Note **15,50** sur 20,00 (**78**%)

Question **1**

Terminer

Non noté

Directives :

1. Cet examen est composé de 12 questions pour une durée totale de 2 heures.
2. Pondération 35%.
3. Aucune réponse aux questions durant l'examen. En cas de doute sur la compréhension de l'énoncé d'une question, énoncez clairement dans votre réponse toutes vos suppositions. Vous pouvez également utiliser cette page pour énoncer clairement vos suppositions. N'oubliez pas d'indiquer le numéro de la question. Nous tiendrons compte de toute supposition/interprétation sensée.
4. Tous les appels système utilisés dans les questions sont supposés exempts d'erreurs et considèrent leurs options par défaut.
5. Il n'est pas demandé de traiter les cas d'erreurs, ni d'inclure les directives d'inclusion dans les codes à compléter.
6. Pour les questions à choix multiples, **vous devez sélectionner une seule réponse**.
7. Lisez au complet chaque question avant d'y répondre.

Question **3**

Terminer

Note de 0,00 sur 1,00

Quel énoncé est vrai pour un système monoprocesseur de traitement par lots monoprogrammé (sans multiprogrammation) ?

- Aucune de ces réponses.
- Avec un tel système, il est possible d'exécuter un processus pendant qu'un autre est bloqué en attente d'une entrée/sortie.
- Un processus en attente active ne va pas s'accaparer le processeur.
- Chaque processus s'exécute pendant un temps prédéterminé par le système avant de passer à l'état bloqué pour un certain temps.
- Chaque processus s'exécute pendant au maximum un temps prédéterminé par le système avant la suspension de son exécution au profit d'un autre processus.
- Chaque processus s'exécute complètement avant de lancer l'exécution d'un autre.

Votre réponse est incorrecte.

La réponse correcte est : Chaque processus s'exécute complètement avant de lancer l'exécution d'un autre.

Question **4**

Terminer

Note de 0,00 sur 1,00

Un processus crée un processus fils via l'appel système fork. Indiquez parmi les éléments suivants ceux qui sont des copies identiques, à la création du fils.

- (●) Les signaux en attente.
- ( ) Le processus parent.
- ( ) Le numéro d'identification du processus.
- ( ) Aucune de ces réponses.
- ( ) La valeur du compteur ordinal.
- ( ) La valeur de retour de l'appel à fork.

Votre réponse est incorrecte.

La réponse correcte est : La valeur du compteur ordinal.

Question **5**

Terminer

Note de 1,00 sur 1,00

Considérez le code suivant :

```
char A[3]= "CBA";
int main() {
        if (fork()==0)  { // F1
              printf("%c \n", A[0]);
              exit(0);
        }
        printf("%c", A[0]);
        if (fork()==0)  { // F2
               printf("%c \n", A[1]);
               exit(0);
        }
   printf("%c", A[1]);
   printf("%c \n",A[2]);
   wait(NULL); wait(NULL);
   return 0;
}
```

Donnez les lettres affichées à l'écran par chacun des processus créés (F1 et F2) et le processus principal (PP).

- aucune de ces réponses.
- PP: CBA F1: C F2: CB
- PP: CBA F1: C F2: B
- PP: CBA F1: CB F2: CB

Votre réponse est correcte.

La réponse correcte est :
PP: CBA F1: C F2: CB

Question **6**

Terminer

Note de 0,00 sur 1,00

dup2(1, fd=open("fich", O_WRONLY)); fait perdre, au processus, l'accès au fichier fich via le descripteur fd.

Sélectionnez une réponse :

- Vrai
- Faux

La réponse correcte est « Vrai ».

Question **7**

Terminer

Note de 1,00 sur 1,00

Considérez les processus A, B et C suivants, où a1; a2, b, c1 et c2 sont des actions atomiques :

Sémaphore x = 1, y=1;

A	B	C
P(x);	P(x);	P(y);
a1;	P(y);	P(x);
V(x);	b;	c1;
a2;	V(x);	V(x);
V(y);	V(y);	c2;

Sélectionnez les scénarios possibles d'exécution des actions de ces processus, sachant qu'ils s'exécutent en concurrence.

- aucune de ces réponses.
- c1; b; c2; a1; a2;
- c1; a1; a2; c2; b;
- a1; c1; c2; b; a2;
- c1; c2; b; a1; a2;

Votre réponse est correcte.

