| | |
|---|---|
| **Commencé le** | samedi 11 mars 2023, 10:00 |
| **État** | Terminé |
| **Terminé le** | samedi 11 mars 2023, 10:55 |
| **Temps mis** | 55 min 15 s |
| **Note** | **20,00** sur 20,00 (**100**%) |

Question **1**

Terminé

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.

C'est quoi le STEP?

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

Un processus se transforme avec succès en appelant la fonction execlp. Indiquez parmi les éléments suivants ceux qui ne sont pas conservés suite à cette transformation.

- Ses liens père-fils
- Son espace d'adressage
- Sa sortie standard
- Sa table des descripteurs de fichiers
- Aucune de ces réponses

Votre réponse est correcte.

La réponse correcte est : Son espace d'adressage

Question **4**

Terminé

Note de 1,00 sur 1,00

Considérez le code suivant :

```
char A[3]= "BCW";

int main()
{
  if (fork()!=0)  { // PP
    /* 1 */
    printf("%c", A[0]);
    if(fork()==0) { // F2

      /* 2 */
      printf("%c \n", A[1]);
      exit(0);
    }

  } else { // F1

    /* 3 */
    printf("%c \n", A[0]);
    exit(0);
  }

  /* 4 */
  printf("%c",A[1]);

  /* 5 */
  printf("%c \n",A[2]);
  wait(NULL); wait(NULL);
  return 0;
}
```

Un affichage possible résultant de l'exécution de ce programme sera:

B
BCW
BC

Dans cet affichage, la ligne "B" s'affiche en premier. À quel endroit devriez-vous placer une instruction "*sleep(1);*" pour faire en sorte que la ligne "B" s'affiche *typiquement* en *dernier*? (Choisissez la meilleure réponse.)

- /* 1 */
- /* 2 */
- /* 3 */ (sélectionné)
- /* 4 */
- /* 5 */

Votre réponse est correcte.

La réponse correcte est :
/* 3 */

Question **5**

Terminé

Note de 1,00 sur 1,00

Un processus crée 5 threads puis se met en attente de la fin d'exécution de ses threads. Lequel des énoncés suivants est faux?

- L'accès à une variable globale créée par l'un des threads sera partagé par tous les threads.
- S'il s'agit de threads noyau, il pourrait y avoir plus d'un thread du processus en cours d'exécution sur des processeurs différents.
- Les threads peuvent communiquer entre eux sans passer par des appels système et ce, peu importe qu'il s'agisse de threads utilisateur ou de threads noyau.
- Typiquement, le temps de création de chaque thread sera plus élevé s'il s'agit de threads noyau, et moins élevé s'il s'agit de threads utilisateur.
- (●) Les registres du processus seront partagés entre les threads.

Votre réponse est correcte.

La réponse correcte est :
Les registres du processus seront partagés entre les threads.

Question **6**

Terminé

Note de 1,00 sur 1,00

Considérez les processus A, B et C suivants, où a; b1, b2, c1 et c2 sont des actions atomiques :

Sémaphore x = 0, y=1;

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

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

- (●) b1; b2; c1; c2; a;
- aucune de ces réponses.
- b1; b2; a; c1; c2;
- b1; c1; b2; a; c2;
- b1; b2; c1; a; c2;

Votre réponse est correcte.

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

Question **7**

Terminé

Note de 1,50 sur 1,50

Un processus exécute le code suivant :

```
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>

int x = 3;
int s;
int main() {
  int y = fork();
  if (y==0) _exit(x); // LIGNE A
   x = 2*x + y - wait(&s);
   printf("x=%d\n", x);
   _exit(5);
}
```

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

- Si l'appel à *fork* échoue, alors l'appel à *wait* va provoquer un interblocage.
- La valeur de *x* affichée par *printf* sera différente selon que l'appel à *fork* réussit ou échoue.
- Si l'appel à *fork* échoue, alors l'appel à *_exit* à la ligne A terminera le processus parent.
- Si l'appel à *fork* échoue, alors l'appel à *_exit* à la ligne A ne sera jamais exécuté.
- Si l'appel à *fork* échoue, alors l'appel à *wait* va retourner la valeur 0.

