l'ordonnanceur d'attribuer le CPU à un autre thread, et ce jusqu'au déverouillage du mutex.
V F Contrairement aux verrous mutex, les moniteurs sont, par définition, immunisés contre les risques de type *deadlock* (interblocage).
V F Pour qu'une opération puisse être atomique, il faut qu'elle soit réalisée par du code noyau et non pas par du code ordinaire.
V F Les problèmes de concurrence posés par le multitâche sont essentiellement les mêmes, que l'on s'intéresse à des threads ou à des processus.

Question 7 Quelles sont les différentes opérations de synchronisation offertes par une variable de condition ? Pour chaque opération, détaillez-en les paramètres formels et le comportement à l'exécution.

Stockage et systèmes de fichiers

Question 8 Pour chacune des affirmations ci-dessous, entourez V si elle est vraie, ou entourez F si elle est fausse ou absurde.

V F L'appel système `copy()` permet de copier un fichier d'un dossier à un autre.
V F L'approche FAT permet d'accéder rapidement à n'importe quel endroit à l'intérieur d'un fichier.
V F Le format des données stockées dans un fichier est indiqué dans l'i-node correspondant.
V F Sur les systèmes de type Unix, le dossier racine d'un volume s'appelle toujours « / ».

Question 9 On s'intéresse à un système de fichiers basé sur une indexation multi-niveau (à la Unix) avec des blocs de 16kio (cf page 8) identifiés par des numéros de 32 bits. Chaque inode contient 5 pointeurs directs, un pointeur indirect, et un pointeur doublement indirect.
Quelle est la taille maximum d'un fichier sur ce système ? répondez en kio/Mio/Gio/etc, la valeur numérique exacte n'est pas intéressante.

2 Exercice : stratégies de remplacement de pages

Rappel Tous les systèmes d'exploitation modernes implémentent la *mémoire virtuelle*, c'est à dire allouent de l'espace mémoire aux applications sans se limiter à la quantité de mémoire RAM physiquement disponible sur la machine. Au contraire, les processus sont «alloués sur le disque», et seules les données effectivement accédées par le CPU seront copiées en RAM, à la manière d'un cache.

Question 10 On s'intéresse dans cette question la capacité de stockage nécessaire sur le disque. Cette taille, qu'on notera C (en octets), dépend de divers paramètres du système, en particulier :

- le nombre maximum de processus simultanés, qu'on notera N
- le nombre maximum de threads par processus, qu'on notera T,
- la taille de l'espace d'adressage virtuel, qu'on notera V (en octets)
- la quantité de mémoire RAM physiquement disponible, qu'on notera R (en octets)
- le nombre de processeurs de la machine, qu'on notera P.

Donnez une formule pour C en fonction des paramètres qui vous paraissent pertinents.

$C \approx$

Question 11 Le résultat précédent vous paraît-t-il réaliste ? Faites le calcul pour un système à quatre processeurs 32-bits, avec 2Gio de RAM (cf page 8) et supportant au maximum 256 processus, avec 256 threads par processus. Répondez en kio, ou Mio, ou Gio etc. (cf page 8)

Un tel système nécessiterait un disque de capacité $C \approx$ [ ]

Question 12 Comment se fait-il que cette technique (la mémoire virtuelle) soit quand même utilisée en pratique ?

Rappel C'est le noyau qui décide où placer les données des applications. La traduction entre adresses virtuelles et adresses physiques est faite par la MMU, qui consulte pour cela la *table de pagination* du processus actif. Cette table est un dictionnaire indiquant les correspondances entre numéros de pages virtuelles (VPN) et numéros de pages physiques (PPN).

Question 13 Soit la table de pagination suivante (le symbole ∅ indique les entrées invalides) :

VPN	0	1	2	3	4	5	6	7	8
PPN	5	∅	7	1	8	2	∅	0	∅

On suppose que la taille de page est de 2kio (2048 octets). Pour les adresses virtuelles 1A42 et 076F, donnez ci-dessous les adresses physiques correspondantes (en hexadécimal), ou répondez par ∅ si la traduction est impossible.

[ ] et [ ]

Rappel La *pagination à la demande* consiste à gérer la RAM comme un cache. Lorsque le CPU accède à une donnée existante mais non chargée en RAM, la MMU lève une exception, et le noyau réagit à ce *défaut de page* en chargeant la page demandée depuis le disque. À cause des entrées-sorties nécessaires, ce va-et-vient (*swap*) prend du temps, et peut affecter négativement les performances.

Question 14 Supposons que le processeur met normalement 1 nanoseconde pour exécuter une instruction, et qu'un défaut de page entraîne un délai supplémentaire de p nanosecondes. Si le programme cause en moyenne un défaut de page toutes les k instructions, quelle sera la durée moyenne effective d'une instruction ?