La réponse correcte est :
c1; a1; a2; c2; b;

Question **8**

Terminer

Note de 1,50 sur 1,50

Un processus exécute le code suivant :

```
int x=0;
int main() {
x=x+1;
 if(fork()>0) {
        x=x+2;
 }   else {
            x=x+2;
            exit(0);
 }
  wait(NULL);
  printf("x=%d\n",x);
  return 0;
 }
```

La valeur de x affichée par « printf » est :

- 5.
- 3.
- 3 ou 5.
- aucune de ces réponses.

Votre réponse est correcte.

La réponse correcte est :
3.

Question **9**

Terminer

Non noté

Justifiez votre choix de réponse (si votre justification est absente ou erronée vous aurez **0 point** pour cette question que votre réponse dans le QCM soit bonne ou mauvaise). Pour la justification, il suffit de lister la séquence d'instructions exécutées par le processus qui affiche la valeur de x.

Au début x = 0

Après x = 1

Après on fork, puis pour le prère x = 3

Dans le else (donc pour le fils) x = 3 (aussi) mais après le exit met fin au processus du fils

Après le père attend la fin du fils

Après le père comprend que le fils a exit(0) donc plus besoin de l'attendre

Après le père affiche sa valeur de x soit 3

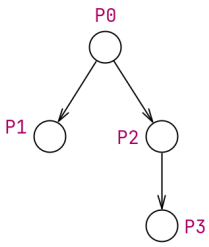
Question **10**

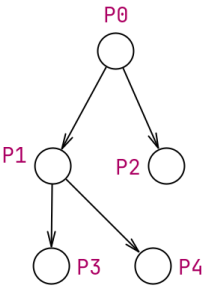
Terminer

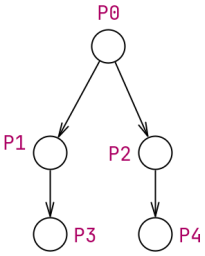
Note de 2,50 sur 2,50

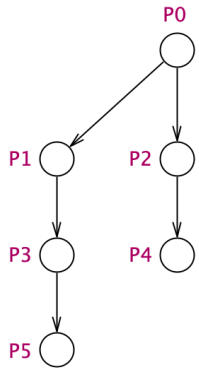
Quel arbre représente la hiérarchie de processus générés par la fonction suivante (où P0 est le processus exécutant la fonction construireArbre) :

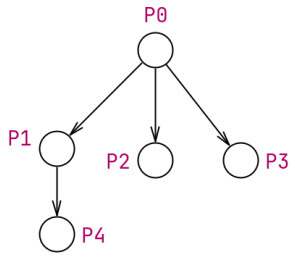
```
void construireArbre() {
    for(int i=0; i<3; i++) {
        if(fork()) {
            if(i>0) break;
        } else {
            fork();  break;
        }
    }
    while(wait(NULL)>0);
    _exit(0);
}
```





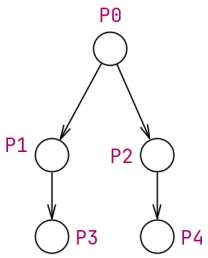




- Aucune de ces réponses
- 

Votre réponse est correcte.

La réponse correcte est :



Question **11**

Terminer

Non noté

Justifiez votre choix de réponse (si votre justification est absente ou erronée vous aurez **0 point** pour cette question que votre réponse dans le QCM soit bonne ou mauvaise). Pour la justification, il suffit de lister la séquence d'instructions exécutées par chaque processus (y compris le processus principal).

Début de la boucle, on devrait itérer 3 fois en tout (i = 0)

On entre dans la boucle

Fork dans la condition du if, le P0 créer un fils P1

Le père ne break pas pcq c'est faut que i > 0

Le fils P1 fork à son tour (il créer P2) dans le else, puis break, donc sort de la boucle

P2 aussi break donc fin de P2

Deuxième itération (concerne seulement le père P0)

On entre dans la boucle

Fork dans la condition du if, le P0 créer un fils P3

Le père (P0) ne break pas pcq que i > 0 (i égale 1)

Le fils P3 fork à son tour (il créer P4) dans le else, puis break, donc sort de la boucle

P4 aussi break donc fin de P4

NOTE importante, j'ai numéroté les fils de 0 à N en ordre de création.
Donc dans mon raisonnement il faut inversé P2 et P3 pour que ça corresponde exactement au dessin que j'ai sélectionné (le graphe reste topologiquement identique, seul les nom des noeuds P2 et P3 sont inversés !)