Votre réponse est correcte.

La réponse correcte est :
Si l'appel à *fork* échoue, alors l'appel à *_exit* à la ligne A ne sera jamais exécuté.

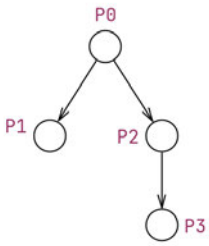
Question **8**

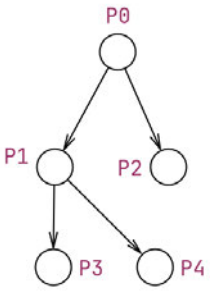
Terminé

Note de 2,50 sur 2,50

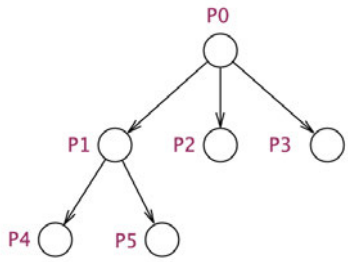
Quel arbre représente la hiérarchie de processus générés par la fonction suivante (où P0 est le processus appelant de la fonction construireArbre) :

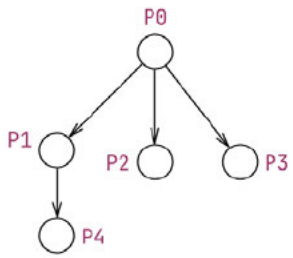
```
void construireArbre()
{
  for(int i=0; i<3; i++) {
    if(fork()==0) {
      if(i == 2) break;
      fork();
      break;
    }
  }
  while(wait(NULL)>0);
  _exit(0);
}
```

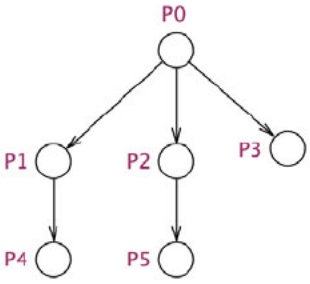



Aucune de ces réponses.

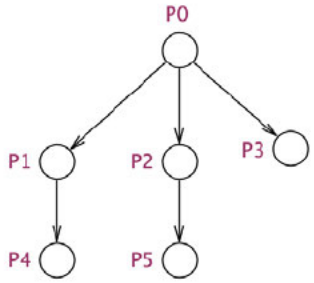





Votre réponse est correcte.

La réponse correcte est :



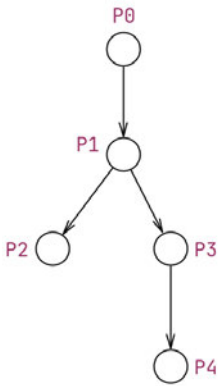
Question **9**

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*/
/*15*/  _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 ou P4).




Pour répondre à cette question, sélectionnez les instructions à insérer aux bons endroits.

| | |
|---|---|
| /*0*/ | if (fork()==0) { |
| /*1*/ | if (fork()==0) { |
| /*2*/ | execlp("echo", "echo", "P2",NULL); |
| /*3*/ | } |
| /*4*/ | if (fork()==0) { |
| /*5*/ | if (fork()==0) { |
| /*6*/ | execlp("echo", "echo", "P4",NULL); |
| /*7*/ | } |
| /*8*/ | while(wait(NULL)>0); |

| | |
|---|---|
| /*9*/ | _exit(0); |
| /*10*/ | } |
| /*11*/ | while(wait(NULL)>0); |
| /*12*/ | _exit(0); |
| /*13*/ | } |
| /*14*/ | while(wait(NULL)>0); |

Votre réponse est correcte.

