

**SI3 - Examen ProgSys - Durée 45 minutes - Aucun document autorisé**

<input type="checkbox"/>	0	<input type="checkbox"/>	0	<input type="checkbox"/>	0	<input type="checkbox"/>	0	<input type="checkbox"/>	0	<input type="checkbox"/>	0	<input type="checkbox"/>	0
<input type="checkbox"/>	1	<input type="checkbox"/>	1	<input type="checkbox"/>	1	<input type="checkbox"/>	1	<input type="checkbox"/>	1	<input type="checkbox"/>	1	<input type="checkbox"/>	1
<input type="checkbox"/>	2	<input type="checkbox"/>	2	<input type="checkbox"/>	2	<input type="checkbox"/>	2	<input type="checkbox"/>	2	<input type="checkbox"/>	2	<input type="checkbox"/>	2
<input type="checkbox"/>	3	<input type="checkbox"/>	3	<input type="checkbox"/>	3	<input type="checkbox"/>	3	<input type="checkbox"/>	3	<input type="checkbox"/>	3	<input type="checkbox"/>	3
<input type="checkbox"/>	4	<input type="checkbox"/>	4	<input type="checkbox"/>	4	<input type="checkbox"/>	4	<input type="checkbox"/>	4	<input type="checkbox"/>	4	<input type="checkbox"/>	4
<input type="checkbox"/>	5	<input type="checkbox"/>	5	<input type="checkbox"/>	5	<input type="checkbox"/>	5	<input type="checkbox"/>	5	<input type="checkbox"/>	5	<input type="checkbox"/>	5
<input type="checkbox"/>	6	<input type="checkbox"/>	6	<input type="checkbox"/>	6	<input type="checkbox"/>	6	<input type="checkbox"/>	6	<input type="checkbox"/>	6	<input type="checkbox"/>	6
<input type="checkbox"/>	7	<input type="checkbox"/>	7	<input type="checkbox"/>	7	<input type="checkbox"/>	7	<input type="checkbox"/>	7	<input type="checkbox"/>	7	<input type="checkbox"/>	7
<input type="checkbox"/>	8	<input type="checkbox"/>	8	<input type="checkbox"/>	8	<input type="checkbox"/>	8	<input type="checkbox"/>	8	<input type="checkbox"/>	8	<input type="checkbox"/>	8
<input type="checkbox"/>	9	<input type="checkbox"/>	9	<input type="checkbox"/>	9	<input type="checkbox"/>	9	<input type="checkbox"/>	9	<input type="checkbox"/>	9	<input type="checkbox"/>	9



← Veuillez noircir les cases correspondantes à votre Numéro Etudiant (NE) (1 case/ligne et 1 case/colonne). Pour NE=21056798, il faut noircir le 2 dans la 1ère colonne, le 1 dans la 2ème colonne etc ...

Ecrivez votre Nom et votre N° étudiant (NE)

.....
.....

Mauvais

Bon

A LIRE OBLIGATOIREMENT AVANT DE COMMENCER ...   Les cases doivent être remplies et non cochées pour assurer une bonne relecture par le système de correction automatique. Crayon de papier, stylo à bille, gomme et correcteur blanc autorisés.

Tricherie: Les questions et les réponses sont mélangées entre les copies! Ne cherchez pas à regarder sur la copie de votre voisin, cela ne sert à rien.

Correction: La correction de ce QCM se fera de manière automatique. Mais comme l'erreur est humaine, on vous demande d'écrire votre nom et numéro d'étudiant manuellement.

Barème: Toutes les questions comportent au moins une bonne réponse. Toute bonne réponse rapporte des points et toute mauvaise réponse en fait perdre. Les questions avec une * peuvent avoir une ou plusieurs bonnes réponses.

Q. 1 La manière de créer un processus est identique sous Unix et sous Windows:

- ☒ Faux
☐ Vrai

Q. 2 En considérant le morceau de code suivant, quelles sont les valeurs de *i* affichées par le processus père (P) et par le processus fils (F):

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

int main() {
    int i=0;
    pid_t father_pid = getpid();
    if (fork()) {
        i++;
    }
    i++;
    printf("%s: i=%d\n", (father_pid == getpid()) ? "PP" : "F ", i);
    wait(NULL);
}
```

- ☐ P: i = 2 et F: i = 1
☐ P: i = 1 et F: i = 1
☒ P: i = 2 et F: i = 2
☐ P: i = 1 et F: i = 2

Q. 3 Les deux codes suivants sont équivalents:

```
dup (fd) ;
close (1) ;

et

dup2 (fd, 1) ;
```

- ☒ Vrai
☐ Faux

Q. 4 Posix utilise des descripteurs de fichiers pour faire référence aux fichiers dans les fonction read, write, close (et ce qui est retourné par *open*). Ce descripteur de fichier est:

- ☒ Un nombre entier
☐ Un tableau de caractères
☐ Un tube
☐ Une chaîne de caractères contenant le nom du fichier (chemin et nom de fichier)

Q. 5 Posix est une norme:

- ☐ D'interface de communication entre machines
☐ D'interface de la bibliothèque C
☒ D'interface direct du noyau



Q. 6 * Quelle(s) affirmation(s) sur les tubes nommés est(sont) valide(s)?

- ☒ Le tube nommé permet de communiquer entre des processus sans parenté directe
- ☒ Le tube nommé doit obligatoirement être créé dans le père avant la création du fils
- ☐ Le tube nommé est un fichier comme les autres
- ☒ Le tube nommé apparaît dans le système de fichier
- ☐ Le tube nommé permet exclusivement la communication entre un père et son fils

Q. 7 Les commandes Unix comme *ls*, *cd*, *cp*, *mv*, *rm*, ... sont programmées en C à l'aide des primitives Posix de gestion des entrées-sorties?

- ☐ Non
- ☒ Oui

Q. 8 Le mécanisme de chargement de bibliothèques dynamiques sous Linux et Windows est complètement différent?

- ☒ Non
- ☐ Oui

Q. 9 * La synchronisation des threads est nécessaire car:

- ☐ Tous les threads d'un processus partagent la même pile d'exécution
- ☒ Tous les threads d'un processus partagent les mêmes fichiers
- ☒ Tous les threads d'un processus partagent les mêmes variables globales
- ☐ Tous les threads d'un processus partagent le même espace d'adressage

Q. 10 * Soit le tube créé par la commande `int pipe(int fd[2])`. Les extrémités d'un tube sont:

- ☐ `fd[0]` sortie du tube (où on lit)
- ☒ `fd[0]` entrée du tube (où on écrit)
- ☒ `fd[1]` sortie du tube (où on lit)
- ☐ `fd[1]` entrée du tube (où on écrit)

Q. 11 Windows fournit des modalités plus avancées de communications inter-processus à l'aide des tubes que sous Unix?

- ☒ Vrai
- ☐ Faux

Q. 12 * Sous Unix, qu'est ce qui est manipulable via les primitives Posix comme un simple fichier (avec les fonctions *open*, *read*, *write*, *close*)?

- ☐ Un processus
- ☐ L'entrée, la sortie standard et la sortie standard d'erreur
- ☒ Un tube nommé
- ☐ Un tube anonyme
- ☒ Un dossier/répertoire

Q. 13 * Un Makefile sert principalement:

- ☒ À décrire comment compiler un projet
- ☐ À décrire comment charger les bibliothèques pour exécuter un programme
- ☐ À faire de la compilation séparée
- ☒ À ne recompiler que les fichiers sources qui ont été modifiés et à recréer les bibliothèques et exécutables qui en dépendent

Q. 14 * Par défaut, un processus possède les descripteurs de fichiers suivants:

- ☒ Entrée standard
- ☒ Sortie standard
- ☐ Entrée standard d'erreur
- ☐ Le dossier racine du système de fichiers
- ☒ Sortie standard d'erreur

Q. 15 * La mise en oeuvre des threads au niveau du système d'exploitation peut se faire:

- ☐ En espace noyau (Kernel Space)
- ☒ Dans la pile d'exécution
- ☒ En espace utilisateur (User Space)

Q. 16 * Soit la fonction suivante. Que provoquera cette fonction?

