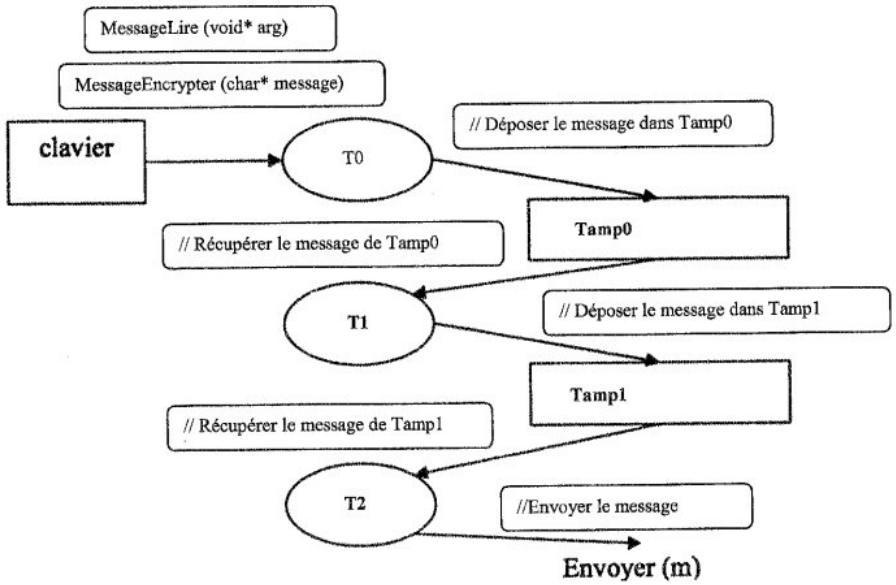


En respectant les données énoncées, complétez le code.

Institut Supérieur d'Informatique et de Mathématiques de Monastir ISIMM

Examen – S2 – 2023/2024

Filière : *1ère LInfo*	**Matière : Système d'exploitation 2**		**Enseignant : Sana BENZARTI**
Date : 22 / 05 / 2024	Nbr de Crédits : **4**	Coefficient : **2**	**Documents autorisés : Non**
Durée de l'examen : 1h30	Régime d'évaluation : **Mixte / CC**		**Nombre de pages : 08**
	EX (70%) + DS (10%) + TP (20%)		

Exercice 1 : QCM

1- Que renvoie cette commande : execlp("find", "find", "/home/rep", "-type", "f", NULL);

a- Rechercher tous les répertoires dans le dossier rep
b- Rechercher tous les fichiers dans le dossier rep
c- Rechercher tous les répertoires avec un type spécifique f

2- Que renvoie cette commande :
char *args[] = {"grep", "lse$", "/etc/passwd", NULL}
execv("/bin/grep", args);

a- Chercher la chaine "lse" sous /etc/passwd
b- Chercher la chaine "lse" sous /bin/grep
c- Chercher les lignes qui se terminent par "lse" sous /bin/grep
d- Chercher les lignes qui se terminent par "lse" sous /etc/passwd

3- Syntaxe du Passage d'arguments avec exec:

a- execlp(argv[1], argv[1], NULL)
b- execlp(argv[1], &argv[1], NULL)
c- execv(&argv[1], argv[1])
d- execvp(argv[1], &argv[1])

4- La famille execl comporte

a- execl
b- execle
c- execlep
d- execlp
e- execel

5- La famille execv comporte

a- execev
b- execvp
c- execev
d- execve
e- execpe

6- Lorsque la commande exec est utilisée

a- Un nouveau processus est créé
b- Le processus en cours d'exécution change de contexte
c- Le processus en cours d'exécution quitte le programme en cours d'exécution
d- Chaque instruction après la commande "exec" n'est plus exécutée

7- 2 threads sont créés, leurs TID peuvent être

a- De type %d
b- De type %ld
c- Un TID égal a 0176538680
d- Un TID égal a 2055

8- WIFEXITED est une macro utilisée

a- Avec la fonction exit
b- Avec la fonction fork
c- Avec la fonction wait

9- WIFEXITED(status) et WEXITSTATUS(status) nous indique

a- Le bon fonctionnement du programme
b- La liste des processus
c- Un signal de terminaison
d- La façon dont le processus s'est terminé

10- WIFSIGNALED(status) et WTERMSIG(status) nous indique

a- Un signal extérieur envoyé vers le processus
b- Une erreur dans le code
c- Une fin urgente pour le processus
d- Une terminaison immédiate en utilisant un signal SIGKILL et un code de retour égal a 9
e- Une terminaison propre en utilisant un signal SIGTERM et un code de retour égal a 15

