	<p>IUT - Université de Bordeaux Département Informatique</p> <p>DS M3.101 - Programmation Système</p>	<p>20 Janvier 2021</p> <p>Groupe</p> <p>Nom</p> <p>Prénom</p>
---	--	---

Épreuve en temps limité (1h30). Documents non autorisés.
Une annexe est donnée en fin de sujet rappelant les principales méthodes.

1 Appels système

Sur un système multi-tâche généraliste, le code du système d'exploitation est situé dans un espace mémoire qui lui est propre, et qui n'est pas accessible par les processus utilisateur.

- 1** Décrire le moyen par lequel un programme peut faire exécuter un **appel système**.

2 Multi-traitement

On suppose qu'il y a en mémoire deux processus A et B, qui répètent chacun une boucle calcul + entrée sortie bloquante. La différence est que A calcule beaucoup (30 ms) contrairement à B (5 ms). Les opérations d'entrée-sortie qui ont lieu sur des périphériques indépendants durent 15 ms.

- 2** Complétez le graphique ci-dessous qui montre l'utilisation des différents organes (processeurs, périphériques) au cours du temps, par tranches de 5 ms.

CPU _ _ _ _ _
PA _ _ _ _ _
PB _ _ _ _ _

- 3** Même question, en supposant qu'on a un système de type **préemptif**, avec un quantum de temps fixé à 20 ms.

CPU _ _ _ _ _
PA _ _ _ _ _
PB _ _ _ _ _

4 Qu'apporte la préemption ?


3 Mémoire paginée

Considérons une machine des années 60, avec des mots (données et adresses) de 16 bits. Les pages contiennent 256 mots.

5 Expliquez comment en déduire le nombre de pages maximum ?

6 Montrez comment on détermine le numéro de page d'une adresse comme 0x1234 (on utilise l'hexadécimal par commodité)

7 Quelles sont la première et la dernière adresse de la page 0 ?

	<p>IUT - Université de Bordeaux Département Informatique</p> <p>DS M3.101 - Programmation Système</p>	<p>20 Janvier 2021</p> <p>Groupe</p> <p>Nom</p> <p>Prénom</p>
---	--	---

4 Processus lourds

On considère le code suivant.

Listing 1 – "fork.c"

```

1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main()
5  {
6      for (int i = 0; i < 2; i++) {
7          fork();
8      }
9      printf(" pid _=%d\n", getpid());
10 }
```

8 Indiquez combien de ligne(s) avec pid seront affichées? Expliquer ce résultat en dessinant l'arbre de création des processus (un noeud est un processus et une arête relie un père à son fils). On supposera le programme est le processus 1000, et que les numéros sont alloués successivement.

5 Processus légers

On propose le code suivant, dans lequel deux processus fils communiquent un nombre aléatoire à leur père via des tubes. Le père en calcule la somme.

Listing 2 – "deuxfils.c"

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #define VALEUR_MAX 20
4
5 int main()
6 {
7     int tube[2][2], tirages_fils[2], i, somme, tirage;
8
9     for (i = 0; i < 2; i++)
10     {
11         pipe(tube[i]);
12         if (fork() == 0)
13         {
14             if (i==1) {
15                 close(tube[0][0]);
16                 close(tube[0][1]);
17             }
18             close(tube[i][0]);
19             tirage = nombreAleatoire();
20             write (tube[i][1], &tirage, sizeof(int));
21             close(tube[i][1]);
22             exit(EXIT_SUCCESS);
23         }
24     }
25
26     somme = 0;
27     for (i = 0; i < 2; i++)
28     {
29         close (tube[i][1]);
30         read (tube[i][0], &tirages_fils[i], sizeof(int));
31         close (tube[i][0]);
32         somme += tirages_fils[i];
33     }
34
35     printf ("La somme vaut %d\n", somme);
36     exit(EXIT_SUCCESS);
37 }
38
39 int nombreAleatoire()
40 {
41     srand (time (NULL) + getpid());
42     return (rand () % VALEUR_MAX) + 1;
43 }

```

9 Ecrivez un programme qui réalise la même tâche en utilisant des processus légers (threads) écrivant dans une variable partagée. Il n'est pas nécessaire d'écrire les "#include", de réécrire la fonction "nombreAleatoire()", ni de traiter les erreurs d'exécution liées à l'utilisation des threads.

6 Redirections et tubes

On veut écrire des programmes qui produisent des images au format PNG (portable network graphics). Pour cela, on dispose de fonctions comme celle ci-dessous, qui produisent des image sous forme de texte au format PNM (portable pixmap) P3.

Listing 3 – "make_p3_rectangle.c"

