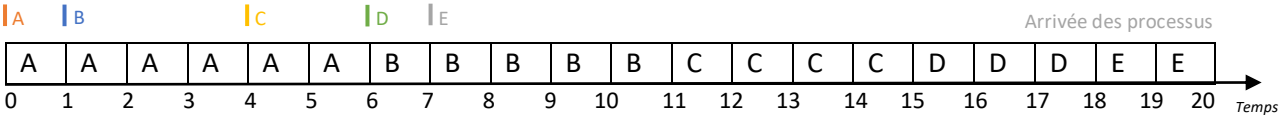
Huang Roger, 11807555

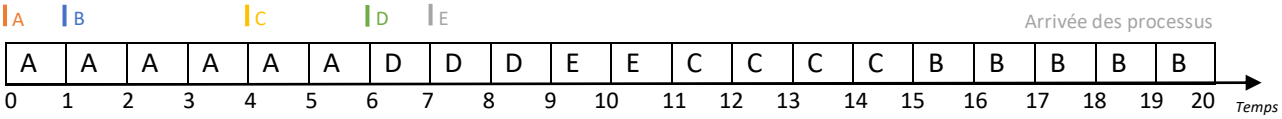
Devoir Systèmes d'exploitation, Réseaux

Exercice 1 :

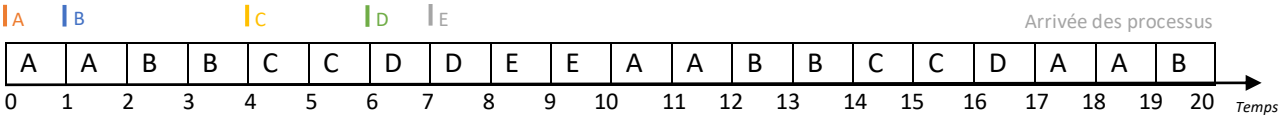
Fifo | $t.m.t = \frac{(6-0)+(11-1)+(15-4)+(18-6)+(20-7)}{5} = 10{,}4\ unités\ de\ temps$



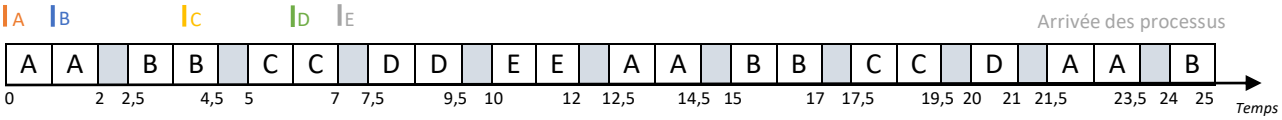
SRTF | $t.m.t = \frac{(6-0)+(20-1)+(15-4)+(9-6)+(11-7)}{5} = 8{,}6\ unités\ de\ temps$



Round Robin | $t.m.t = \frac{(19-0)+(20-1)+(16-4)+(17-6)+(10-7)}{5} = 12{,}8\ unités\ de\ temps$



Round Robin avec commutation | $t.m.t = \frac{(23{,}5-0)+(25-1)+(19{,}5-4)+(21-6)+(12-7)}{5} = 16{,}6\ unités\ de\ temps$



Exercice 2 :

1- Le système de tâches avec un ordre t1< t4< t2< t5 est-il déterminé ?
Domaines de lecture et d'écriture des tâches :

	t1	t2	t3	t4	t5
L_i	{C8,C6}	{C8,C7}	{C1,C7}	{C1,C2}	{C4,C3}
E_i	{C1}	{C2}	{C3}	{C4}	{C5}

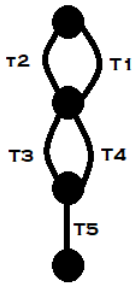
Or t5 et t3 sont des tâches interferentes avec cet ordre, car il n'existe pas de relation d'ordre entre t3 et t5 et de plus, E3 ∩L5 ≠ ∅ (=C3). Il existe donc deux tâches interférentes, ainsi le système ne peut pas être déterminé.

2- Le système de tâches avec l'ordre t1 < t2 < t3 < t4 < t5 est-il déterminé ?
C'est un système de tâches avec un ordre total, alors le système est nécessairement déterminé (toutes les tâches sont non interférentes car on trouvera toujours une relation d'ordre entre deux tâches).

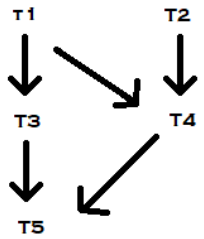
3- Donner le système de parallélisme maximal calculant l'expression.

D'après notre système de tâche, on déduit au minimum l'ordre suivant quant à l'exécution des tâches : t1<t3,t4<t5, t2<t4.

