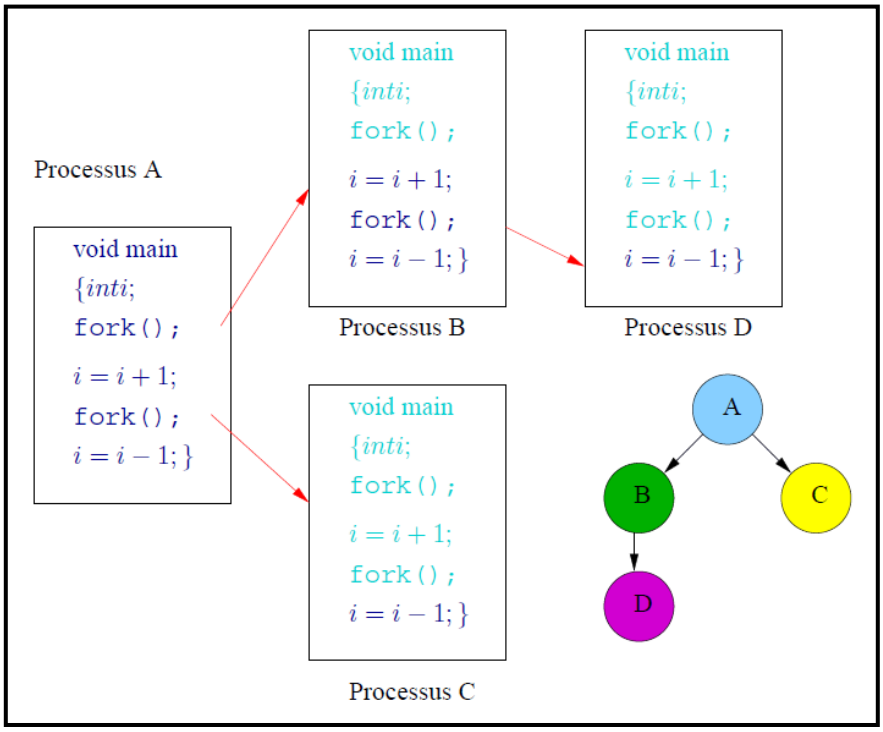
Semaine 2 – Questions sur la gestion des processus (solutions)

1. Dessinez l'arbre d'exécution de ce bout de code :

Processus A

```
void main
{inti;
fork();
i = i + 1;
fork();
i = i − 1;}
```

Réponse :



2. Un processus peut passer par les états (Prêt, Élu, Bloqué) dans l'ordre suivant :
   a. Élu -> Bloqué -> Prêt
   b. Bloqué -> Élu -> Prêt
   c. Élu -> Prêt -> Bloqué

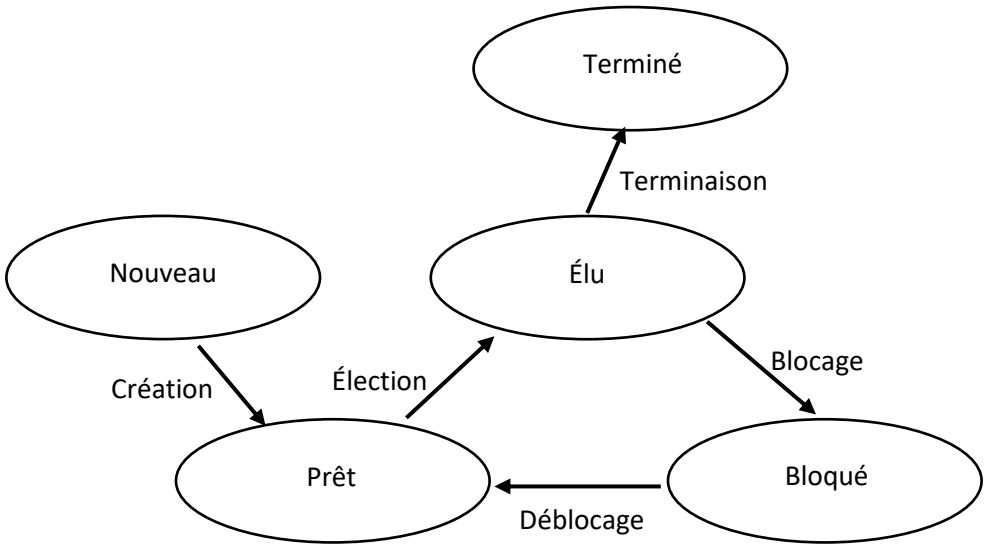
Réponse : Il existe 5 états : nouveau, prêt, élu, bloqué et terminé. Lorsqu'un processus est en train de s'exécuter (qu'il utilise le microprocesseur), on dit que le processus est dans l'état "élu".

Un processus qui se trouve dans l'état élu peut demander à accéder à une ressource pas forcément disponible instantanément (par exemple lire une donnée sur le disque dur). Le processus ne peut pas poursuivre son exécution tant qu'il n'a pas obtenu cette ressource. En attendant de recevoir cette ressource, il passe de l'état "élu" à l'état "bloqué".