La réponse correcte est :

/*0*/ → if (fork()==0) {,

/*1*/ → if (fork()==0) {,

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

/*3*/ → },

/*4*/ → if (fork()==0) {,

/*5*/ → if (fork()==0) {,

/*6*/ → execlp("echo", "echo", "P4",NULL);,

/*7*/ → },

/*8*/ → while(wait(NULL)>0);,

/*9*/ → _exit(0);,

/*10*/ → },

/*11*/ → while(wait(NULL)>0);,

/*12*/ → _exit(0);,

/*13*/ → },

/*14*/ → while(wait(NULL)>0);

Question **10**

Terminé

Note de 1,50 sur 1,50

Considérez le code suivant :

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

int main()
{
  int p[2];  pipe(p);
  if (fork()==0) { //
    close(p[0]); // PREMIER 'CLOSE'
    char A[100];
    sprintf(A, "Message X");
    write(p[1], A, strlen(A));
    close(p[1]); // DEUXIEME 'CLOSE'
    execlp("echo", "echo", "Message transmis.", NULL);
  }
  // LIGNE X
  close(p[1]); // TROISIEME 'CLOSE'
  char c;
  const char * const m = "Message recu: ";
  const char * const f = "Fin du programme.\n";
  write(1, m, strlen(m));
  while(read(p[0],&c,1)>0)
    write(1,&c,1);
  write(1,"\n",1);
  close(p[0]); // QUATRIEME 'CLOSE'
  while(wait(NULL)>0); // LIGNE DU 'WAIT'
  write(1, f, strlen(f));
  return 0;
}
```

Ce programme permet à un processus de transmettre un message à un autre processus qui l'enverra ensuite à la sortie standard. L'ordre d'arrivée à la console des affichages "Message recu: Message X" et "Message transmis" n'est pas considéré comme important.

Lequel des énoncés suivants est *faux*?

- Si on enlevait l'instruction *wait*, le programme pourrait fonctionner sans qu'un interblocage soit susceptible de survenir.
- Ce programme fonctionne tel quel sans qu'un interblocage soit susceptible de survenir.
- Si on déplaçait l'instruction *wait* à la LIGNE X, le programme pourrait fonctionner sans qu'un interblocage soit susceptible de survenir.
- Si on enlevait la deuxième instruction *close*, le programme pourrait fonctionner sans qu'un interblocage soit susceptible de survenir.
- **Si on enlevait la troisième instruction *close*, le programme pourrait fonctionner sans qu'un interblocage soit susceptible de survenir.**

Votre réponse est correcte.

La réponse correcte est :
Si on enlevait la troisième instruction *close*, le programme pourrait fonctionner sans qu'un interblocage soit susceptible de survenir.

Question **11**

Terminé

Note de 2,00 sur 2,00

Considérez le code suivant :

```
int main () {
  /*0*/
  if(fork()==0)  {  // premier fils
        /*1*/
        /*2*/
         char c;
         while (read(0,&c,1) >0)
              write(1,&c, 1);
         _exit(0);
   }
   if(fork()==0)  {  // second fils
         /*3*/
          /*4*/
         write(1, ”message du fils2\n”, 17);
         _exit(0);
   }
 /*5*/
 /*6*/
  write(1, ”message du pere\n”, 16);
 /*7*/
 /*8*/
 return 0;
}
```

Complétez le code pour que le premier processus fils récupère, via un tube nommé et les appels système read et write figurant dans le code, les messages de son père et de son frère. 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.

| | |
|---|---|
| Commentaire /*0*/ | mkfifo("inf2610",0600); |
| Commentaire /*1*/ | int fd=open("inf2610", O_RDONLY); |
| Commentaire /*2*/ | dup2(fd,0); close(fd); |
| Commentaire /*3*/ | int fd=open("inf2610", O_WRONLY); |
| Commentaire /*4*/ | dup2(fd,1); close(fd); |
| Commentaire /*5*/ | int fd=open("inf2610", O_WRONLY); |
| Commentaire /*6*/ | dup2(fd,1); close(fd); |
| Commentaire /*7*/ | close(1); |
| Commentaire /*8*/ | while (wait(NULL)>0); |

Votre réponse est correcte.

La réponse correcte est :
Commentaire /*0*/ → mkfifo("inf2610",0600);,

Commentaire /*1*/ → int fd=open("inf2610", O_RDONLY);,

Commentaire /*2*/ → dup2(fd,0); close(fd);,

Commentaire /*3*/ → int fd=open("inf2610", O_WRONLY);,

Commentaire /*4*/ → dup2(fd,1); close(fd);,

Commentaire /*5*/ → int fd=open("inf2610", O_WRONLY);,

Commentaire /*6*/ → dup2(fd,1); close(fd);,

Commentaire /*7*/ → close(1);,

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

Question **12**

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 SIGSTOP ne peut être masqué.
- La commande *signal(SIGINT, f)* assigne le gestionnaire de signal *f* au processus courant pour le signal SIGINT.
- La commande *signal (SIGINT, SIG_IGN)* assigne le gestionnaire de signal par défaut au processus courant pour le signal SIGINT.
- Le signal SIGKILL 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 **13**

Terminé

Note de 2,00 sur 2,00

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

**int NbL = 0;**

**Semaphore redact=1, mutex=1, tour =1;**

