Commencé le	lundi 23 octobre 2023, 18:00
État	Terminé
Terminé le	lundi 23 octobre 2023, 19:48
Temps mis	1 heure 48 min
Note	**19,55** sur 20,00 (**97,75**%)

Question 1

Non répondue

Non noté

Directives :

1. Cet examen est composé de 12 questions pour une durée totale de 2 heures.
2. Pondération 30%.
3. En guise de documentation, vous aurez accès à votre compte Moodle et, par conséquent, à la totalité du site du cours INF2610, incluant les diapos, etc. Aucune autre documentation ne sera permise. Vous aurez droit à la calculatrice non-programmable. Nous vous fournissons également une feuille brouillon.
4. Aucune réponse aux questions durant l'examen.
5. Tous les appels système utilisés dans les questions sont supposés exempts d'erreurs et considèrent leurs options par défaut.
6. Il n'est pas demandé de traiter les cas d'erreurs, ni d'inclure les directives d'inclusion dans les codes à compléter.
7. Pour les questions à choix multiples, **vous devez sélectionner une seule réponse**.
8. Il n'est pas possible de joindre des fichiers à vos réponses.
9. Lisez au complet chaque question avant d'y répondre.
10. Vous pouvez inclure vos commentaires au responsable du cours dans l'espace ci-après.

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

Terminé

Note de 1,00 sur 1,00

Juste après un fork, sélectionnez l'élément du parent qui peut être différent de celui de son processus enfant.

- a. Les valeurs des variables globales.
- b. Le programme exécuté
- c. Le contenu de la table des descripteurs de fichiers
- d. Les signaux en attente
- e. La valeur du compteur ordinal

Votre réponse est correcte.

La réponse correcte est :
Les signaux en attente

Question 4

Terminé

Note de 1,00 sur 1,00

Considérez le code suivant:

```
char X[3]= "RST";

int main() {
  if (fork()==0)  { // F1
    printf("%c \n", X[0]);
    exit(0);
  }

  printf("%c", X[0]);

  if (fork()==0)  { // F2
    printf("%c \n", X[1]);
    exit(0);
  }

  printf("%c", X[1]);
  printf("%c \n",X[2]);
  wait(NULL); wait(NULL);
  return 0;
}
```

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

- a. PP: RST F1: RS F2: RS
- b. PP: RST F1: R F2: RS
- c. PP: RST F1: R F2: S
- d. PP: RRSST F1: RS F2: RRS
- e. PP: RRSST F1: R F2: RS

Votre réponse est correcte.

La réponse correcte est :
PP: RST F1: R F2: RS

Question 5

Terminé

Note de 1,00 sur 1,00

Lequel des énoncés représentent une vraie distinction entre un thread utilisateur et un thread noyau?

- ○ Les changements de contexte sont moins couteux pour les threads noyau.
- ◉ L'ordonnanceur du système alloue du temps CPU aux threads noyau.
- ○ Les threads de la bibliothèque pth sont gérés par le noyau.
- ○ Les threads noyau sont en général plus portables.

Votre réponse est correcte.

La réponse correcte est : L'ordonnanceur du système alloue du temps CPU aux threads noyau.

Question 6

Terminé

Note de 0,75 sur 1,00

Complétez la phrase suivante.

Suite aux appels systèmes *pipe(fd)* puis *dup2(fd[1], 1)*, la position 1 de la [table des descripteurs de fichiers] contiendra une référence vers la position de la [table des descripteurs de fichiers] correspondant [au tube créé] et ce, en [écriture].

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 3.
La réponse correcte est :

Complétez la phrase suivante.

Suite aux appels systèmes *pipe(fd)* puis *dup2(fd[1], 1)*, la position 1 de la [table des descripteurs de fichiers] contiendra une référence vers la position de la [table des fichiers ouverts] correspondant [au tube créé] et ce, en [écriture].

Question 7

Terminé

Note de 1,00 sur 1,00

Lequels des énoncés suivants est FAUX?

a. L'utilisation d'un verrou actif peut mener à une inversion de priorité sous certaines conditions.

b. Il est possible d'implémenter un sémaphore à l'aide d'un verrou actif, d'un compteur et d'une file d'attente.

c. Un sémaphore initialisé à 1 ne prendra jamais de valeur supérieure à 1.

d. Les instructions atomiques se prêtent mal à la prévention de l'exclusion mutuelle pour des sections critiques de grande taille.

e. Le masquage des interruptions n'est pas possible en mode utilisateur sous Linux.

Votre réponse est correcte.

La réponse correcte est :
Un sémaphore initialisé à 1 ne prendra jamais de valeur supérieure à 1.

