UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence 3 Informatique
Info0601 – 2017/2018
***Resp. :* Cyril Rabat**

Nom :
Prénom :
Num. inscription :

Devoir surveillé

Durée 2h

Aucun document, outil de calcul ou de communication autorisé.

Exercice 1 (QCM - *5 pts*)

Vous obtiendrez 0,5 point aux questions si vous cochez exactement toutes les bonnes réponses. Une mauvaise réponse cochée entraîne des points négatifs.

1. La taille occupée sur le disque par un fichier...
   - ☐ ...dépend de la taille des blocs
   - ☐ ...est indépendante de la taille des blocs
   - ☐ ...peut être supérieure à sa taille réelle
   - ☐ ...est différente suivant le système de fichiers
2. L'écriture d'une structure dans un fichier en utilisant les fonctions de bas-niveau...
   - ☐ ...conserve l'alignement des structures
   - ☐ ...ne conserve pas l'alignement des structures
   - ☐ ...est indépendante de l'interprétation des données
   - ☐ ...risque de ne pas être lisible sur un autre ordinateur
3. L'appel à `fork`...
   - ☐ ...produit l'envoi d'un signal au processus père
   - ☐ ...produit l'envoi d'un signal au processus fils
   - ☐ ...duplique les données, mêmes celles allouées dynamiquement
   - ☐ ...duplique les données, sauf celles allouées dynamiquement
4. En utilisant le tableau fourni en annexe (page suivante), quelle est la taille de la structure suivante ?

```
typedef struct {
  long a;
  char b;
  char c[2];
  int d;
  short e;
} structure_t;
```

   ☐ 13o ☐ 16o ☐ 24o ☐ Une autre réponse
5. En considérant la question 4, combien d'octets de bourrage sont ajoutés par le compilateur ?

   ☐ 0 ☐ 1 ☐ 3 ☐ Une autre réponse
6. En considérant la question 4, quelle est la taille minimale que l'on peut obtenir en réarrangeant les champs de la structure (sans modifier les types) ?

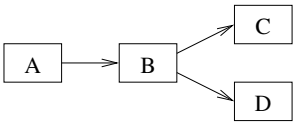
   ☐ 13o ☐ 16o ☐ 24o ☐ Une autre réponse

[redacted]

7. Concernant les tubes anonymes :

   ☐ L'écriture n'est pas forcément atomique
   ☐ Par défaut, la lecture n'est pas bloquante
   ☐ Par défaut, l'écriture n'est pas bloquante
   ☐ Les descripteurs de fichier sont transmis lors de la duplication du processus avec `fork`

8. Combien de sémaphores de *Dijkstra* sont nécessaires au minimum pour résoudre les contraintes du diagramme de précédence suivant (A, B, C et D étant des sections de code de processus quelconques) ?

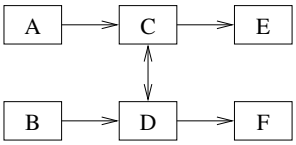


   ☐ 1 ☐ 2 ☐ 3 ☐ 4

9. Soit le diagramme de précédence de la question 8. Combien d'opérations (hors `init`) sont nécessaires au minimum ?

   ☐ 4 ☐ 6 ☐ 8 ☐ 10

10. Combien de sémaphores de *Dijkstra* sont nécessaires au minimum pour résoudre les contraintes du diagramme de précédence suivant (A, B, C, D, E et F étant des sections de code de processus quelconques) ?



   ☐ 1 ☐ 3 ☐ 5 ☐ 7

Annexe : tailles et alignements

Nous considérons les tailles et les alignements suivants (ne correspondant à rien de connu, excepté le devoir surveillé d'Info0601 de M. Rabat) :

Type	**Taille**	**Aligne.**	**Type**	**Taille**	**Aligne.**	**Type**	**Taille**	**Aligne.**
`char`	1o	1o	`short`	2o	2o	`int`	4o	4o
`long`	4o	8o	`float`	4o	4o	`double`	8o	8o
`long long`	8o	8o	`long double`	16o	12o	`pointer`	4o	8o

*Sauf indications contraires, la gestion d'erreur **ne doit pas** être réalisée et les inclusions ne doivent pas être spécifiées. Toute réponse doit être justifiée.*

Exercice 2 (Le voyageur de commerce - *15 pts*)
Nous souhaitons réaliser une application distribuée permettant de résoudre le problème du voyageur de commerce. Pour rappel, ce problème consiste à trouver le trajet le plus court entre un ensemble de villes qui passe une et une seule fois par chaque ville. Les distances et les villes sont fixées et connues de toutes les applications.

Nous développerons une solution basée sur le modèle producteur/consommateur. Le producteur crée les tâches (qui correspondent à un début de parcours) et les consommateurs les résolvent (le résultat est le parcours le plus court commençant par le début de parcours spécifié). Le producteur collecte les résultats envoyés par les consommateurs et conserve le meilleur résultat.

