Commencé le	lundi 31 octobre 2022, 09:00
État	Terminé
Terminé le	lundi 31 octobre 2022, 11:26
Temps mis	2 heures 26 min
Note	**15,50** sur 20,00 (**77,5**%)

Question **1**

Non répondue

Non noté

Directives :

1. Cet examen est composé de 13 questions pour une durée totale de 2 heures.
2. Pondération 30%.
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. Ne pas joindre de fichiers à vos réponses.
8. Lisez au complet chaque question avant d'y répondre.

Bon examen à tous

Question **2**

Terminé

Non noté

Sur mon honneur, j'affirme que je compléterai cet examen en vertu du code de conduite de l'étudiant de Polytechnique Montréal et de sa politique sur le plagiat. J'affirme également que je compléterai cet examen par moi-même, sans communication avec personne, et selon les directives diffusées sur les canaux de communication.

Écrivez votre nom complet ainsi que votre matricule en guise d'approbation dans la zone de texte ci-dessous.

[redacted]

Question **3**

Correct

Note de 1,00 sur 1,00

Dans un système monoprocesseur et multiprogrammé en temps partagé, il n'est pas possible d'exécuter un processus pendant qu'un autre est bloqué en attente d'une entrée/sortie.

Veuillez choisir une réponse.

- Vrai
- Faux ✔

La réponse correcte est « Faux ».

Question **4**

Correct

Note de 1,00 sur 1,00

Un processus crée un fils via l'appel système fork. Juste après sa création, le processus fils ferme le descripteur 0 (STDIN). Le processus père va alors perdre l'accès au fichier associé à STDIN.

Veuillez choisir une réponse.

- Vrai
- Faux ✔

La réponse correcte est « Faux ».

Question **5**

Correct

Note de 1,00 sur 1,00

Sous Linux, le père d'un processus peut être différent du processus créateur.

Veuillez choisir une réponse.

- Vrai ✔
- Faux

La réponse correcte est « Vrai ».

Question **6**

Correct

Note de 1,00 sur 1,00

Un processus se transforme avec succès en appelant la fonction execlp. Indiquez parmi les éléments suivants celui qui est conservé suite à cette transformation.

- Son espace d'adressage.
- Aucune de ces réponses.
- Sa sortie standard. ✔
- Sa pile d'exécution.
- Sa table des pages.

La réponse correcte est : Sa sortie standard.

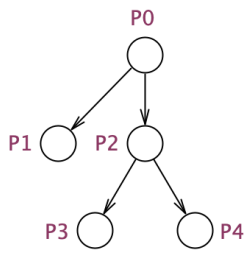
Question **7**

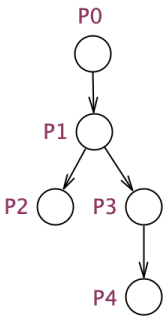
Correct

Note de 2,50 sur 2,50

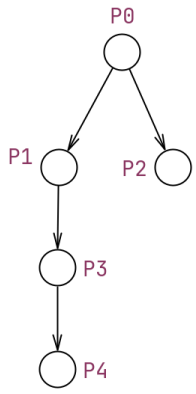
Quel arbre représente la hiérarchie de processus générés par la fonction construireArbre suivante (où P0 est le processus exécutant la fonction) ?

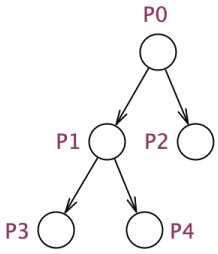
```
void construireArbre() {
    for(int i=0; i<2; i++) {
        if(fork()) {
            fork();
            break;
        } else
            if(i==1) break;
    }
   while(wait(NULL)>0);
  _exit(0);
}
```



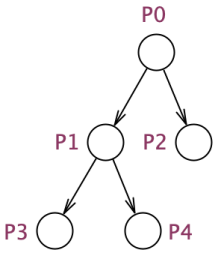


Aucune de ces réponses





La réponse correcte est :



Question **8**

Terminé

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).

i = 0

fork () ; Création de P1

Le père entre rempli la condition if (P)

Le père fork() ; création de P2 (P)

Le père excécute la prochaine ligne ; break et sort de la boucle for (P)

Le P2 excécute lui aussi la prochaine ligne ; break et sort de la boucle for (P2)