```
1 /**
2  * Output a colored rectangle in P3 format to the standard output.
3  *
4  * @param width
5  * @param height
6  * @param colors    max color level
7  * @param red       red component (0 to colors)
8  * @param green     green level (0 to colors)
9  * @param blue      blue level (0 to colors)
10 */
11
12 void make_p3_rectangle(int width, int height,
13                       int colors,
14                       int red, int green, int blue) {
15     // header
16     printf("P3\n%d_%d\n%d\n", width, height, colors);
17     // body
18     for(int i = 0 ; i < width*height; i++) {
19         printf("%d_%d_%d\n", red, green, blue);
20     }
21 }
```

Si on effectue l'appel suivant :

```
// pink rectangle 200x100, 15 colors #F6B
make_p3_rectangle(200, 100, 15, 15, 6, 11);
```

10 quelle est l'entete produite par la fonction ? que contiennent les lignes suivantes ?

On considère maintenant la séquence suivante :

```
int ppm = open("output.ppm", O_WRONLY | O_CREAT | O_TRUNC, 0644);
...
make_p3_rectangle(200, 100, 15, 15, 6, 11);
```

11 Que faut-il ajouter à ce code pour que l'image soit produite dans le fichier `output.ppm` ? (on utilisera telle quelle `make_p3_rectangle`, sans aucune modification)

On veut maintenant produire un fichier PNG, sans passer par un fichier intermédiaire. Pour cela :

- le programme crée un processus fils qui redirige la sortie de `make_p3_rectangle` dans un tuyau et se termine.
- puis il lance la commande `pnmtopng` sans paramètres, en prenant son entrée dans le tube, et en redirigeant sa sortie vers le fichier `output.png`.

Indication :

```
execl("/usr/bin/pnmtopng", "pnmtopng", (char *) 0);
```

12 Ecrivez le code.

La commande `pnmtopng` produit, sur la sortie d'erreur, des messages que l'on juge indésirables, par exemple :

```
pnmtopng: 1 colors found
```

13 Comment les rediriger vers `/dev/null` ?

Annexe, rappels

Processus légers

- un thread est lancé au moment de sa création :
`pthread_create(pthread_t *tid, attributs, fonction, void *args).`
Le prototype de la fonction est : `void * fonction(void *).`
Le paramètre `attributs` sera non renseigné (`NULL`).
- le thread se termine à la sortie de la fonction.
- la fonction `pthread_join(pthread_t tid, void **value_ptr)` attend la fin du thread.

Processus, signaux

- l'appel de fonction `fork()` crée un processus fils identique au processus courant, et partageant les descripteurs ouverts, les signaux etc. Il retourne 0 au processus fils, et le numéro du processus fils au processus père.
- l'appel `wait(adresse)` attend la fin d'un des processus, dont il retourne le numéro (de type `pid_t`). Le paramètre est l'adresse d'un entier, où est placé le *status* du processus.
- la fonction `getpid()` permet à un processus de connaître son numéro, `getppid()` celui de son père.
- l'appel `waitpid(pid, adresse, 0)` attend la fin du processus indiqué
- l'appel de `signal(sig, fonction)` définit le *handler* pour le signal indiqué.
- `kill(pid, sig)` envoie un signal.

Lecture et écriture de bas niveau

- l'appel système `read(fd, adresse, taille)` retourne le nombre d'octets lus sur le *file descriptor* `fd` et placés dans le tampon à l'adresse et avec la taille indiquée. Il retourne un nombre négatif ou nul à la fin ou en cas d'erreur.
- l'appel système `write(fd, adresse, taille)` transmet les octets contenus dans le tampon indiqué.
- `fd=open(chemin, options)` ouvrir un descripteur sur le fichier donné par `chemin` et avec les options : (`O_RDONLY`, `O_WRONLY`, `O_RDWR`).
- `dup2(old, new)` duplique le descripteur `old` dans `new`, en ayant d'abord fermé ce dernier. Les opérations que l'on fera ensuite sur les 2 descripteurs agiront sur le "fichier" désigné par `old`
- `close(fd)` ferme un descripteur.

Pipes

- un pipe est créé par l'appel à `pipe(fds)`, où le paramètre est un tableau de 2 entiers, qui seront les *file descriptors* respectifs pour lire le contenu du tuyau et y écrire.
- la lecture et l'écriture dans un tampon sont atomiques et bloquantes