IF-3-SYS : Examen du 31 mai 2022

Q1. First Come First Serve, Interrupt Service Routine, Memory Management Unit, Process IDdentifier

Q2. Faux (exec cause un changement d'exécutable dans le même processus) ; Vrai (le shell fait des `read()` sur son entrée standard) ; Vrai (par définition du noyau) ; Faux (même pas le droit de la consulter)

Q3. exec, exit, pthread_exit

Q4. 10 lignes en tout, avec 8 numéros distincts

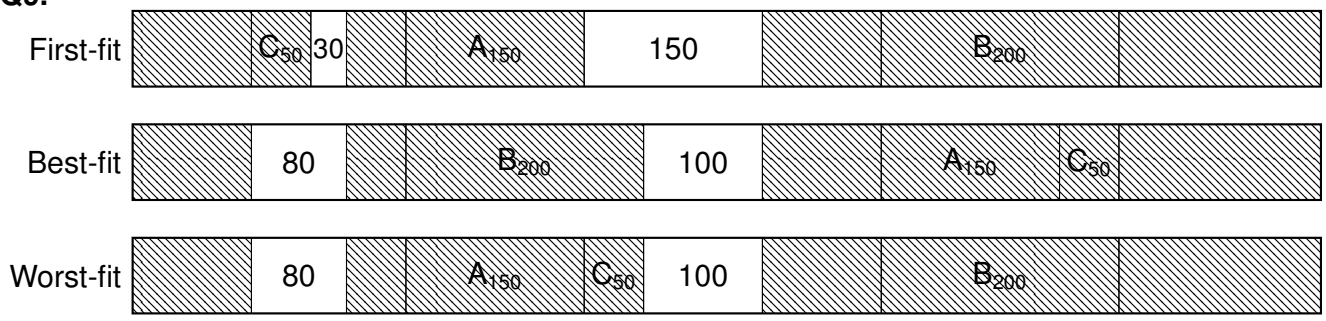
Q5. deux mécanismes nécessaires :
- des interruptions matérielles ; avec un timer matériel on est sûr d'en avoir régulièrement
- la préemption, i.e. la capacité de suspendre une tâche en cours d'exécution, même si celle-ci n'est pas d'accord.

Q6.
```
time 0  60  90 110 150
 CPU | A | C | D | B |
```

Q7. Faux ; Vrai ;Vrai ;Vrai (on l'a fait au TP3)

Q8. On remarque que chaque traduction conserve 12 bits d'offset, on en déduit que la taille des pages est 2^{12} = 4 kio.
Les adresses sont exprimées sur 16 bits, donc la taille max d'un VAS est 2^{16} = 64 kio.
Pour la taille de RAM on ne peut pas conclure : le PAS fait visiblement 64 kio aussi, mais on ne peut pas savoir quelle proportion correspond à de la RAM.

Q9.



Q10. on peut faire par exemple `return (value>>pos)%2` ou encore `return (value>>pos)&1`

Q11. Faux (par définition) ; Vrai (par définition aussi) ; Faux (pas nécessairement) ; Faux (pas nécessairement)

Q12. cd, mkdir, rmdir, ls, tar, cp, mv...

Q13. Un chemin absolu commence par un « / », sinon c'est un chemin relatif.
Exemple absolu : /home/gsalagnac/doc.txt
Exemples relatifs : doc.txt, ./doc.txt, ../doc.txt, dir/doc.txt, dir, ./dir

Q14. 1 Tio = 2^{40} octets et 1 secteur = 2^{12} octets, donc il y a $2^{42} \div 2^{12} = 2^{42-12} = 2^{30}$ secteurs $\Rightarrow$ 30 bits

Q15. il suffit d'un mutex : Init : Semaphore S=1
A,C : P(S)
B,D : V(S)

Q16. Voilà *une* solution qui marche (cf little book of semaphores readers-writers). Rien de fracassant : on garde une mutex côté les rédacteurs, et côté lecteurs on se débrouille pour que ladite mutex soit verrouillée tant qu'il y a un ou plusieurs lecteurs dans leur section critique.

init : Semaphore mutexNBReaders=1, variable entière nbReaders=0, Semaphore libre=1

A : P(mutexNBReaders)
    if(nbReaders == 0) *// c'est que je suis le premier lecteur à rentrer*
        then P(libre) *// marquer la ressource comme non-libre pour bloquer les rédacteurs*
    end if
    nbReaders += 1
    V(mutexNBReaders)

B : P(mutexNBReaders)
    nbReaders –= 1
    if(nbReaders == 0) *// c'est que je suis le dernier lecteur à repartir*
        then V(libre) *// marquer la ressource comme libre pour débloquer les rédacteurs*
    end if
    V(mutexNBReaders)

Pour les rédacteurs c'est beaucoup plus simple puisqu'on parle toujours d'une pure mutex :
C : P(libre) *// prendre le mutex*
    `modifier()`
D : V(libre) *//rendre le mutex*

Q17. Il est possible pour les lecteurs de monopoliser la ressource, ce qui empêche indéfiniment les rédacteurs de travailler.

Q18. Une possibilité est de rajouter un troisième sémaphore que le rédacteur peut prendre pour empêcher des lecteurs supplémentaires de rentrer :

init : Sem mutexNBReaders=1, variable nbReaders=0, Sem libre=1, Sem toto=1

A : P(toto) *// vérifier qu'on a l'autorisation de rentrer*
    V(toto)
    P(mutexNBReaders)
    ... *// la suite est identique à la question précédente*

Et côté rédacteurs, ça donne :
C : P(toto) *//empêcher de nouveaux lecteurs de rentrer*
    P(libre) *// prendre le mutex*
    `modifier()`
D : V(libre) *//rendre le mutex*
    V(toto)

Downey, qui a une solution similaire, parle de "turnstile" qui se traduit par portillon.