Le producteur crée deux fils : le premier, nommé *gestionnaire de connexions*, sera affecté à la gestion des connexions/déconnexions des consommateurs et le second, nommé *gestionnaire de tâches* sera affecté à la génération des tâches et à la récupération des résultats. Les deux fils enverront leurs informations au producteur en utilisant un unique tube anonyme. Le producteur affichera les résultats au fur-et-à-mesure, ainsi que les connexions/déconnexions.

Pour se connecter, les consommateurs échangeront des signaux avec le gestionnaire de connexions. Le nombre maximum de consommateurs est fixé par la constante `MAX_CONSOMMATEURS` : lors de sa demande de connexion, le consommateur doit être averti si le nombre maximum de connexions est atteint. Lorsqu'un consommateur s'arrête (quand il reçoit un signal `SIGINT`), il en informe le gestionnaire de connexions. De même, lorsque le producteur reçoit un signal `SIGINT`, il doit arrêter ses deux fils et les consommateurs éventuels.

Lorsqu'un consommateur est connecté, il lit une tâche dans un tube nommé. Il envoie le résultat dans un second tube nommé. Les tâches et les résultats seront une suite d'entiers (chacun correspondant à une ville) de longueur variable.

Généralités (4 points)

1°) Proposez une modélisation de votre application sous forme de schéma en précisant chaque entité et les éléments systèmes mis en place. **(1 point)**

2°) Donnez l'ensemble des échanges entre les différentes entités, en précisant le type d'échange et les données échangées. **(1,5 points)**

3°) Sachant que le nom des tubes est fixé dans les constantes `TUBE_TACHES` et `TUBE_RESULTATS` et que les fonctions correspondant aux deux fils du producteur sont `connexion` et `tache`, donnez le code du producteur permettant de créer les deux fils avec le tube anonyme. **(1,5 points)**

Les signaux (6 points)

4°) Nous supposons que le consommateur envoie un signal `SIGRTMIN` au gestionnaire de connexions avec une valeur 1 (on suppose le PID du gestionnaire de connexions connu). Il reçoit à son tour un signal avec un code d'état lui indiquant s'il peut ou non se connecter. Donnez l'extrait de code du consommateur permettant de mettre en place la phase de connexion. **(2 points)**

5°) Comment le gestionnaire de connexions peut-il répondre au consommateur, sachant que le PID du consommateur n'est pas connu ? **(1 point)**

6°) Quand l'utilisateur presse CRTL + C, le consommateur s'arrête. Il envoie alors un signal `SIGRTMIN` au gestionnaire de connexions avec une valeur 2. Donnez les extraits de code du consommateur permettant de mettre en place ce mécanisme. **(2 points)**

7°) Que se passe-t-il du point de vue de l'exécution d'un programme lorsqu'un signal est reçu ? Expliquez ce que l'on doit mettre en place au niveau de la gestion d'erreur. **(1 point)**

Les tubes (5 points)

8°) Expliquez le fonctionnement général du gestionnaire de tâches (sans donner le code mais en précisant les appels systèmes nécessaires). **(1 point)**

9°) Quels sont les problèmes de synchronisation liés aux tubes ? Proposez une solution pour chacun. **(1 point)**

10°) En considérant que la taille des tâches est suffisamment petite, en quoi la taille maximale d'un tube peut-elle poser problème pour l'envoi de tâches et la réception des résultats ? Y'a-t-il une possibilité d'obtenir un cas d'inter-blocage ? Comment l'éviter ? **(1 point)**

11°) Que se passe-t-il lors de la connexion ou la déconnexion de consommateurs du côté du gestionnaire de tâches ? Expliquez les problèmes liés à l'utilisation des tubes nommés et proposez une solution sans ajouter de moyens de communications supplémentaires entre le gestionnaire de tâches et les consommateurs. **(2 points)**