On obtient par exemple le graphe de flot suivant et le graphe de précédence :



Le système de tâche S = (T,<) avec T = (t1,t2,t3,t4,t5), avec l'ordre t1<t3<t5, t2<t4<t5, t1<t4 est de parallélisme maximal.



C'est d'abord un système déterminé, car toutes les tâches sont deux à deux non interférentes. Les couples qu'on peut former entre t1, t3, t5 sont non interférentes d'après la relation d'ordre, idem pour (t2,t4), (t1,t4), et (t4,t5). On a aussi t2<t5 par transitivité, donc (t2,t5) non interférentes.

Il reste 3 couples qui n'ont pas de relation d'ordre entre eux, il faut s'assurer que les domaines d'écriture et de lecture ne se superposent pas:

(t1,t2) : $L1 \cap E2 = L2 \cap E1 = E1 \cap E2 = \emptyset$
(t2,t3) : $L2 \cap E3 = L3 \cap E2 = E2 \cap E3 = \emptyset$
(t4,t3) : $L4 \cap E3 = L3 \cap E4 = E3 \cap E4 = \emptyset$

Le système est déterminé, est-il de parallélisme maximal ?
Si on supprime t2<t4, alors t2 et t4 sont interférentes car L4∩E2 = C2
Si on supprime t3<t5, alors t3 et t5 sont interférentes car L5∩E3 = C3
Si on supprime t4<t5, alors t4 et t5 sont interférentes car L5∩E4 = C4
Si on supprime t1<t3, alors t1 et t3 sont interférentes car L3∩E1 = C1
Si on supprime t1<t4, alors t1 et t4 sont interférentes car L4∩E1 = C1
La réponse est oui car tout S' = (T, < ') obtenu en supprimant du graphe de < un couple qui est dans le graphe de précédence de < est non déterminé, il y a toujours une interférence.

4 – Programme utilisant les primitives de Dijkstra

```
begin
        parbegin t1|t2 parend; ;
        parbegin t3|t4 parend; ;
        t5
end;
```

Exercice 3 :

1- Ecrire la fonction estExecutable.

Extrait du fichier nommé « exe3-3.c »

```c
void estExecutable(char *nom_fichier) {
    printf("ID du thread: %d\n", gettid());

    int tab[4]={127,69,76,70}; //caracteristiques d'un fichier executable

    int fd;
    if( (fd = open(nom_fichier, O_RDONLY) ) == -1) {
        perror("erreur d'ouverture du fichier");
        return;
    }

    char buffer[10];
    read(fd, buffer, 4); //on lit les 4 premiers octets du fichier

    for(int i=0; i < 4; i++){
        if(buffer[i] != tab[i]){
            close(fd);
        return;
        }
    }

    printf("%s\n", nom_fichier);
    close(fd);
    return;
}
```

2- Ecrire la fonction main (et des fonctions auxiliaires si nécessaire).

Extrait du fichier nommé « exe3-3.c »

```c
void *executer(void *s){
    char* nom_fichier = (char*) s;
    estExecutable(nom_fichier);
    return 0;
}


int main(int argc, char *argv[]) {

  while(1){
    pthread_t unthread;

    char* string = malloc(sizeof(char*));
    printf("Entrer un nom de fichier :\n");
    scanf("%s",string);

    pthread_create(&unthread, NULL, executer, string);
    pthread_join(unthread, NULL);
  }
  return 0;
}
```

3- Ajouter ce qui est nécessaire pour que le programme puisse, en cas de réception du signal SIGUSR1, afficher « Fin du programme » et terminer.

Extrait du fichier nommé « exe3-3.c »

```c
#include <signal.h>

/*Signal*/
void quefaire(int sig){
    if(sig == SIGUSR1){
    printf("Fin du programme\n");
    exit(1);
    }
}
```

```c
int main(int argc, char *argv[]) {
  signal(SIGUSR1, quefaire);
  printf(" [kill -10 %d] pour arrêter le processus \n", getpid());

  while(1){
    pthread_t unthread;

    char* string = malloc(sizeof(char*));
    printf("Entrer un nom de fichier :\n");
    scanf("%s",string);

    pthread_create(&unthread, NULL, executer, string);
    pthread_join(unthread, NULL);
  }
  return 0;
}
```

4 – (a) Sans considérer les problèmes d'exclusion mutuelle, écrire le programme.

Réponse dans le fichier « exe3-4-a.c »

(b) Ajouter les sémaphores nécessaires au bon fonctionnement du tableau partagé.

Réponse dans le fichier « exe3-4-b.c »

Remarque : pour compiler les fichiers utiliser une commande du type :
gcc -Wall exe3-3.c -o exe3-3.exe -lpthread
Pour tester, on peut taper : ./exe3-3.exe