11- pthread_create(&thread1, NULL, fct_th, (void*)tab) utilise un argument de type

a- Un entier
b- Un tableau d'entiers

12- pthread_create(&thread1, NULL, fct_th, (void*)&tab) utilise un argument de type
a- Un entier
b- Un tableau d'entiers

13- pthread_create(&thread1, NULL, fct_th, (int*)(intptr_t)tab) utilise un argument de type

a- Un entier
b- Un tableau d'entiers

14- Le heap ou le tas est

a- Une zone mémoire statiquement allouée à l'exécution pour stocker des données qui sont créées et détruites de manière statique
b- Une zone de mémoire dynamique allouée à l'exécution pour stocker des données qui sont créées et détruites de manière dynamique
c- C'est une pile

15- Les threads partagent

a- La mémoire Heap
b- Les données entre eux
c- La pile
d- Le processus

Exercice 2: Les threads

1- Rappeler le cycle de vie d'un thread

2- Ecrire un programme en c ayant le comportement suivant :
- Des threads sont créées en utilisant **un tableau de threads** (leur nombre étant **passé en paramètre** lors du lancement du programme principal) ;
- Chaque thread affiche le message **"FREE PALESTINE !"** en utilisant **le passage d'argument avec la fonction exécutée** par le thread. Utiliser **un pointeur spécifique de type char*.**
- Le thread principal attend la terminaison des différents threads créées.

Exercice 3: Les Mutex

On considère un système multicouche composé de trois couches T0, T1 et T2. Les couches sont des threads concurrents qui communiquent au moyen de deux tampons Tamp0 et Tamp1 de même taille N :

- Thread T0 et Thread T1 partagent le tampon Tamp0
- Thread T1 et Thread T2 partagent le tampon Tamp1.

Chaque couche se charge d'un traitement particulier :

Le thread T0 se charge de lire du clavier et d'encrypter des messages avant de les déposer dans le tampon Tamp0 via la fonction: MessageLire (void* arg) et MessageEncrypter (char* message)

Le thread T1 se charge de transférer directement les messages du tampon Tamp0 vers le tampon Tamp1. Le thread T2 récupère les messages du tampon Tamp1 pour les envoyer à un destinataire.

La figure ci-dessous explique le principe d'interaction entre les threads et les tampons.

```
    // Déposer le message dans Tamp0
    .....//Utiliser mutex

    // Code pour déposer le message dans Tamp0

    printf("T0 : Message traité déposé dans Tamp0\n");

    ..... //Utiliser mutex
    pthread_exit(NULL);
  }
}
```

```
// Fonction exécutée par le Thread T1

void *T1(void *arg) {
  while (1) {
    // Récupérer un message de Tamp0
    ..... //Utiliser mutex
    // Code pour récupérer un message de Tamp0

    printf("T1 : Message récupéré depuis Tamp0\n");
    ..... //Utiliser mutex

    // Transférer le message vers Tamp1
    ..... //Utiliser mutex
    // Code pour transférer le message vers Tamp1
    printf("T1 : Message transféré vers Tamp1\n");
    ..... //Utiliser mutex
    pthread_exit(NULL);
  }
}
```

```
// Fonction exécutée par le thread T2

void *T2 (void *arg) {
  while (1) {
    // Récupérer un message de Tamp1
    ..... //Utiliser mutex
    // Code pour récupérer un message de Tamp1
    printf("T2 : Message récupéré depuis Tamp1\n");
    ..... //Utiliser mutex

    // Envoyer le message à un destinataire
    // Code pour envoyer le message
    printf("T2 : Message envoyé\n");
    pthread_exit(NULL);
  }
}
```


Institut Supérieur d'Informatique et de Mathématiques de Monastir **ISIMM**

Examen – S2 – 2023/2024

Filière : *1ère LInfo*	**Matière : Système d'exploitation 2**		**Enseignant : Sana BENZARTI**
Date : 22 / 05 / 2024	Nbr de Crédits : **4**	Coefficient : **2**	**Documents autorisés : Non**
Durée de l'examen : 1h30	Régime d'évaluation : **Mixte / CC** EX (70%) + DS (10%) + TP (20%)		**Nombre de pages : 08**
Nom & Prénom :			**Matricule :** .....
Signature :	**Code confidentiel :**		**Classe :** ..... **N° Place :** .....

