TD8 : Ordonnancement de processus

On veut modéliser le partage par plusieurs processus du temps de calcul sur un processeur, dans une version très simplifiée de l'algorithme de *round robin*. Le principe de ce partage est le suivant : le processeur possède une liste de processus auxquels il accorde, à tour de rôle et dans l'ordre, un temps de calcul fixé ; si le processus a terminé son travail dans le temps imparti, il disparaît de la liste des processus existants.

1 Processus

Un processus est (grossièrement) un programme en train de s'exécuter. Pour simplifier, on suppose qu'il correspond au lancement d'un exécutable (ex : `ls`, `emacs`, `firefox`, etc.). Il est identifié par un entier (son *identifiant*).
On introduit le type structuré

```
struct processus {
  char *exec; // nom de l'exécutable
  int id;     // identifiant du processus
};
```

On suppose qu'on dispose un constructeur

```
struct processus *lance_processus(char *exec);
```

qui prend en argument un nom d'exécutable et renvoie l'adresse d'un processus où les deux champs `exec` et `id` ont été initialisés.
On suppose qu'on dispose d'une fonction

```
bool est_fini(struct processus *p);
```

qui permet de savoir si l'exécution du processus est terminée ou non, et libère la mémoire associée à `p` le cas échéant.

2 Processeur et ordonnanceur

Dans cette partie, on modélise la file d'attente des processus en cours par une liste chaînée circulaire (voir Fig. 1) :

```
struct process {
  struct processus *actif;
  struct process *suivant;
  struct process *precedent;
};
```

Chaque processus est référencé par à un maillon de la liste. Ce maillon contient en plus une référence vers le maillon précédent et une référence vers le maillon suivant.
L'entité d'un système qui s'occupe de l'ordre dans lequel les processus sont exécutés s'appelle un *ordonnanceur*.

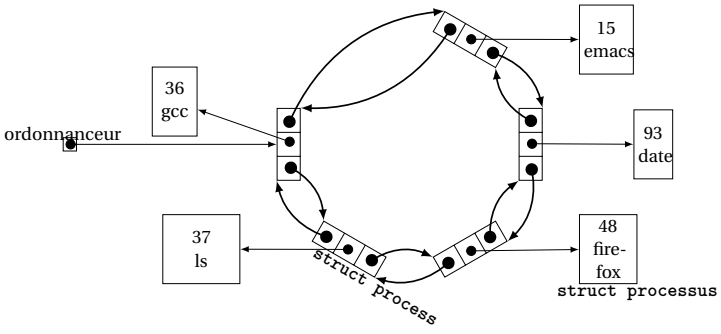


FIGURE 1 –

Dans une liste circulaire doublement chaînée, la première cellule possède une référence vers la dernière cellule (c'est sa cellule précédente) et la dernière cellule possède une référence vers la première (c'est sa cellule suivante).
On veut implémenter les fonctions suivantes :

```
/** affiche la liste des processus (id+nom pour chaque processus) */
void ps(struct process *ordonnanceur);
/** ajoute un nouveau processus dans la liste
 * ordonnanceur : référence la liste de processus
 * p : processus à ajouter */
struct process *add_process(struct process *ordonnanceur, struct processus *p);
/** stoppe le processus d'identifiant id (le retire de la liste des processus)
 * ordonnanceur : référence la liste de processus
 * id : identifiant du processus a stopper
 * sortie : maillon retiré de la liste et mémoire libérée */
struct process *kill(struct process *ordonnanceur, int id);
/** stoppe les processus dont le nom est donne en parametre
 * ordonnanceur : référence la liste de processus
 * nom : nom de l'exécutable associé aux processus à stopper
 * sortie : maillon retiré de la liste et mémoire libérée */
struct process *killall(struct process *ordonnanceur, char *nom);
/** si le processus courant est terminé, il est supprimé de la liste (avec mémoire
 * libérée)
 * ordonnanceur : référence la liste de processus */
struct process *cpu(struct process *ordonnanceur);
```

Question 1 : Écrire la fonction `ps`.
Dans la situation de la Fig. 1, un appel à `ps` provoque l'affichage :

```
PID    CMD
36     gcc
15     emacs
93     date
48     firefox
37     ls
```

Question 2 : Écrire la fonction `add_process`.
Dans la situation de la Fig. 1, un appel à `add_process(ordonnanceur, lance_processus("jupyter"))` mène à :

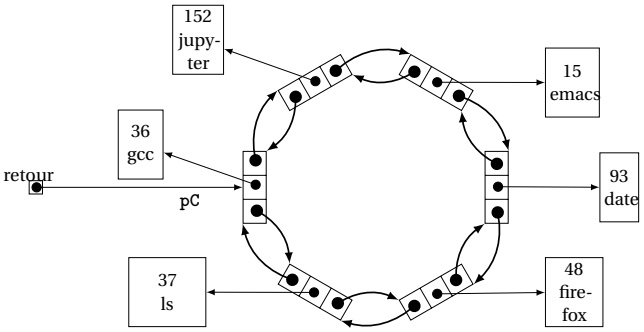
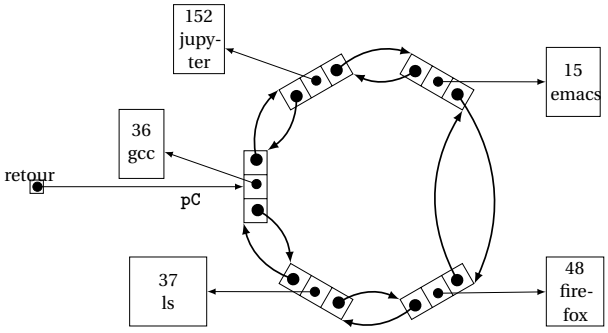
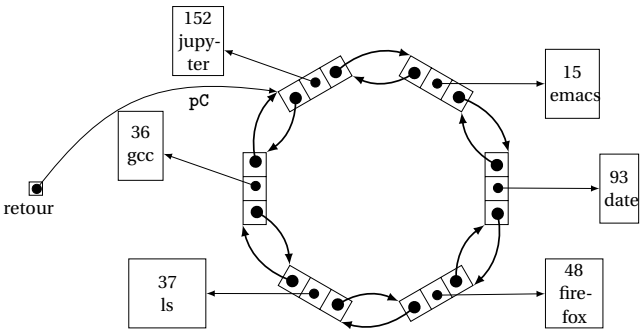


FIGURE 2 –

Question 3 : Écrire les fonctions `kill` et `killAll`.
Dans la situation de la Fig. 2, un appel à `kill(ordonnanceur, 93)` mène à :



Question 4 : Écrire la fonction `cpu`.
Dans la situation de la Fig. 2, un appel à `cpu(ordonnanceur)` mène à :



3 Algorithme Round Robin

Question 5 : Écrire une fonction `round_robin` qui prend en paramètre un pointeur sur `struct process` et qui fait tourner le processeur tant qu'il y a des processus dans la liste.