P1 ne remplit pas la condition if de fork() (P1)

P1 ne remplit pas la condition if(i==1), donc il continue

i est incrémenté de 1

i = 1

P1 fork() ; création de P3

P1 fork() encore une fois ; création de P4

P1 excécute la prochaine ligne ; break et sort de la boucle for

P3 excécute la prochaine ligne ; break et sort de la boucle for

P4 ne remplit pas la condition de if fork() et excécute le else

P4 remplit bel et bien la condition if(i==1), donc il exécute break et sort de la boucle for

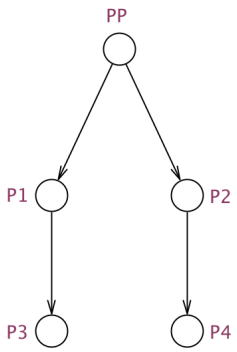
Question **9**

Correct

Note de 2,50 sur 2,50

```
int main() {
/*0*/
/*1*/
/*2*/
/*3*/
/*4*/
/*5*/
/*6*/
/*7*/
/*8*/
/*9*/
/*10*/
/*11*/
/*12*/
/*13*/
/*14*/
/*15*/  _exit(0);
}
```

Complétez le code ci-dessus pour créer l'arbre de processus décrit par la figure suivante (où PP 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 (P3 ou P4). **Rappel : les appels système ne retournent aucune erreur.**



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", "P3",NULL);	✔
/*3*/	}	✔
/*4*/	while(wait(NULL)>0);	✔
/*5*/	_exit(0);	✔
/*6*/	}	✔
/*7*/	if (fork()==0) {	✔

/*8*/	if (fork()==0) {	✔
/*9*/	execlp("echo", "echo", "P4",NULL);	✔
/*10*/	}	✔
/*11*/	while(wait(NULL)>0);	✔
/*12*/	_exit(0);	✔
/*13*/	}	✔
/*14*/	while(wait(NULL)>0);	✔

La réponse correcte est :
/*0*/ → if (fork()==0) {,

/*1*/ → if (fork()==0) {,

/*2*/ → execlp("echo", "echo", "P3",NULL);,

/*3*/ → },

/*4*/ → while(wait(NULL)>0);,

/*5*/ → _exit(0);,

/*6*/ → },

/*7*/ → if (fork()==0) {,

/*8*/ → if (fork()==0) {,

/*9*/ → execlp("echo", "echo", "P4",NULL);,

/*10*/ → },

/*11*/ → while(wait(NULL)>0);,

/*12*/ → _exit(0);,

/*13*/ → },

/*14*/ → while(wait(NULL)>0);

Question **10**

Terminé

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.

```
int main() {
    if(fork()==0){
        if(fork()==0){
            execlp("echo", "echo", "P3", NULL);
        }
        while(wait(NULL)>0);
        _exit(0);
    }
    if(fork()==0){
        if(fork()==0){
            execlp("echo", "echo", "P4", NULL);
        }
        while(wait(NULL)>0);
        _exit(0);
    }
    while(wait(NULL)>0);
    _exit(0);
}
```

Question **11**

Incorrect

Note de 0,00 sur 1,00

Dans un système monoprocesseur, un processus crée 5 threads utilisateur puis se met en pause en appelant la fonction pause(). Le système pourrait allouer le processeur à l'un des threads utilisateur du processus.

Veuillez choisir une réponse.

- Vrai ✖
- Faux

La réponse correcte est « Faux ».

Question **12**

Partiellement correct

Note de 0,50 sur 1,50

Un processus crée 4 threads POSIX, se met en attente de la fin d'exécution des threads, affiche la valeur d'un compteur globale v initialisé à 0, puis se termine. Chaque thread créé incrémente v de 4 (v=v+4) puis se termine. La valeur de v affichée par le thread principal pourrait être 12.

Veuillez choisir une réponse.

- Vrai ✔
- Faux

La réponse correcte est « Vrai ».

Question **13**

Terminé

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 donner un scénario d'exécution ou une explication claire qui confirme votre choix de réponse.

Puisque les threads accèdent de manière concurentielle en écriture et en lecture à la variable globale a. Ainsi, le programme est non déterministe et donc, il est impossible de déterminer la valeur exacte que prendra v à la fin de l'excécution du processus.