```
int *f() {  
    int x = 42;  
    return &x;  
}
```

- ☐ Une erreur à l'édition de liens
- ☒ Une erreur à la compilation
- ☐ Un avertissement à la compilation
- ☐ Un problème à l'exécution (comportement indéfini)

Q. 17 Quand je lie une bibliothèque statique à un programme C

- ☐ Tous les `.o` de la bibliothèque qui ont au moins une fonction utilisée par mon programme sont ajoutés à l'exécutable
- ☐ Toutes les fonctions de la bibliothèque sont ajoutées à mon exécutable
- ☒ Seules les fonctions utilisées par mon programme sont ajoutées à l'exécutable

Q. 18 Pour un exécutable donné, il y a moyen de connaître tous les symboles définis (nom des fonctions, variables globales, ...) y compris celles issues des bibliothèques statiques utilisées?

- ☒ Oui, grâce à la commande `ldd`
- ☐ Non
- ☐ Oui, grâce à la commande `nm`



Q. 19 Soit deux fichiers objet *file1.o* *file2.o* et soit une bibliothèque statique créée avec ces deux fichiers *.o*. Un programme *prog1.exe* utilisant la bibliothèque statique et le *prog2.exe* créé en incluant les deux *.o* auront **obligatoirement** une taille:

- ☐ *prog1.exe* = *prog2.exe*
- ☐ *prog1.exe* > *prog2.exe*
- ☒ *prog1.exe* <= *prog2.exe*

Q. 20 * Qu'est ce qu'un système d'exploitation?

- ☒ Une interface entre le matériel et les programmes utilisateurs
- ☐ Une collection de programmes qui gèrent les ressources matérielles
- ☒ Un fournisseur de services pour les programmes utilisateurs

Q. 21 Il est possible de se déplacer dans les données fournies par un tube (fonction *lseek*):

- ☐ Non
- ☒ Oui

Q. 22 * En Posix, à l'aide d'un descripteur de fichier vous pouvez accéder à:

- ☐ A des données issues de la carte réseau
- ☒ Aux données issues du clavier
- ☒ Un fichier sauvegardé sur un support de stockage quelque soit le formatage
- ☒ La sortie standard d'un processus

Q. 23 Un Processus est:

- ☐ Un programme
- ☐ L'état d'un processeur (un contexte processeur)
- ☐ Un processeur capable d'exécuter un programme
- ☒ Une instance de programme en cours d'exécution

Q. 24 * Quelle(s) est(sont) la(les) caractéristique(s) des tubes qui est(sont) correcte(s):

- ☐ Les tubes nommés sont similaires sous Unix et Windows
- ☒ Les tubes anonymes sont similaires sous Unix et Windows
- ☒ Un tube anonyme présente un coût moins important qu'un tube nommé
- ☐ Un tube anonyme est mis en oeuvre grâce à une zone de mémoire partagée
- ☒ Un tube anonyme est unidirectionnel
- ☐ Un tube nommé sous Unix permet une communication entre machines distinctes

Q. 25 Si deux threads écrivent la même variable entière sans synchronisation avec deux valeurs différentes, le résultat sera:

- ☐ Une valeur aléatoire autre que celles écrites
- ☐ Une des deux écritures, mais sans pouvoir déterminer laquelle
- ☒ Un mélange des deux valeurs, chaque octet venant aléatoirement d'une des deux valeurs écrites
- ☐ La valeur écrite par le premier thread créé (celui qui aura le tid le plus petit)

Q. 26 Quand j'appelle la fonction *exit*:

- ☐ Seul le premier thread du processus est arrêté
- ☐ Aucun thread n'est arrêté dans le processus
- ☐ Tous les threads sont arrêtés
- ☒ Seul le thread appelant la fonction *exit* est stoppé

Q. 27 Un processus, autre que le processus *init*:

- ☒ N'a qu'un seul processus parent
- ☐ Peut avoir 2 processus parents
- ☐ Peut ne pas avoir de processus parent

Q. 28 Windows fournit la possibilité de réaliser des communications inter-processus à l'aide de tubes anonymes et nommés comme sous Unix.

- ☐ Faux
- ☒ Vrai

Q. 29 * Quelles commandes permettent de créer une bibliothèque:

- ☐ *nm*
- ☐ *ldd*
- ☐ *ar*
- ☒ *gcc*

Q. 30 Le noyau _____ des threads de l'espace utilisateur ?

- ☐ est le créateur
- ☒ est au courant de l'existence
- ☐ n'est pas au courant de l'existence



Q. 31 * Quelle exécution de commande est équivalente au code Posix suivant (*foo* et *bar* sont des fichiers):