Question 8

Terminé

Note de 1,00 sur 1,00

Un processus exécute le code suivant:

```
int x=0;

int main()
{
  x=x+2;

  if(fork()>0) {
    x=x+3;
  } else {
    x=x+2;
    _exit(0);
  }

  x=x+1;
  wait(NULL);
  printf("x=%d\n",x);
  return 0;
}
```

Lequel des énoncés suivants est FAUX? (Considérez que les appels système réussissent sauf lorsqu'indiqué autrement.)

a. À l'exécution, le programme affichera *x=6*.

b. Si l'appel à *fork* échoue, alors le programme poursuivra son exécution mais n'affichera rien.

c. Immédiatement après l'exécution de *wait*, le processus parent enverra à son enfant le signal SIGCHLD.

d. Le processus parent pourrait récupérer par l'exécution de *wait* la valeur du paramètre de la fonction *_exit* ainsi que le PID du processus enfant terminé, mais il ne le fait pas.

e. Immédiatement après l'exécution de *fork*, nous serons en présence de deux processus distincts, et la valeur de x sera de 2 dans les deux cas.

Votre réponse est correcte.

La réponse correcte est :
Immédiatement après l'exécution de *wait*, le processus parent enverra à son enfant le signal SIGCHLD.

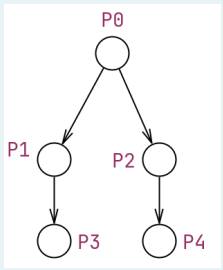
Question 9

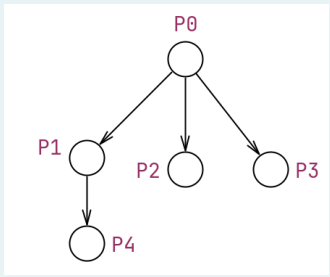
Terminé

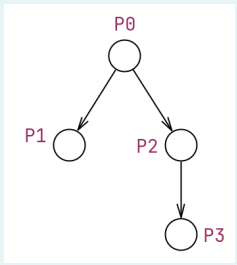
Note de 2,00 sur 2,00

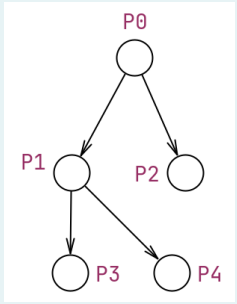
Quel arbre représente la hiérarchie de processus générés par la fonction suivante (où P0 est le processus appelant de la fonction construireArbre) :

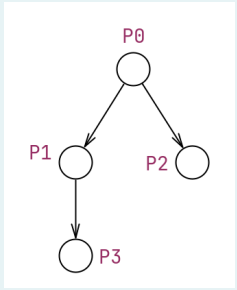
```
void construireArbre()
{
  for(int i=0; i<3; i++) {
    if(i!=2) {
      if (fork()==0) {
        fork();
        break;
      }
    } else break;
  }

  while(wait(NULL)>0);
  _exit(0);
}
```





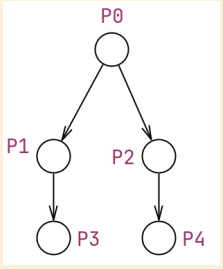






Votre réponse est correcte.

La réponse correcte est :



Question 10

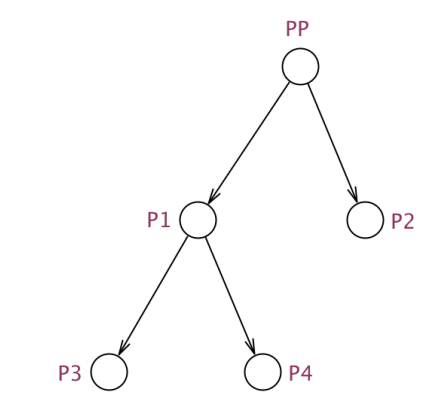
Terminé

Note de 3,00 sur 3,00

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

Examinez attentivement le code ci-dessus, constitué d'une fonction *main* comportant 14 instructions manquantes suivies d'un appel système *_exit*. Indiquez les instructions manquantes pour créer l'arbre de processus décrit à la figure suivante (où PP est le processus principal). Veillez à respecter les consignes suivantes:

- Chaque processus parent doit attendre la fin de ses processus enfants, puis 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).
- Vous n'avez pas à répéter l'appel système *_exit* à la toute fin du programme.