Annexes : quelques en-têtes de fonctions C

```
void           _exit(int code);
unsigned int   alarm(unsigned int nb_secondes);
int            atexit(void (*procedure)(void));
void           clearerr(FILE *flux);
int            close(int fd);
int            closedir(DIR *repertoire);
int            creat(const char *chemin, mode_t mode);
int            dup(int ancien_fd);
int            dup2(int ancien_fd, nouveau_fd);
int            execve(const char *fichier, char *const argv[], char *const envp[]);
void           exit(int statut);
int            fcntl(int fd, int commande, ...);
int            fclose(FILE *flux);
FILE          *fdopen(int fd, const char *mode);
int            feof(FILE *flux);
int            ferror(FILE *flux);
int            fflush(FILE *flux);
int            fileno(FILE *flux);
FILE          *fopen(const char *chemin, const char *mode);
pid_t          fork();
size_t         fread(void *tampon, size_t taille, size_t nombre_elements, FILE *flux);
FILE          *freopen(const char *chemin, const char *mode, FILE *flux);
int            fseek(FILE *flux, long offset, int point_depart);
int            fstat(int fd, struct stat *tampon);
int            ftell(FILE *flux);
size_t         fwrite(const void *tampon, size_t taille, size_t nombre_elements, FILE *flux);
pid_t          getpid();
pid_t          getppid();
int            kill(pid_t pid, int numero_signal);
off_t          lseek(int fd, off_t offset, int point_depart);
int            lstat(const char *chemin, struct stat *tampon);
void          *memcpy(void *destination, const void *source, size_t taille);
void          *memset(void *tampon, int caractere, size_t nombre_elements);
int            mkfifo(const char *nomFichier, mode_t mode);
int            open(const char *chemin, int options);
int            open(const char *chemin, int options, mode_t mode);
DIR           *opendir(const char *chemin);
int            pause(void);
int            pipe(int tube[2]);
ssize_t        read(int fd, void *tampon, size_t taille);
struct dirent *readdir(DIR *repertoire);
void           rewind(FILE *flux);
void           rewinddir(DIR *repertoire);
void         (*signal(int numero_signal, void (*handler)(int)))(int)
int            sigaction(int numero_signal, const struct sigaction *action,
                         struct sigaction *ancienne_action);
int            sigaddset(sigset_t *ensemble, int numero_signal);
int            sigdelset(sigset_t *ensemble, int numero_signal);
int            sigemptyset(sigset_t *ensemble);
int            sigfillset(sigset_t *ensemble);
int            sigismember(sigset_t *ensemble, int numero_signal);
int            sigpending(sigset_t *ensemble);
int            sigprocmask(int comment, const sigset_t *ensemble, sigset_t *ancien);
int            sigqueue(pid_t pid, int numero_signal, const union sigval valeur);
int            sigsuspend(const sigset_t *ensemble);
int            sigwaitinfo(const sigset_t *ensemble, siginfo_t *info);
int            sigtimedwait(const sigset_t *ensemble, siginfo_t *info,
                            const struct timespec *timeout);
unsigned int   sleep(unsigned int nombre_secondes);
int            stat(const char *chemin, struct stat *tampon);
```

```
time_t          time(time_t *temps);
int             truncate(const char *chemin, off_t offset);
int             ftruncate(int fd, off_t offset);
int             truncate64(const char *chemin, off64_t offset);
int             ftruncate64(int fd, off64_t offset);
int             unlink(const char *nomFichier);
pid_t           wait(int *statut);
pid_t           waitpid(pid_t pid, int *statut, int options);
ssize_t         write(int fd, const void *tampon, size_t taille);
```

Annexes : constantes associées à des paramètres ou des champs de structures

options (open, fcntl) : O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_TRUNC, O_APPEND
mode (creat, open) : S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP, S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH
commande (fcntl) : F_GETFL, F_SETFL
mode (fopen, fdopen, freopen) : "r", "r+", "w", "w+", "a", "a+"
point_depart (lseek, fseek) : SEEK_SET, SEEK_CUR, SEEK_END
mode (mkfifo) : O_RDONLY, O_WRONLY, O_CREAT, O_EXCL, S_IRWXU, S_IRUSR, S_IWUSR, S_IXUSR, S_IRWXG, S_IRGRP, S_IWGRP, S_IXGRP, S_IRWXO, S_IROTH, S_IWOTH, S_IXOTH
handler (signal, champ sa_handler de struct sigaction) : SIG_IGN, SIG_DFL
sa_flags (champ sa_flags de struct sigaction) : SA_ONESHOT, SA_NOMASK, SA_SIGINFO
comment (sigprocmask) : SIG_BLOCK, SIG_UNBLOCK, SIG_SET
options (waitpid) : WNOHANG, WUNTRACED

Annexes : macros

Sur **statut** de wait, waitpid : WIFEXITED, WEXITSTATUS, WIFSIGNALED, WTERMSIG, WIFSTOPPED, WSTOPSIG
Sur le champ **st_mode** de struct stat : S_ISREG, S_ISDIR, S_ISFIFO

Annexes : quelques structures C

```
struct stat {
  dev_t      st_dev;
  ino_t      st_ino;
  mode_t     st_mode;
  nlink_t    st_nlink;
  uid_t      st_uid;
  gid_t      st_gid;
  dev_t      st_rdev;
  off_t      st_size;
  blksize_t  st_blksize;
  blkcnt_t   st_blocks;
  time_t     st_atime;
  time_t     st_mtime;
  time_t     st_ctime;
};
```

```
struct siginfo_t {
  int      si_signo;
  int      si_code;
  sigval_t si_value;
  pid_t    si_pid;
  ...
};
```

```
struct dirent {
  char d_name[256];
};
```

```
struct sigaction {
  void     (*sa_handler) (int);
  void     (*sa_sigaction) (int, siginfo_t*, void*);
  sigset_t sa_mask;
  int      sa_flags;
};
```

```
union sigval_t {
  int   sival_int;
  void *sival_ptr;
};
```

```
struct timespec {
  long tv_sec;
  long tv_nsec;
};
```