[ ]

Question 15 Dans cette question, on considère un centre de calcul hébergeant un grand nombre de serveurs. Les ingénieurs système ont entendu parler d'une nouvelle technique permettant d'implémenter le va-et-vient non pas vers le disque dur de la machine locale, mais vers la mémoire vive d'un serveur voisin. Cette approche vous paraît-elle techniquement possible ? Si oui, expliquer comment on pourrait la mettre en oeuvre ; si non, expliquer pourquoi ce n'est pas faisable. Dans tous les cas, sous quelles conditions une telle approche serait-elle rentable ? Donnez des ordres de grandeur pour les différents paramètres impliqués.

Remplacement de pages Au fur et à mesure de l'exécution, de plus en plus de pages virtuelles sont chargées en mémoire. Après un temps suffisant, toute la mémoire physique est occupée. Pour traiter un nouveau défaut de page, l'OS devra alors libérer une page physique, c'est à dire *évincer* la page virtuelle qui s'y trouvait en la copiant vers le disque. Dans la suite de ce sujet, on va s'intéresser à différentes façons de choisir quelle page évincer, c'est à dire aux différentes *stratégies de remplacement de pages*. Pour simplifier, on se restreint au cas d'un unique processus.

Exemple : stratégie FIFO Soit une machine avec 4 pages de mémoire physique, initialement libres. Supposons que le programme accède d'abord à la page virtuelle n° 1, puis à la page n° 5 etc, comme indiqué dans la *séquence d'accès* suivante : 1, 5, 2, 5, 0, 4, 1, 7.
Le diagramme ci-dessous représente l'état du système au cours de l'exécution : la première ligne reprend la séquence d'accès, et les lignes suivantes indiquent le contenu de la mémoire physique. À l'instant initial (non représenté) toutes les pages physiques sont libres. Lorsque le programme accède à la page n° 1, le noyau charge celle-ci à la première place libre. Le programme fait peut-être plusieurs accès successifs dans cette même page, mais les détails ne nous intéressent par car ils ne changent rien en termes de stratégies de remplacement.
Au bout d'un moment, le programme accède à une adresse située dans la page n° 5, donc le noyau doit la charger en RAM, et de même pour la page n° 2 ensuite. Par convention, on ne représentera dans notre tableau que les défauts de page, et on laissera en blanc les cases qui ne changent pas. Ainsi la quatrième colonne est complètement vide : à ce moment de l'exécution, la page n° 5 est déjà présente en mémoire et donc le programme peut y accéder sans causer de défaut.

1	5	2	5	0	4	1	7
1					4		
	5					1	
		2					7
				0			

Lors de l'accès à la page n° 4, par contre, le système devra d'abord évincer l'une des pages présentes pour faire de la place. La stratégie de remplacement FIFO (pour *First In First Out*) consiste à systématiquement choisir la page virtuelle qui est présente en RAM depuis le plus longtemps. Dans notre exemple, l'OS choisit donc la page n° 1.

Peu après, le programme accède de nouveau à la page n° 1, causant de nouveau un défaut. L'OS choisit alors d'évincer la page n° 5, et ainsi de suite.

Question 16 Dans le diagramme ci-dessous, représentez le comportement de la stratégie FIFO pour la séquence d'accès suivante : 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
FIFO																				

Question 17 On voit bien que la stratégie FIFO est un peu simpliste et fait souvent de «mauvais choix». À l'inverse, le «meilleur choix» consisterait à toujours évincer de la mémoire la page qui va rester inutilisée pour la plus grande période de temps. Mais il faudrait pour cela savoir prédire le futur ! Cette approche n'est donc pas implémentable en pratique. Pourtant, elle garde un intérêt théorique : en particulier, elle permet de déterminer le nombre *optimal* de défauts de pages pour une certaine séquence d'accès.
Dans le diagramme ci-dessous, représentez le comportement de la stratégie optimale OPT sur notre séquence d'accès.

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
OPT																				

Question 18 Les stratégies implémentées dans les vrais systèmes sont souvent des approximations de OPT. Par exemple, la stratégie LRU (pour *Least Recently Used*) fait la supposition que le futur proche va ressembler au passé récent. Une page accédée récemment a de bonnes chances d'être accédée de nouveau prochainement. À l'inverse, une page qui n'a pas été touchée depuis longtemps a de bonnes chances de rester inutilisée pendant encore longtemps. Se basant sur cette observation, la stratégie LRU consiste à toujours choisir d'évincer la page qui n'a pas été accédée depuis le plus longtemps.
Dans le diagramme ci-dessous, représentez le comportement de la stratégie LRU sur notre séquence d'accès.

	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
LRU																				

Question 19 La stratégie LRU promet de meilleures performances que FIFO mais elle est bien plus complexe à mettre en oeuvre. En quoi consiste la difficulté principale ? Donnez des ordres de grandeur réalistes pour les différents paramètres impliqués.