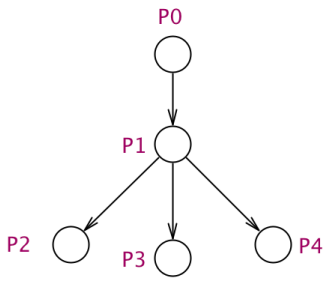
Question **12**

Terminer

Note de 2,50 sur 3,00

```
int main() {
/*0*/
/*1*/
/*2*/
/*3*/
/*4*/
/*5*/
/*6*/
/*7*/
/*8*/
/*9*/
/*10*/
/*11*/
/*12*/
/*13*/
/*14*/ _exit(0);
}
```

Complétez le code ci-dessus pour créer l'arbre de processus décrit par la figure suivante (où P0 est le processus principal). Chaque processus parent doit attendre la fin de ses processus fils juste avant de se terminer. Chaque processus sans enfant doit se transformer pour exécuter la commande : echo <NOM> où <NOM> est le nom du processus (P2, P3 ou P4).



Pour répondre à cette question, sélectionnez les instructions à insérer aux bons endroits. Sélectionnez "rien" s'il n'y a aucune instruction à insérer dans cette ligne.

/*0*/	if (fork()==0) {
/*1*/	if (fork()==0) {
/*2*/	execlp("echo", "echo", "P2",NULL);
/*3*/	_exit(0);
/*4*/	}
/*5*/	if (fork()==0) {
/*6*/	execlp("echo", "echo", "P3",NULL);
/*7*/	_exit(0);
/*8*/	}
/*9*/	if (fork()==0) {
/*10*/	execlp("echo", "echo", "P4",NULL);

/*11*/	_exit(0);
/*12*/	}
/*13*/	}

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 5.
La réponse correcte est :
/*0*/ → if (fork()==0) {,

/*1*/ → if (fork()==0) {,

/*2*/ → execlp("echo", "echo", "P2",NULL);,

/*3*/ → },

/*4*/ → if (fork()==0) {,

/*5*/ → execlp("echo", "echo", "P3",NULL);,

/*6*/ → },

/*7*/ → if (fork()==0) {,

/*8*/ → execlp("echo", "echo", "P4",NULL);,

/*9*/ → },

/*10*/ → while(wait(NULL)>0);,

/*11*/ → _exit(0);,

/*12*/ → },

/*13*/ → while(wait(NULL)>0);

Commentaire :

Question **13**

Terminer

Non noté

Si vous éprouvez des difficultés à sélectionner les instructions à insérer aux endroits adéquats, vous pouvez répondre à la question en donnant tout le code ici. Limitez-vous cependant à la liste de choix d'instructions.

Idée générale du code

On fork, si on est le fils (P1), rentrons dans le if

On fork (P2) si on est le fils (P2) affichons le bon le message P2, puis terminons le processus de ce fils

On fork (P3) si on est le fils (P3) affichons le bon le message P3, puis terminons le processus de ce fils

On fork (P4) si on est le fils (P4) affichons le bon le message P4, puis terminons le processus de ce fils

Question **14**

Terminer

Note de 1,50 sur 1,50

Considérez le code suivant :

```
int main()  {
  int p;  mkfifo("inf2610",0660);
  if (fork()!=0)
   {     p=open("inf2610",O_WRONLY);
         write(p,"HELLO",5);
         close(p);
         return 0;
   }
   char c; p=open("inf2610",O_RDONLY);
   while(read(p,&c,1)>0)
         write(1,&c,1);
   close(p);
   return 0;
}
```

Le processus fils va-t-il se retrouver bloqué pour toujours, dans le cas où son père ne ferme pas le descripteur d'écriture ?

- (•) Non
- ( ) Oui

Votre réponse est correcte.

La réponse correcte est :
Non

Question **15**

Terminer

Non noté

Justifiez votre choix de réponse (si votre justification est absente ou erronée vous aurez **0 point** pour cette question que votre réponse dans le QCM soit bonne ou mauvaise).

Si je comprends bien la question, "meme sans que le pere ferme le descripteur d'écriture" veut dire, "si on supprime la ligne close(p)" en jaune orange

Je répondrais que non, le fils ne va pas nécessairement se retrouver bloqué pour toujours.