```
Redacteur() {
   while(1) {
        P(tour);
        P(redact);
        V(tour);
        ecrire();
        V(redact);
   }
}
```

```
Lecteur() {
   while(1) { P(tour);
        P(mutex);
        if (NbL == 0)  P(redact);
        NbL++;
        V(mutex);
        lire();
        V(tour);
        P(mutex);
        NbL--;
        if(NbL == 0) V(redact);
        V(mutex);
   }
}
```

Est-ce que cette modification peut altérer le fonctionnement des lecteurs et rédacteurs, 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.

- Aucune de ces réponses.
- Non, car les famines des écrivains vont continuer d'être possibles.
- Non, car les lecteurs vont pouvoir continuer de partager l'accès en lecture à la base de données.
- Oui, car il ne pourra y avoir plus d'un lecteur dans la base de données.
- Oui, car il y aura un risque d'interblocage.

Votre réponse est correcte.

La réponse correcte est :
Oui, car il ne pourra y avoir plus d'un lecteur dans la base de données.

Question **14**

Terminé

Note de 2,50 sur 2,50

Pour assurer l'exclusion mutuelle entre deux threads P0 et P1, on propose la solution suivante:

```
bool flag[2] = {false, false};

void P0()
{
  while(1) {
    flag[0] = true;
    while(flag[1]==true);
    section_critique(0);
    flag[0] = false;
  }
}

void P1()
{
  while(1) {
    flag[1] = true;
    while(flag[0]==true);
    section_critique(1);
    flag[1] = false;
  }
}
```

Il vous est demandé de vous prononcer sur le fait que cette solution est correcte ou non. À cette fin, complétez la phrase suivante:

Les deux threads [ne pourraient pas] se retrouver en même temps dans leurs sections critiques respectives. Par ailleurs, les processus [pourraient se retrouver tous les deux dans des attentes actives infinies]. En conclusion, la solution proposée est [incorrecte].

pourraient

pourraient se retrouver tous les deux dans des attentes passives infinies

ne seront pas sujets à la possibilité d'interblocage

correcte

Votre réponse est correcte.

La réponse correcte est :

Pour assurer l'exclusion mutuelle entre deux threads P0 et P1, on propose la solution suivante:

```
bool flag[2] = {false, false};

void P0()
{
  while(1) {
    flag[0] = true;
    while(flag[1]==true);
    section_critique(0);
    flag[0] = false;
  }
}

void P1()
{
  while(1) {
    flag[1] = true;
    while(flag[0]==true);
    section_critique(1);
    flag[1] = false;
  }
}
```

Il vous est demandé de vous prononcer sur le fait que cette solution est correcte ou non. À cette fin, complétez la phrase suivante:

Les deux threads [ne pourraient pas] se retrouver en même temps dans leurs sections critiques respectives. Par ailleurs, les processus [pourraient se retrouver tous les deux dans des attentes actives infinies]. En conclusion, la solution proposée est [incorrecte].