Pour répondre à cette question, sélectionnez les instructions à insérer aux bons endroits. *Il n'est pas censé rester d'instruction vide.*

/*0*/	if (fork()==0) {
/*1*/	if (fork()==0) {
/*2*/	execlp("echo", "echo", "P3",NULL);
/*3*/	}

/*4*/ if (fork()==0) {

/*5*/ execlp("echo", "echo", "P4",NULL);

/*6*/ }

/*7*/ while(wait(NULL)>0);

/*8*/ _exit(0);

/*9*/ }

/*10*/ if (fork()==0) {

/*11*/ execlp("echo", "echo", "P2",NULL);

/*12*/ }

/*13*/ while(wait(NULL)>0);

Votre réponse est correcte.

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

/*1*/ → if (fork()==0) {,

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

/*3*/ → },

/*4*/ → if (fork()==0) {,

/*5*/ → execlp("echo", "echo", "P4",NULL);,

/*6*/ → },

/*7*/ → while(wait(NULL)>0);,

/*8*/ → _exit(0);,

/*9*/ → },

/*10*/ → if (fork()==0) {,

/*11*/ → execlp("echo", "echo", "P2",NULL);,

/*12*/ → },

/*13*/ → while(wait(NULL)>0);

Question 11

Terminé

Note de 2,00 sur 2,00

Considérez le code suivant :

```
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
  char c;
  int p;  mkfifo("inf2610p", 0660);
  int q;  mkfifo("inf2610q", 0660);
  if (fork()!=0) { //
    p = open("inf2610p", O_WRONLY);
    q = open("inf2610q", O_RDONLY);
    // LIGNE X
    const char * const m1 = "Bonjour! Comment allez-vous?\n";
    write(p, m1, strlen(m1));
    close(p);
    while(wait(NULL)>0); // LIGNE DU 'WAIT'
    while(read(q,&c,1)>0)
      write(1,&c,1);
    close(q); // LIGNE QQ
    return 0;
  }
  p = open("inf2610p", O_RDONLY);  // LIGNE DE LA TROISIEME INSTRUCTION 'OPEN'
  q = open("inf2610q", O_WRONLY);
  const char * const m2 = "Tres bien merci, je vous en prie!\n";
  write(q, m2, strlen(m2));
  close(q);
  // LIGNE Y
  while(read(p,&c,1)>0)
    write(1,&c,1);
  close(p); // LIGNE PP
  execlp("echo", "echo", "(Changement de tour...)", NULL);
}
```

Ce programme est censé permettre à deux processus de s'échanger deux messages (l'un dans une direction, l'autre dans l'autre direction). Chacun des deux messages est ensuite envoyé à la sortie standard. Un message de changement de tour est intercalé entre ces deux messages. La sortie attendue est:

```
Bonjour! Comment allez-vous?
(Changement de tour...)
Tres bien merci, je vous en prie!
```

Lequel des énoncés suivants est *vrai*?

- (●) Si on éliminait l'instruction *wait*, cela affecterait la fonctionnalité du programme mais n'accroitrait ni ne réduirait le risque d'interblocage.
- ( ) Si on enlevait l'instruction *close* à la ligne QQ, cela accroitrait le risque d'interblocage.
- ( ) Si on enlevait l'instruction *close* à la ligne PP, cela réduirait le risque d'interblocage.
- ( ) Si on déplaçait l'instruction *wait* vers la LIGNE X, le programme pourrait s'exécuter sans qu'un interblocage soit susceptible de survenir.
- ( ) Si on déplaçait la première instruction *open* vers la LIGNE Y, le programme pourrait s'exécuter sans qu'un interblocage soit susceptible de survenir.

Votre réponse est correcte.

La réponse correcte est :
Si on éliminait l'instruction *wait*, cela affecterait la fonctionnalité du programme mais n'accroitrait ni ne réduirait le risque d'interblocage.

Question 12

Terminé

Note de 1,80 sur 2,00

Considérez le code suivant :

```
int main () {
  /*0*/
  if(fork()==0)  {  // fils
          if(fork()==0)  {  // petit fils
          /*1*/
          /*2*/
           write(1, ”message du petit fils\n”, 22);
           _exit(0);
           }
         // fils
          char c;
          /*3*/
          /*4*/
          while (read(0,&c,1) >0)
            write(1,&c, 1);
          /*5*/
           _exit(0);
   }
        /*6*/
         /*7*/
 write(1, ”message du pere\n”, 16);
          /*8*/
          /*9*/
  return 0;
}
```

Complétez le code pour que le processus fils récupère, via un tube anonyme et les appels système read et write figurant dans le code, les messages de son fils et de son 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*/	close(fd[1]); close(fd[0]);
Commentaire /*3*/	dup2(fd[0],0);
Commentaire /*4*/	dup2(fd[1],1);
Commentaire /*5*/	while (wait(NULL)>0);
Commentaire /*6*/	dup2(fd[1],1);
Commentaire /*7*/	close(fd[1]); close(fd[0]);
Commentaire /*8*/	close(1);
Commentaire /*9*/	while (wait(NULL)>0);

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 9.
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[0],0);,

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

Commentaire /*5*/ → while (wait(NULL)>0);,

Commentaire /*6*/ → dup2(fd[1],1);,

Commentaire /*7*/ → close(fd[1]); close(fd[0]);,

Commentaire /*8*/ → close(1);,

Commentaire /*9*/ → while (wait(NULL)>0);

Question 13

Terminé

Note de 1,00 sur 1,00

Lequel des énoncés suivants est *faux*?