Dans le fond si le père arrive à retourne return 0; avant que le fils à son open (char c; p=open("inf2610",O_RDONLY);) tout ira bien car je crois que le return 0 ferme automatiquement le fichier. (on arrivera à write au final)

Par contre, si le fils se rend au open avant que le père n'ait le temps de return 0, alors on a 2 écrivains et le fils ne pourra pas ouvrir le fichier (on arrivera pas à write au final)

J'espère que je suis clair

Question **16**

Terminer

Note de 1,50 sur 2,50

Considérez le code suivant :

```
int main () {
   /*0*/
   if(fork()==0)  {  // fils
          if(fork()!=0)  {
                /*1*/
               /*2*/
                write(1, ”message du fils\n”, 16);
                /*3*/
                /*4*/
               _exit(0);
          }
          // petit fils
          char c;
         /*5*/
         /*6*/
          while (read(0,&c,1)>0)
                write(1,&c, 1);
           _exit(0);
    }
  /*7*/
  /*8*/
  /*9*/
   write(1, ”message du père\n”, 16);
   /*10*/
   /*11*/
 return 0;
}
```

Complétez le code pour que le processus petit fils récupère, via un tube anonyme et les appels système read et write figurant dans le code, les messages de son père et de son grand père. Chaque processus parent doit attendre la fin de son fils juste avant de se terminer.

Pour répondre à cette question, choisissez les instructions à insérer aux endroits appropriés. Sélectionnez "rien" s'il n'y a aucune instruction à insérer dans cette ligne.

Commentaire /*0*/	int fd[2]; pipe(fd);
Commentaire /*1*/	dup2(fd[1],1);
Commentaire /*2*/	dup2(fd[0],0);
Commentaire /*3*/	close(fd[1]); close(fd[0]);
Commentaire /*4*/	while (wait(NULL)>0);
Commentaire /*5*/	dup2(fd[1],1);
Commentaire /*6*/	dup2(fd[0],0);
Commentaire /*7*/	close(fd[1]); close(fd[0]);
Commentaire /*8*/	while (wait(NULL)>0);
Commentaire /*9*/	dup2(fd[1],1);
Commentaire /*10*/	close(1);

Commentaire /*11*/ while (wait(NULL)>0);

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 5.
La réponse correcte est :
Commentaire /*0*/ → int fd[2]; pipe(fd);,

Commentaire /*1*/ → dup2(fd[1],1);,

Commentaire /*2*/ → close(fd[1]); close(fd[0]);,

Commentaire /*3*/ → close(1);,

Commentaire /*4*/ → while (wait(NULL)>0);,

Commentaire /*5*/ → dup2(fd[0],0);,

Commentaire /*6*/ → close(fd[1]); close(fd[0]);,

Commentaire /*7*/ → dup2(fd[1],1);,

Commentaire /*8*/ → close(fd[1]); close(fd[0]);,

Commentaire /*9*/ → rien,

Commentaire /*10*/ → close(1);,

Commentaire /*11*/ → while (wait(NULL)>0);

Commentaire :

Question **17**

Terminer

Non noté

Si vous éprouvez des difficultés à sélectionner les instructions à insérer aux endroits adéquats, vous pouvez répondre à la question en donnant tout le code ici. Limitez-vous cependant à la liste de choix d'instructions.

L'idée générale est la suivante :

le fils redirige la sortie et l'entrée standard

il écrit

il ferme

il attend

le petit fils redirige la sortie et l'entrée standard (il récupère l'héritage du fils)

il écrit

il ferme

il attend

le père redirige la sortie et l'entrée standard (il récupère l'héritage du fils et du petit fils)

il écrit

il ferme

il attend

Question **18**

Terminer

Note de 1,50 sur 1,50

Le pseudo-code suivant est une variante de la solution au problème des producteurs-consommateurs vue en classe.

```
const int N = 2;
int tampon [N], ip=0, ic=0;
Semaphore libre=N, occupe=0, mutex=1;

Producteur ()
{   while(1)
    {   P(libre);
        produire(tampon,ip);
        Pmutex);
        ip = (ip +1)%N;
        V(mutex);
        V(occupe);
    }
}

Consommateur ()
{   while(1)
    {   P(occupe);
        consommer(tampon,ic);
        P(mutex);
        ic= (ic+1)%N;
        V(mutex);
        V(libre) ;
    }
}
```

Est-ce que cette modification peut altérer le fonctionnement des producteurs et consommateurs, comparativement à la solution vue en classe ?