```
char buffer[MAX];
int n;
int fd1 = open("bar", O_WRONLY|O_TRUNC|O_CREAT, 0600);
int fd2 = open("foo", O_RDONLY|O_EXCL);

while((n = read(fd2, buffer, MAX)) != 0)
    write(fd1, buffer, n);
close(fd1);
close(fd2);
```

- ☐ cp bar foo
- ☐ mv foo bar
- ☒ cp foo bar
- ☐ cat foo > bar
- ☐ cat foo bar
- ☐ cp -R foo bar

Q. 32 * Quelle(s) affirmation(s) sont vraie(s)?

- ☐ Le processus parent connaît le processus enfant qu'il a engendré
- ☒ Le processus parent partage ses fichiers ouverts avec le processus enfant engendré
- ☐ Le processus parent peut communiquer avec son processus enfant
- ☐ Le processus parent partage son espace d'adressage avec son processus enfant

Q. 33 * Un système d'exploitation est composé:

- ☐ D'un ensemble de programmes utilitaires
- ☒ D'un noyau
- ☐ De bibliothèques

Q. 34 Une bibliothèque partagée .dll peut être utilisée pour tout exécutable sous Linux?

- ☒ Non ☐ Oui

Q. 35 * Un processus écrivain dans un tube anonyme doit fermer le descripteur d'écriture (*fd[1]*):

- ☐ Avant de commencer à écrire dans le tube
- ☐ Pour que le lecteur puisse commencer à lire
- ☐ Pour que le lecteur sache qu'il a atteint la fin des données
- ☐ Après avoir fait la dernière écriture dans le tube

Q. 36 L'adresse de la prochaine instruction qui sera exécutée par le processus courant est fournie par:

- ☒ Le "Program Counter" (ou "pointeur dans le code")
- ☐ Les registres du processeurs
- ☐ Le tube
- ☐ La pile du processus

Q. 37 * La pile d'un processus contient:

- ☐ Les variables locales
- ☐ Le PID du processus
- ☒ Les paramètres d'appel d'une fonction
- ☒ Les adresses de retour

Q. 38 Le temps nécessaire pour créer un thread dans un processus existant est:

- ☐ Plus grand que le temps nécessaire pour créer un nouveau processus
- ☒ Plus petit que le temps nécessaire pour créer un nouveau processus
- ☐ Équivalent au temps requis pour créer un nouveau processus

Q. 39 * Quelles sont les options de compilation de *gcc* nécessaires lors des différentes étapes de création d'une bibliothèque dynamique:

- ☐ -Wall ☒ -shared ☐ -fpic ☐ -g

Q. 40 * Un thread:

- ☐ Est composé de plusieurs processus
- ☐ S'exécute sur plusieurs coeurs ou processeurs, simultanément
- ☐ Partage le même espace de mémoire avec tous les threads composant son processus
- ☒ Possède sa propre pile d'exécution
- ☐ Partage le même espace de mémoire avec tous les processus composant son thread

Q. 41 * Le format ELF est un format permettant d'organiser le code:

- ☐ Du code source du programme
- ☐ D'un programme
- ☐ D'une bibliothèque partagée
- ☐ D'une bibliothèque statique

Q. 42 * Que me permet la commande suivante: *ldd prog.exe*

- ☒ Vérifier les bibliothèques dynamiques utilisées par le programme
- ☐ Vérifier les bibliothèques statiques utilisées par le programme
- ☐ Debugger le programme
- ☒ Créer une bibliothèque dynamique nommée *prog.exe*

Q. 43 La notion de thread existe sous Unix mais pas sous Windows:

- ☒ Faux
- ☐ Vrai



Q. 44 * Il existe plusieurs modes de gestion des entrées-sorties pour un système d'exploitation:

- ☐ Mode programmé simple (boucle d'attente active)
- ☐ Mode par canal DMA (Direct Memory Access)
- ☐ Mode programmé par interruption
- ☐ Mode par vol de priorité au processus

Q. 45 * Il est possible de faire les combinaisons suivantes:

- ☐ Bibliothèque statique, édition de liens dynamique
- ☐ Bibliothèque dynamique, édition de liens statique
- ☐ Bibliothèque statique, édition de liens statique
- ☐ Bibliothèque dynamique, édition de liens dynamique

Q. 46 * Après l'exécution de la fonction *exec*, il est possible de mettre du code pour gérer:

- ☐ La suite du programme
- ☐ Le code de retour de la fonction *exec* qui se serait mal passé
- ☐ La suite du programme dans le processus fils

Q. 47 * Si on veut faire communiquer deux processus à l'aide d'un tube anonyme, on doit créer le tube:

- ☐ Après l'appel à la fonction *exec*
- ☐ Avant l'appel à la fonction *exec*
- ☐ Avant l'appel à la fonction *fork*
- ☐ Après avoir fait l'appel à la fonction *fork*

Q. 48 * Quelle est l'utilité de la commande *ranlib*?

- ☒ Faire l'édition de liens d'une bibliothèque statique
- ☐ Créer une bibliothèque dynamique
- ☐ Ajouter un index à une bibliothèque statique

Q. 49 * Un debugger permet:

- ☒ D'exécuter un programme pas à pas
- ☒ De trouver facilement la ligne de code sur laquelle un programme fait une violation mémoire
- ☐ De trouver automatiquement les bugs d'un programme
- ☐ De modifier le fil d'exécution du programme
- ☒ D'afficher les valeurs de variables pendant l'exécution

Q. 50 * Un thread partage ses ressources avec:

- ☐ Les threads similaires qui appartiennent à d'autres processus
- ☐ Les autres threads appartenant au même processus
- ☐ Les autres processus possédant des threads

Q. 51 * L'ouverture d'un tube nommé est bloquante (par défaut). Un problème pourrait-il survenir si l'ouverture est non bloquante et que le processus tente de lire ou d'écrire dans le tube juste après l'ouverture?

- ☐ Non (aucun problème)
- ☐ Oui, si le processus est écrivain, il pourrait être tué par le signal SIGPIPE dès la première tentative d'écriture
- ☐ Oui, si le processus est lecteur, il pourrait obtenir une fin de fichier dès la première tentative de lecture

Q. 52 * En C/Posix, le chargement de bibliothèque dynamique avec édition de liens dynamique est utilisé pour:

- ☒ Pour changer dynamiquement l'implémentation de fonctions
- ☐ Pour rendre le programme plus rapide
- ☒ Pour la mise en oeuvre de plugins dans des programmes
- ☐ Pour permettre l'exécution du programme sous plusieurs OS (Windows, Linux, ...)

Q. 53 * Juste après un appel à *fork*, quel(s) élément(s) peuvent différer entre le processus père et le processus fils ?

- ☐ Les variables locales
- ☐ Les fichiers ouverts
- ☐ Le programme exécuté
- ☒ Le PID

Q. 54 * Un debugger permet de revenir en arrière dans l'exécution:

- ☒ Non
- ☐ Oui

Q. 55 * Qu'est ce qui est vrai parmi les affirmations suivantes?

- ☒ Le noyau gère le cycle de vie de tous les processus
- ☐ Les différents programmes utilitaires du système d'exploitation (*cp*, *rm*, *mv*, ...) sont programmés en assembleur
- ☒ Le noyau est le programme qui constitue le coeur d'un système d'exploitation

Soit le code suivant:

```
int var1;
char var2[] = "buf1";
main() {
    int var3;
    static int var4;
    static char var5[] = "buf2";
    char * var6;
    var6 = malloc(512);
}
```



Q. 56 * Quelles sont les variables qui seront dans la pile?

- ☐ var1
- ☐ var2
- ☒ var3
- ☐ var4
- ☐ var5
- ☒ var6

Q. 57 Si un thread ouvre un fichier avec le droit de lecture:

- ☐ Aucun autre thread ne peut lire depuis ce descripteur de fichier
- ☒ Les autres threads du même processus peuvent lire depuis ce descripteur de fichier
- ☐ Les threads d'un autre processus peuvent lire depuis ce descripteur de fichier

Soit le code suivant que nous utiliserons pour les deux questions suivantes:

```
#include <stdio.h>
#include <unistd.h>

int main() {
    int n = ;
    while (n > 0) {
        fork();
        fork();
        n--;
        wait(NULL);
        wait(NULL);
    }
    printf("%d\n", getpid());
}
```

Q. 58 Quel est le nombre de *pid* qui seront affichés par ce code avec *n* initialisé à 2:

- ☒ 4
- ☐ 8
- ☐ 16
- ☐ 32

Q. 59 Quel est le nombre de *pid* qui seront affichés par ce code, avec *n* initialisé à 1:

- ☒ 2
- ☐ 3
- ☐ 4
- ☐ > 4

Q. 60 * Quelle(s) est(sont) l'(les) erreur(s) qui est(sont) gérée(s) par le système d'exploitation?

- ☒ Le débordement de la pile d'exécution d'un processus
- ☐ Une défaillance matérielle (ex: impossible d'écrire sur une carte SD défectueuse)
- ☒ La violation d'accès à une zone mémoire