- La commande *kill(pid, SIGINT)* permet d'envoyer le signal SIGINT au processus comportant le numéro *pid*.
- Le signal SIGKILL ne peut être masqué.
- La commande *signal (SIGINT, SIG_IGN)* assigne le gestionnaire de signal par défaut au processus courant pour le signal SIGINT.
- La commande *signal(SIGINT, f)* assigne le gestionnaire de signal *f* au processus courant pour le signal SIGINT.
- Le signal SIGSTOP ne peut être masqué.

Votre réponse est correcte.

La réponse correcte est :
La commande *signal (SIGINT, SIG_IGN)* assigne le gestionnaire de signal par défaut au processus courant pour le signal SIGINT.

Question 14

Terminé

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, et A, B et C s'exécutent en concurrence.

Au départ: Sémaphore x = 1, y=0;

A	B	C
P(x);	P(y);	P(x);
P(y);	P(x);	c1;
a;	b1;	V(x);
V(x);	V(x);	c2;
V(y);	b2;	V(y);

Un seul scénario d'exécution des actions de ces processus est possible parmi les suivants. Lequel?

- c1; a; c2; b1;b2
- c1; c2; b1; b2; a;
- c1; c2; a; b1; b2;
- a; b1; b2; c1; c2;
- c1; c2; b1; a; b2;

Votre réponse est correcte.

La réponse correcte est :
c1; c2; a; b1; b2;

Question 15

Terminé

Note de 1,00 sur 1,00

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

```
const int N = 2;
int tampon [N], ip=0, ic=0;
```

Semaphore libre=N, occupe=0, mutex=1;

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

```
Consommateur ()
{   while(1)
    {   P(occupe);
        P(mutex);
        consommer(tampon,ic);
        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 ?

- Non, le fonctionnement des producteurs et consommateurs n'est pas altéré.
- Oui, car il y a un risque d'insérer dans le tampon plusieurs fois la même production.
- Oui, car il y a un risque pour les consommateurs de consommer alors que le tampon est vide.

  Un producteur réalise P(libre); V(occupe). Un consommateur P(occupe); P(mutex); consommer... avant de produire
- Oui, car il y a un risque d'insérer dans un tampon plein.
- Oui, car il y a un risque d'interblocage.

Votre réponse est correcte.

La réponse correcte est :
Oui, car il y a un risque pour les consommateurs de consommer alors que le tampon est vide.

Question 16

Terminé

Note de 2,00 sur 2,00

Complétez, en ajoutant les sémaphores nécessaires le code ci-après en tenant compte des fonctionnalités et contraintes suivantes:

- L'action a1 doit être exécutée avant l'action b3.
- L'action b2 doit être exécutée avant l'action a2.
- Les deux processus doivent pouvoir s'exécuter librement en concurrence, pourvu que les deux contraintes précédemment énoncées soient respectées. *Le code ne doit pas forcer un ordre un ordre d'exécution particulier au delà de ces contraintes*.
- Vous devez utiliser deux sémaphores S1 et S2 pour bloquer/débloquer respectivement les processus P1 et P2. Autrement dit: vous devez utiliser le sémaphore S1 pour bloquer/débloquer le processus P1, et vous devez utiliser le sémaphore S2 pour bloquer/débloquer le processus P2. *Veuillez ne pas les inverser*.
- Veuillez produire la solution qui soit *la plus économique* en termes de nombre d'instructions. La majorité des cases ne devrait contenir aucune instruction.

```
/*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.

/*0*/ Semaphore S1=0, S2=0;

/*1*/ rien

/*2*/ V(S2); P(S1);

/*3*/ rien

/*4*/ rien

/*5*/ rien

/*6*/ V(S1); P(S2);

/*7*/ rien

Votre réponse est correcte.

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

/*1*/ → rien,

/*2*/ → V(S2); P(S1);,

/*3*/ → rien,

/*4*/ → rien,

/*5*/ → rien,

/*6*/ → V(S1); P(S2);,

```
/*7*/ → rien
```