Question 20 Dans les exemples ci-dessus, on a étudié le comportement de diverses stratégies sur une séquence bien précise. Supposons maintenant une séquence d'accès quelconque de longueur n, comportant p numéros de page distincts. La machine dispose de k pages de mémoire physique. Indépendamment de la stratégie adoptée, on peut borner le nombre f de défauts de pages qui se produiront pendant l'exécution. Donnez ci-dessous un minorant et un majorant de f, en fonction de ces paramètres.

$$\boxed{} \leqslant f \leqslant \boxed{}$$

Question 21 On voudrait tester les performances de nos stratégies de remplacement de pages sur davantage d'exemples, mais le temps manque pour recueillir de vraies traces mémoire. Un chef de projet propose de générer rapidement de nombreuses séquences grâce à un LLM, en lui faisant simplement tirer des numéros de page au hasard. Que lui répondez-vous ?

Annexe : aide pour les calculs en binaire

Les premiers nombres entiers, notés en décimal, en hexadécimal, et en binaire :

Dec	Hex	Bin
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100

Dec	Hex	Bin
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001

Dec	Hex	Bin
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110

Dec	Hex	Bin
15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011

Les premières puissances de 2 :

$2^0 = 1$	$2^{16} = 65\,536$	$2^{32} = 4\,294\,967\,296$	$2^{48} = 281\,474\,976\,710\,656$
$2^1 = 2$	$2^{17} = 131\,072$	$2^{33} = 8\,589\,934\,592$	$2^{49} = 562\,949\,953\,421\,312$
$2^2 = 4$	$2^{18} = 262\,144$	$2^{34} = 17\,179\,869\,184$	$2^{50} = 1\,125\,899\,906\,842\,624$
$2^3 = 8$	$2^{19} = 524\,288$	$2^{35} = 34\,359\,738\,368$	$2^{51} = 2\,251\,799\,813\,685\,248$
$2^4 = 16$	$2^{20} = 1\,048\,576$	$2^{36} = 68\,719\,476\,736$	$2^{52} = 4\,503\,599\,627\,370\,496$
$2^5 = 32$	$2^{21} = 2\,097\,152$	$2^{37} = 137\,438\,953\,472$	$2^{53} = 9\,007\,199\,254\,740\,992$
$2^6 = 64$	$2^{22} = 4\,194\,304$	$2^{38} = 274\,877\,906\,944$	$2^{54} = 18\,014\,398\,509\,481\,984$
$2^7 = 128$	$2^{23} = 8\,388\,608$	$2^{39} = 549\,755\,813\,888$	$2^{55} = 36\,028\,797\,018\,963\,968$
$2^8 = 256$	$2^{24} = 16\,777\,216$	$2^{40} = 1\,099\,511\,627\,776$	$2^{56} = 72\,057\,594\,037\,927\,936$
$2^9 = 512$	$2^{25} = 33\,554\,432$	$2^{41} = 2\,199\,023\,255\,552$	$2^{57} = 144\,115\,188\,075\,855\,488$
$2^{10} = 1024$	$2^{26} = 67\,108\,864$	$2^{42} = 4\,398\,046\,511\,104$	$2^{58} = 288\,230\,376\,151\,711\,744$
$2^{11} = 2048$	$2^{27} = 134\,217\,728$	$2^{43} = 8\,796\,093\,022\,208$	$2^{59} = 576\,460\,752\,303\,423\,488$
$2^{12} = 4\,096$	$2^{28} = 268\,435\,456$	$2^{44} = 17\,592\,186\,044\,416$	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
$2^{13} = 8\,192$	$2^{29} = 536\,870\,912$	$2^{45} = 35\,184\,372\,088\,832$	$2^{61} = 2\,305\,843\,009\,213\,693\,952$
$2^{14} = 16\,384$	$2^{30} = 1\,073\,741\,824$	$2^{46} = 70\,368\,744\,177\,664$	$2^{62} = 4\,611\,686\,018\,427\,387\,904$
$2^{15} = 32\,768$	$2^{31} = 2\,147\,483\,648$	$2^{47} = 140\,737\,488\,355\,328$	$2^{63} = 9\,223\,372\,036\,854\,775\,808$
			$2^{64} = 18\,446\,744\,073\,709\,551\,616$

On rappelle également que :

— 1 kio (un kilooctet) = 1024 octets,
— 1 Mio (un mégaoctet) = 1024 kio,
— 1 Gio (un gigaoctet)= 1024 Mio,
— 1 Tio (un téraoctet)= 1024 Gio,
— et ainsi de suite, avec dans l'ordre : Pio (pétaoctet), Eio (exaoctet), Zio (zettaoctet), Yio (yottaoctet)

(Vous avez aussi le droit d'utiliser les «préfixes binaires» kibi, mébi, gibi, etc si vous préférez)
En cas de doute, n'hésitez pas à demander des précisions.