Université de Reims
Champagne-Ardenne
U.F.R. de Sciences
Exactes et Naturelles

Licence INFO
INFO0403
2020/2021

Devoir Surveillé

2 heures

Exercice 1 (Questions de Cours (2 pts))

- Expliciter les spécificités de la 3eme génération d'ordinateurs 1965-1980 (1pt).
- Définir et expliquer le mécanisme d'ordonnancement (1pt).

Exercice 2 (Concurence (4 pts))

Soit 4 processus (P1, P2, P3 et P4). P1 et P3 sont découpés en deux parties (P1A, P1B, P3A et P3B) et P2 et P4 sont découpés en trois parties (P2A, P2B, P2C, P4A, P4B et P4C). Nous avons les règles de précédance suivantes :

- P4A précède P3A,
- P3A précède P2A,
- P3A Précède P1B,
- P3A précède P4C,
- P2B précède P4B
- et P2C précède P4C.

De plus

- P2C doit être réalisé en exclusion mutuelle avec P1B et P3B
- et P3B doit être réalisé en exclusion mutuelle avec P2B et P2C.

1) Donnez le graphe de précédence.

2) Ecrire le pseudo code des 4 processus en utilisant des sémaphores pour résoudre les problèmes de concurence et de synchronisation. Vous n'omettrez pas de préciser l'initialisation des sémaphores.

Exercice 3 (Ordonnancement (5 pts))

Nous considérons 6 jobs qui ont respectivement un temps d'exécution de 8, 13, 12, 5 , 8 et 14 secondes et une priorité de 3, 3, 5, 1,5 et 4 (la priorité 5 étant la plus élevée). Ces processus sont soumis respectivement aux temps (en secondes) $t = 1, t = 3, t = 2, t = 5, t = 3$ et $t = 8$. Pour chacun des algorithmes d'ordonnancement suivants, donnez le temps moyen d'attente (temps d'attente : temps nécessaire pour que le processus soit élu pour la première fois par l'ordonnanceur) et la durée moyenne d'exécution (temps d'exécution : temps écoulé entre la soumission et la fin de l'exécution).

FIFO

Job le plus long en premier

Job le plus court en premier

Au plus court reste d'abord

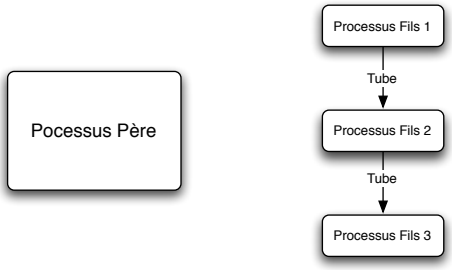
Round Robin (Quantum = 2s)

Ordonnancement par priorité (Quantum = 2s)

Exercice 4 (Tubes (4 pts))

On souhaite mettre en place l'automatisation de traitement de fichiers. Ces traitements seront réalisés par trois processus distincts qui exécuteront respectivement les tâches *action_1*(), *action_2*() et *action_3*(). Ces processus seront initialisés par un processus père. Les processus de traitement communiqueront entre eux en utilisant des tubes. Voici l'architecture de communication des processus.

Le processus réalisant la tâche *action_1*() lit le fichier de données *data.txt*, effectue l'action *action_1*() et transfert le résultat au deuxième processus de traitement via un tube . Celui-ci effectuera alors son traitement (*action_2*()) et transférera le résultat au troisième processus via un deuxième tube. Ce dernier effectuera alors son traitement (*action_3*()) avant de sauvegarder le résultat dans un fichier *toto.txt*. Vous considèrerez à chaque transfert via un tube envoyer un $buffer$ de 255 caractères.



1) Ecrire le code de la procédure $fils(int\, i)$ (i étant le numéro de traitement).

2) Ecrire le code du programme principal.

Exercice 5 (Fichiers et Répertoires (5 pts))
1) Ecrire une fonction qui prend en paramètre un nom de fichier fic et un caractère s et qui retourne 1 si fic contient s et 0 sinon.

2) Ecrire un programme qui affiche le nom des fichiers du répertoire courant qui contiennent un caractère saisi par l'utilisateur.

3) Ecrire un programme qui affiche le nom des fichiers du répertoire courant et des répertoires qu'il contient, qui contiennent un caractère saisi par l'utilisateur.

Annexes

- `void (*signal(int sig, void (*func)(int)))(int);`
- `int beep(void);`
- `int close(int fd);`
- `int closedir(DIR *dir);`
- `int chdir(const char *file);`
- `int execlp(const char *file, const char *arg, ... /*, (char *)0 */);`
- `int flash(void);`
- `int fork();`
- `int kill(pid_t pid, int signo);`
- `off_t lseek(int fd, off_t offset, int whence);`
- `int open(const char *pathname, int flags, mode_t mode);`
- `DIR *opendir(const char *name);`
- `int pipe(int filedes[2]);`
- `int printf(const char * restrict format, ...);`
- `int rand(void);`
- `ssize_t read(int fd, void *buf, size_t count);`
- `struct dirent *readdir(DIR *dir);`
- `int scanf(const char * restrict format, ...);`
- `void (*signal(int sig, void (*func)(int)))(int);`
- `void srand(unsigned seed);`
- `int stat(const char *path, struct stat *buf);`
- `int system(const char *string);`
- `int raise(int signo);`
- `ssize_t write(int fd, const void *buf, size_t count);`
- `int wait(int * status);`
- `int semget (key_t key, int nsems, int semflg);`
- `int semctl (int semid, int semno, int cmd, ...);`

- `int semop(int semid, struct sembuf *sops, unsigned nsops);`
- ```
  struct sembuf{
  unsigned short sem_num;
  short sem_op;
  short sem_flg;
  }
  ```
- ```
  union semun {
  int val;
  struct semid_ds *buf;
  unsigned short *array;
  struct seminfo *_buf;
  };
  ```
- ```
  struct stat {
  dev_t st_dev; /* ID du périphérique contenant le fichier */
  ino_t st_ino; /* Numéro in½ud */
  mode_t st_mode; /* Protection */
  nlink_t st_nlink; /* Nb liens matériels */
  uid_t st_uid; /* UID propriétaire */
  gid_t st_gid; /* GID propriétaire */
  dev_t st_rdev; /* ID périphérique (si fichier spécial) */
  off_t st_size; /* Taille totale en octets */
  blksize_t st_blksize; /* Taille de bloc pour E/S */
  blkcnt_t st_blocks; /* Nombre de blocs alloués */
  time_t st_atime; /* Heure dernier accès */
  time_t st_mtime; /* Heure dernière modification */
  time_t st_ctime; /* Heure dernier changement */
  };
  ```
- ```
  struct dirent {
  ino_t d_ino; /* numéro d'inœud */
  off_t d_off; /* décalage jusqu'à la dirent suivante */
  unsigned short d_reclen; /* longueur de cet enregistrement */
  unsigned char d_type; /* type du fichier */
  char d_name[256]; /* nom du fichier */
  };
  ```