NOTE : Répondre directement sur les feuilles de l'examen /

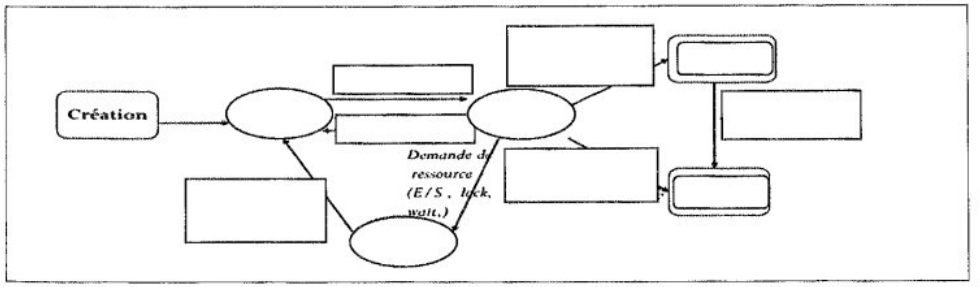
Note /20

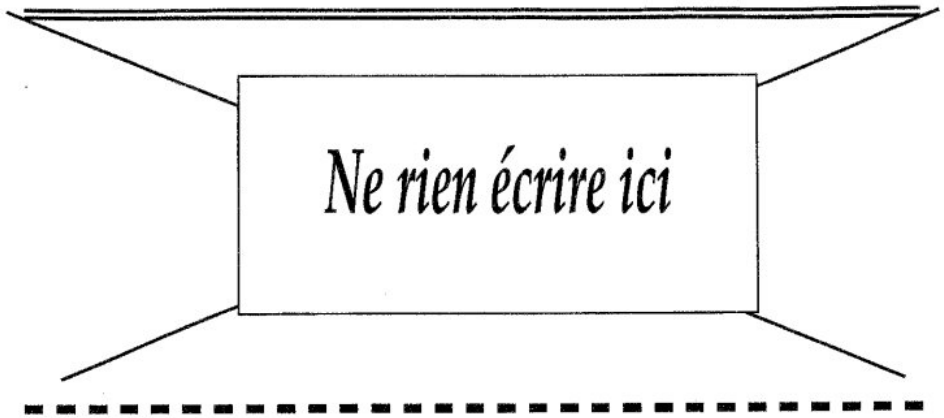
Exercice 1 : QCM

1-	
2-	
3-	
4-	
5-	
6-	
7-	
8-	
9-	
10-	
11-	
12-	
13-	
14-	
15-	

Exercice 2: Les threads

1- Cycle de vie des threads





2- Écrire le code en C:

Exercice 3: Les Mutex

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define N 100 // Taille des tampons

char Tamp0[N]; // Tampon Tamp0 partagé entre T0 et T1
```



```
char Tamp1[N]; // Tampon Tamp1 partagé entre T1 et T2

pthread_mutex_t mutex_T0 ;
pthread_mutex_t mutex_T1 ;
```

```
int main() {
   pthread_t thread_T0, thread_T1, thread_T2;

   pthread_mutex_init(.....);
   pthread_mutex_init(.....);

   pthread_create(&thread_T0, NULL, T0, NULL); //.....
   pthread_create(&thread_T1, NULL, T1, NULL);//.....
   pthread_create(&thread_T2, NULL, T2, NULL);//.....
   pthread_join(thread_T0, NULL);//.....
   pthread_join(thread_T1, NULL);//.....
   pthread_join(thread_T2, NULL) ;//.....

pthread_mutex_destroy(.....);
pthread_mutex_destroy(.....);

   return 0;
}
```

```
// Fonction pour encrypter un message
void MessageEncrypter(char *message) {
   // Code d'encryption (Exemple: décalage de
   while (*message != '\0') {                    6/8
      if (*message >= 'a' && *message <= 'z')
         *message = ((*message - 'a' + 1) % 26) + 'a'; // Décalage d'un caractère vers la droite
      else if (*message >= 'A' && *message <= 'Z')
         *message = ((*message - 'A' + 1) % 26) + 'A'; // Décalage d'un caractère vers la droite
      message++;
   }
}

// Fonction pour lire un message du clavier
void *MessageLire(void *arg) {
   .....  // Conversion d'un pointeur générique vers un pointeur spécifique
   printf("Entrez un message : ");
   .....// Lire un message avec scanf
   return NULL;
}
```

```
// Fonction exécutée par le thread T0

void *T0 (void *arg) {
   char message[N];
   pthread_t thread;

   while (1) {
      pthread_create(....., ....., MessageLire,.....);
      pthread_join(.....);

      MessageEncrypter(message);
```