On peut affirmer que la valeur de v affichée par le thread principal pourrait être 12 puisque l'éventail des réponses pourrait être {0, 4, 8 , 12, 16}. 12 se retrouve dans l'éventail des réponses possibles et donc, il serait possible que la valeur v affichée par le thread principal pourrait être 12.

Question **14**

Correct

Note de 1,50 sur 1,50

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

Sémaphore x = 1, y=0;

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.

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

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

Question **15**

Terminé

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).

Les trois processus sont excécutés en concurrence.

A prend le jeton de X (x est décrémenté de 1)- B est dans la file d'attente de X - C est dans la file d'attente de Y (x= 0 , y = 0)

a1 est excécuté (x = 0 , y = 0)

V(x) incrémente le sémaphore x (x =1, y = 0)

B prend le jeton X (x = 0, y = 0)

B se met dans la file d'attente de Y (file d'attente (FIFO) : C, B)

a2 est excécuté (x = 0, y = 0)

V👍 incrémente le sémaphore y (x = 0, y = 1)

C prend le jeton y (x = 0 , y = 0)

Le processus B est en attente du jeton Y

Le processus C est en attente du jeton X

Il n'y a plus aucun jetons de disponible, le processus B et C sont donc bloqués

Toutefois, il existerait un scénario "possible" à excécuter pour ce processus, admettant des hypothèses :

B est avant C dans la file pour le jeton Y lorsque le jeton y est libéré après l'excécution de a2.

Ainsi, la séquence serait la suivante :

B possède le jeton X et Y et donc, la fonction b est excécutée.

Le jeton X est libéré (x = 1, y = 0)

Le jeton Y est libéré (x = 1, y =1)

C prend le jeton Y (x = 1, y = 0)

C prend le jeton X (x = 0, y = 0)

C possède le jeton X et Y et donc, la fonction c1 est excécutée

Le jeton X est libéré (x= 1, y = 0)

La fonction c2 est exécutée.

L'ordre serait donc a1; a2; b; c1; c2

Question **16**

Correct

Note de 1,50 sur 1,50

Considérez le code suivant :

```
 int main()  {
   int p[2];  pipe(p);
   if (fork()==0)
    {     char A[100]; close(p[0]);
          sprintf(A,"%d",p[1]);
          execl("./prog", "prog", A, NULL);
    }
    wait(NULL);
    char c;
    while(read(p[0],&c,1)>0)
          write(1,&c,1);
    return 0;
}
```

L'exécutable "prog" récupère le paramètre A et le résultat de sa conversion en un entier dans une variable entière fd. Il dépose correctement un message dans le tube puis se termine. Est-ce que le processus père va pouvoir lire, afficher le message déposé dans le tube et se terminer ?

- Oui
- Non ✔

La réponse correcte est :
Non

Question **17**

Terminé

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).

Non puisque le fils n'a pas correctement fermé le descripteur d'écriture. Tant qu'il n'y a un descripteur d'ouvert, le lecteur va attendre la fermeture du tube. De plus, le père n'a pas fermé son descripteur de lecture et d'écriture et donc, le tube va rester ouvert, prévenant ainsi la terminaison du programme.

Ainsi, le père ne va jamais pouvoir correctement se terminer.

Question **18**

Correct

Note de 1,50 sur 1,50

Le code d'un processus consiste, dans l'ordre, à créer un tube nommé, ouvrir le tube en lecture, créer un processus fils, lire du tube le message de son fils, afficher à l'écran le message lu, fermer le descripteur de lecture, se mettre en attente de la fin d'exécution de son fils, et se terminer. Le code du processus fils consiste, dans l'ordre, à ouvrir le tube en écriture, déposer un message dans le tube, fermer le descripteur d'écriture, et se terminer. Le processus père va pouvoir créer le processus fils et communiquer avec lui.

Veuillez choisir une réponse.

- Vrai
- Faux ✔

La réponse correcte est « Faux ».

Question **19**

Terminé

Non noté

Justifiez votre choix de réponse (si votre justification est absente ou erronée vous aurez **0 point** pour cette question que votre réponse au QCM soit bonne ou mauvaise). Pour la justification, il suffit de donner une explication claire qui confirme votre choix de réponse.