Lorsque le processus finit par obtenir la ressource attendue, celui-ci peut potentiellement reprendre son exécution. Or étant donné que les systèmes d'exploitation permettent de gérer plusieurs processus "en même temps", mais un seul processus peut se trouver dans un état "élu" (le microprocesseur ne peut "s'occuper" que d'un seul processus à la fois). Quand un processus passe d'un état "élu" à un état "bloqué", un autre processus peut alors "prendre sa place" et passer dans l'état "élu". Le processus qui vient de recevoir la ressource attendue ne va donc pas forcément pouvoir reprendre son exécution tout de suite, car pendant qu'il était dans à état "bloqué" un autre processus a "pris sa place". Un processus qui quitte l'état bloqué ne repasse pas forcément à l'état "élu", il peut, en attendant que "la place se libère" passer dans l'état "prêt" (sous-entendu "j'ai obtenu ce que j'attendais, je suis prêt à reprendre mon exécution dès que la "place sera libérée"").

Le passage de l'état "prêt" vers l'état "élu" constitue l'opération "d'élection". Le passage de l'état élu vers l'état bloqué est l'opération de "blocage". Un nouveau processus est toujours créé dans l'état "prêt". Pour se terminer, un processus doit obligatoirement se trouver dans l'état "élu".

On peut résumer tout cela avec le diagramme suivant :



Donc, on constante que seul l'ordre Élu -> Bloqué -> Prêt est possible.

3. Lorsqu'un processus enfant se termine avant la fermeture du processus parent, lequel des éléments suivants est vrai ?
    a. Le processus fils devient orphelin
    b. Le processus parent disparaît
    c. Si le processus parent ne gère pas SIGCHLD, le processus fils devient un zombie
    d. Aucune de ces réponses

    Réponse : Lorsqu'un processus enfant s'arrête ou se termine, SIGCHLD est envoyé au processus parent. La réponse par défaut au signal est de l'ignorer. Le signal peut être intercepté et le statut de sortie du processus fils peut être obtenu en appelant immédiatement wait (2) et wait3 (3C). Cela permet de supprimer les entrées de processus zombie aussi rapidement que possible.

4. Un utilisateur émet la séquence de commandes suivantes:

```
1. $ c.out
2. $ bash
3. $ c.out
```

    Si l'utilisateur tue le processus bash, alors lequel des énoncés suivants est vrai?
    a. Le deuxième processus c.out est également terminé
    b. Le deuxième processus c.out devient un processus orphelin
    c. Le premier processus c.out devient un processus zombie
    d. Le processus init devient parent du second processus c.out

    Réponse : Lorsqu'un processus meurt, ses enfants sont adoptés par init, vous verrez donc beaucoup de processus dont le parent est 1 sur un système typique.

5. Qui est chargé en mémoire au démarrage du système?
    a. Noyau
    b. Shell
    c. Commandes
    d. Script

    Réponse : Le noyau (mais pas ses modules) sera chargé en mémoire. S'il y a des modules dont le noyau aura besoin avant que les systèmes de fichiers soient disponibles (cela signifie généralement les pilotes et les périphériques), ces modules seront dans initramfs (en mémoire) et le noyau les chargera à partir de là. D'autres modules peuvent être chargés ultérieurement à partir du système de fichiers.

6. Le processus de démarrage d'un ordinateur est connu sous le nom de _____ ?
    a. Boot Loading
    b. Boot Record
    c. Boot Strapping
    d. Booting

    Réponse : Boot strapping fait référence à un processus de démarrage automatique, et fait référence au processus de chargement du logiciel de base dans la mémoire d'un

ordinateur après la mise sous tension ou la réinitialisation générale, en particulier le système d'exploitation qui charge les autres logiciels si nécessaire.

7. L'identifiant du processus « init » est ____ ?
    a. 2
    b. 1
    c. 0
    d. -1

    Réponse : L'identifiant du processus « init » est 1.

8. À la fin du démarrage du noyau, quel processus est lancé?
    a. /etc/init
    b. /etc/sched
    c. /etc/swap
    d. /etc/kernel

    Réponse : Le Kernel ou le noyau est la partie du système d'exploitation qui se charge en premier, et elle reste dans la mémoire principale. Généralement, le Kernel est responsable de la gestion mémoire, de la gestion des processus et de la gestion des disques. Le Kernel connecte le matériel du système au logiciel d'application. Chaque système d'exploitation a un Kernel.

    init est le parent de tous les processus sur le système, il est exécuté par le noyau(kernel) et est responsable du démarrage de tous les autres processus; il est le parent de tous les processus dont les parents naturels sont morts et il est responsable de les récolter lorsqu'ils meurent. Les processus gérés par init sont appelés jobs et sont définis par des fichiers dans le répertoire /etc/init.

9. Laquelle des valeurs suivantes pour la colonne STAT (statut) de la commande « ps » n'est pas vraie?
    a. Statut R signifie Running
    b. Statut S signifie Sleeping
    c. Statut E signifie Exited
    d. Statut Z signifie Zombie

    Réponse : R, S, Z, etc. Ce sont les états du processus. Les états de processus indiqués par ps sont les suivants:

    D : Sommeil sans interruption (généralement une IO)

    R : en cours d'exécution ou exécutable (dans la file d'attente d'exécution)

    S : sommeil interruptible (en attente d'un événement à compléter)

    T : Arrêté, soit par un signal de contrôle du Job, soit parce qu'il est en train d'être tracé.