Sélectionnez une réponse. Justifiez votre réponse à la page suivante. Si votre réponse est oui, vous devez donner un scénario complet qui montre le problème. Si votre réponse est non, vous devez expliquer pourquoi.

- Non, le fonctionnement des producteurs et consommateurs n'est pas altéré.
- Aucune de ces réponses.
- Oui, car il y a un risque d'écraser une production avant sa consommation.
- Oui, car il y a un risque d'interblocage.
- Oui, car il y a un risque pour les consommateurs de consommer avant aucune production.

Votre réponse est correcte.

La réponse correcte est :
Oui, car il y a un risque d'écraser une production avant sa consommation.

Question **19**

Terminer

Non noté

Justifiez votre choix de réponse (si votre justification est absente ou erronée vous aurez **0 point** pour cette question que votre réponse dans le QCM soit bonne ou mauvaise).

On peut sur-produire.
Les producteur pourrait produire beaucoup trop et écraser complètement les consommateur (donc une production serait écrasé avant même qu'elle n'ait le temps de se faire consommer)
Dans le cas ou P(libre) serait disponible, mais pas P(mutex), la boucle exécuterait en boucle P(libre) et produire(tampon, ip)

Pour régler le problème, il faudrait en inverser les deux lignes suivantes :

produire(tampon, ip)
P(mutex)

P(mutex) devrait être la première (avant produire(tampon, ip))

Question **20**

Terminer

Note de 2,50 sur 2,50

Complétez, en ajoutant les sémaphores nécessaires le code suivant pour que l'action a1 soit exécutée avant b3 et que b1 soit exécutée avant a1.

```
/*0*/
P1 {            P2 {
/*1*/           /*4*/
a1;             b1;
                /*5*/
/*2*/           b2;
a2;             /*6*/
/*3*/           b3;
                /*7*/

}               }
```

Pour répondre à cette question, sélectionnez les instructions à insérer aux endroits appropriés. Sélectionnez "rien", s'il n'y a aucune instruction à insérer. Vous devez utiliser deux sémaphores S1 et S2 pour bloquer/ débloquer respectivement les processus P1 et P2.

/*0*/	Semaphore S1=1, S2=0;
/*1*/	P(S2);
/*2*/	V(S1);
/*3*/	rien
/*4*/	P(S1);
/*5*/	V(S2);
/*6*/	P(S1);
/*7*/	rien

Votre réponse est correcte.

La réponse correcte est :
/*0*/ → Semaphore S1=0, S2=0;,

/*1*/ → P(S1);,

/*2*/ → V(S2);,

/*3*/ → rien,

/*4*/ → rien,

/*5*/ → V(S1);,

/*6*/ → P(S2);,

/*7*/ → rien

Commentaire :

Question **21**

Terminer

Non noté

Si vous éprouvez des difficultés à sèlectionner les instructions à insérer aux endroits adéquats, vous pouvez répondre à la question en donnant tout le code ici. Limitez-vous, cependant, à la liste de choix d'instructions.

Alors au début on a deux sémaphore s1=1 et s2 = 0

on veut cette suite: b1 -> a1 -> b3

Donc on veut pas que a1 s'exécute tt de suite, on va mettre un P(s2) devant, comme s2=0 au début, impossible d'exécuter a1

Ensuite on peut mettre s1 à 0 donc P(s1) puis exéctuer b1 (première instruction obligatoire)

Après on peut tjrs rien faire coté a1, donc on libère s2 à 1 avec V(s2) dans le P2. On s'en fout de b2 (mais bon, il s'exectute) et après a P(s1) mais s1=0, donc c'est impossible de se rendre à b3.

Côté P1, là s2=1 donc on peut le produire, ensuite on exécute a1 (deuxieme instruction obligatoire), puis on libère S1 avec V(s1) ce qui permmet de ...

... de continuer côté P2 et d'exécuter b3 (instruction obligatoire)

La suite côté P1 on s'en fout, on a déjà exécuter les 3 actions dans le bon ordre (même si au final on pourrait ajouter que a2 s'exécute)

Aller à...

Documentation personnelle pour le final INF2610 - Hiver 2022 - Déposez ici vos résumés en format Pdf de vos notes de cours ►