Père :

Créer tube (mkfifo)

Ouvrir tube en lecture ; int fd = open(... , O_RONLY)

Créer processus fils ; fork() == 0

Ouvrir tube en lecture ; int fd = open(... , O_RONLY)

Fermer descripteur fichier ; close(fd)

Attendre fin processus fils ; wait(NULL)

Terminer ; exit(0)

Fils :

Ouvrir tube en écriture ; int fd = open(..., O_WRONLY)

Écrire message tube : write(...)

close(fd);

exit(0);

NON!

Si on regarde bel et bien l'ordre dans laquelle le père excécute les instructions, on remarque que celui-ci ouvre en lecture le tube nommé avant de créer le processus fils. Ainsi, lorsqu'il ouvre le tube en lecture, le processus est bloqué (puisque l'ouverture d'un tube nommé est bloquante). Ainsi, le processus père va rester bloquer en attendant l'ouverture du tube en écriture qui n'arrivera jamais.

Question **20**

Partiellement correct

Note de 1,50 sur 2,50

Considérez le code suivant :

```
int main () {
  /*0*/
  if(fork()==0)  {  // premier fils
        /*1*/
        /*2*/
        write(1, ”message du fils1\n”, 17);
       _exit(0);
  }
   if(fork()==0)  {  // second fils
         /*3*/
         /*4*/
         write(1, ”message du fils2\n”, 17);
        _exit(0);
  }
  char c;
  /*5*/
  /*6*/
  /*7*/
  while (read(0,&c,1) >0)
       write(1,&c, 1);
 /*8*/
  return 0;
}
```

Complétez le code pour que le processus principal récupère, via un tube anonyme et les appels système read et write figurant dans le code, les messages de ses fils. Le processus principal doit attendre la fin de ses 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*/	close(fd[1]); close(fd[0]);	✔
Commentaire /*3*/	dup2(fd[1],1);	✔
Commentaire /*4*/	close(fd[1]); close(fd[0]);	✔
Commentaire /*5*/	while (wait(NULL)>0);	✘
Commentaire /*6*/	close(fd[1]); close(fd[0]);	✔
Commentaire /*7*/	dup2(fd[0],0);	✘
Commentaire /*8*/	rien	✘

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*/ → dup2(fd[1],1);,

Commentaire /*4*/ → close(fd[1]); close(fd[0]);,

Commentaire /*5*/ → dup2(fd[0],0);,

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

Commentaire /*7*/ → rien,

Commentaire /*8*/ → while (wait(NULL)>0);

Question **21**

Terminé

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.

```
int main() {
    int fd[2];
    pipe(fd);
    if(fork()==0){
        dup2(fd[1],1);
        close(fd[1]);
        close(fd[0]);
        write(1, "message du fils1\n", 17);
        _exit(0);
    }
    if(fork()==0){
        dup2(fd[1],1);
        close(fd[1]);
        close(fd[0]);
        write(1, "message du fils2\n", 17);
        _exit(0);
    }
    char c;
    while(wait(NULL)>0);
    dup2(fd[0],0);
    close(fd[0]);
    close(fd[1]);
    while (read(0,&c,1) >0)
        write(1,&c, 1);
}
```

Question **22**

Incorrect

Note de 0,00 sur 1,50

Considérez le programme suivant :

```
void action(int sig) {
   wait(NULL);
}
int main() {
   signal(SIGUSR1,action);
   if(fork()==0)
      {  signal(SIGUSR1,SIG_IGN);
         for(int i=0; i<=1000000; i++);
         kill(getppid(),SIGUSR1);
       _exit(0);
      }
   pause();
   _exit(0);
}
```

Est-ce que le père pourrait se retrouver en pause pour toujours ?

Veuillez choisir une réponse.

- Vrai
- Faux ✖

La réponse correcte est « Vrai ».

Question **23**

Terminé

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 donner un scénario d'exécution ou une explication qui confirme votre choix de réponse.

Non puisque le père à sa propre table de gestion des signaux. Dans celle-ci, on y retrouve "action" qui est lancé par SIGUSR1. De ce fait, lorsque le processus père se met en pause et attend un signal, le fils va lui envoyer le signal dont il a besoin pour sortir de sa pause (soit attendre la fin de l'exécution du processus fils). Si on avait appelé kill(getpid(), SIGUSR1), alors à ce moment là, le père aurait bel et bien pu se retrouver en pause pour toujours.