    W : paging

X : mort (ne devrait jamais être vu)

Z : Processus (« zombie »), s'est terminé sans que le processus parent ne soit mis au courant.

10. Un processus
    a. contient des instructions
    b. peut engendrer des sous-processus
    c. est une "tâche"
    d. les 3 dernières réponses
    e. aucune des 4 dernières réponses

11. Définir la notion de PCB.

    Réponse : PCB (Process Control Block) est une structure de données associée à un processus et contenant toute l'information décrivant le contexte du processus => bloc de contrôle.

12. Donner quatre attributs parmi ceux qui constituent le PCB.

    Réponse : PID, PPID, État, Priorité, Compteur Ordinal, Fichiers ouverts, pointeurs (seg de code, seg de données, seg de pile, ..), temps d'exécution...

13. Expliquez pourquoi, dans UNIX, lorsqu'un processus exécute l'appel système « exit », ses ressources ne sont pas libérées tout de suite.

    Réponse : Le processus passe à l'état zombie et reste dans cet état jusqu'à ce que le père exécute un « wait (&vp) » et récupère dans vp la valeur du paramètre de retour de exit ainsi que d'autres informations concernant la terminaison du fils comme par exemple fin normale ou anormale.

14. Expliquez pourquoi les processeurs ont deux modes de fonctionnement (noyau et utilisateur).

    Réponse : Pour protéger le système d'exploitation contre les intrusions et les erreurs des autres. Les instructions privilégiées du système d'exploitation s'exécutent en mode noyau. Le mode utilisateur permet d'isoler les processus utilisateur et de fournir une meilleure protection aux processus du noyau face à ces processus utilisateur.

15. Dans le système UNIX, est-ce que tout processus a un père ? Que se passe-t-il lorsqu'un processus devient orphelin (mort de son père) ? Quand est-ce qu'un processus passe à l'état Zombie ?

    Réponse : Oui à l'exception du processus INIT. Le processus INIT devient son père. Un processus devient Zombie lorsqu'il effectue l'appel système exit et envoie donc un signal à son père puis se met en attente que le père ait reçu le signal.

16. Que fait chacun des programmes suivants?

A.

```
int main( )
{
        int p=1 ;
        while(p>0) p=fork() ;
        execlp("prog", "prog", NULL) ;
        return 0 ;
}
```

Réponse : Le père crée des processus fils tant qu'il n'y a pas d'échec. Le père et les processus créées se transforment en prog.

B.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int i = 2;

int i=2 ;
int main()
{
   j=10;
   int p;
   while(i-- && p = fork())
   if(p<0) exit(1);
   j += 2;
   if (p == 0)
   {     i *= 3;
         j *= 3;
   }
   else
   {
         i *= 2;
         j *= 2;
   }
   printf(« i=%d, j=%d », i,j);
   return 0;
}
```

Réponse : Le processus père crée un fils et rentre dans la boucle. La valeur de i du père et du fils est de 1. Le fils exécute la boucle et crée un processus fils. Il sort ensuite de la boucle, car l'expression (--i) devient égale à 0. Le processus principal exécute j+=2; i=2; j=2; et enfin, il affiche les valeurs 0 et 24. Le fils du processus principal effectue la condition du while et i tombe à 0. Il affiche alors 0, 36. Le fils restant effectue la condition du while. Comme le langage C évalue les conditions de droite à gauche et que sa valeur de p est de 0, ce processus sort de la boucle avec une valeur de i égale à 1 et affiche alors 3 et 36.

C.

```
int i=4,            j=10;
int main ( )
{
   int p ;
   p = fork();
   if(p<0) exit(1) ;
   j += 2;
   if (p == 0)
   {          i *= 3;
              j *= 3;
 }
 else
 {
              i *= 2;
              j *= 2;
 }
printf("i=%d, j=%d", i,j) ;
return 0 ;
}
```

Réponse : Le père tente de créer un fils. S'il ne parvient pas, il se termine. Sinon, il affiche i=8, j=24. Le fils affiche i=12, j=36.

D.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int i = 2;

int main() {

	int j = 10;
	int p;
	while(--i && p = fork())
		if (p<0) exit(1);
	j+=2;
	if (p==0){
		i*=3;
		j*=3;
	}
	else
	{
		i*=2;
		j*=2;

	}
	printf("i=%d, j=%d\n", i, j);
	return 0;
}
```

Réponse : Le processus père tente de créer un fils et rentre dans la boucle. Si la création échoue, le processus père se termine (p<0). Sinon, il sort de la boucle car l'expression (--i) devient égale à 0. Il exécute ensuite j+=2; i*=2; j*=2; et enfin, il affiche les valeurs 0 et 24.

Le fils ne rentre pas dans la boucle car i=1 mais p=0. Il exécute j+=2; i*=3; j*=3; et enfin, il affiche les valeurs 3 et 36.