TD 1 : Mise en œuvre de la gestion des processus

2SN ASR

2019-2020

Nous nous proposons d'écrire un noyau d'exécution et de synchronisation pour un système d'exploitation de gestion d'une machine monoprocesseur. Ce noyau doit assurer les diverses opérations sur les processus, l'attribution de l'unité centrale (UC) et la synchronisation des processus.
Un processus est représenté par un descriptif de processus, initialisé à la création du processus. Ce description contient toutes les informations décrivant le processus.

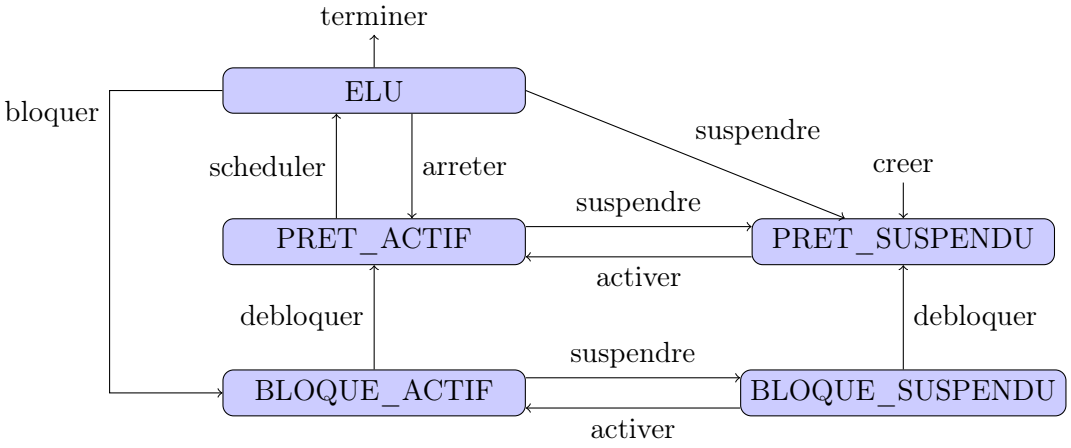

Figure 1 – États des processus

Comme le montre la figure 1, un processus passe au cours de son existence par une succession d'états. Ces états sont définis par le type PROCESS_STATE :

typedef enum {ELU, PRET_ACTIF, PRET_SUSPENDU,
BLOQUE_ACTIF, BLOQUE_SUSPENDU} PROCESS_STATE;

- Un processus dans l'état PRET_ACTIF ou PRET_SUSPENDU est dans l'attente de l'unité centrale.
- Un processus dans l'état BLOQUE_ACTIF ou BLOQUE_SUSPENDU est en attente d'une ressource.
- La suspension d'un processus est un blocage logique : elle est demandée explicitement par un processus qui s'exécute sur l'unité centrale à l'encontre d'un processus actif.
- Un processus suspendu attend d'être activé.
- Les processus PRET_ACTIF sont mémorisés dans une file d'attente des processus prêts.
- Un processus bloqué est consigné dans une file d'attente associée à la ressource demandée.
- Une fonction SCHEDULER assure la gestion de l'unité centrale : elle doit s'assurer que la priorité du processus en cours d'exécution sur l'unité centrale, dans l'état ELU, est supérieure ou égale à celle de tout processus se trouvant dans l'état PRET_ACTIF. Elle est appelée à chaque fois qu'un processus quitte l'état ELU ou passe dans l'état PRET_ACTIF. Prototype : **void** scheduler();

Structure de données

Le descriptif de processus :

```
struct process_t {
        void *stack;
        PROCESS_STATE state;
        PROCESS_ID pid;
        RESOURCES_LIST resources;
};
```

Le système gère les processus sous la forme d'un tableau :

```
struct process_t process_table[MAX_PROCESS];
```

MAX_PROCESS correspond au nombre maximal de processus du système. Le processus de pid 100 sera stocké dans la table à la ligne 100, *i.e.* dans la case process_table[100].
On suppose écrites les fonctions suivantes :

- Gestion de la file d'attente des processus prêts :

```
void addProcess(PROCESS_ID pid); /* Add the process pid
        in the queue */
void removeProcess(PROCESS_ID pid); /* Remove the process
        from the queue */
```

- Gestion de la file d'attente des ressources :

```
void addResource(RESOURCE_ID rid, PROCESSUS_ID pid) ;
/* Add the process pid to the queue of resource rid */
void removeResource(RESOURCE_ID rid, PROCESSUS_ID pid) ;
/* remove the process pid from the queue of resource rid */
```

- Gestion de l'unité centrale :

```
void scheduler();
void *stop_process_uc(); /* stops process execution
        and returns its stack */
PROCESS_ID getpid(); /* pid of the running process */
```

Création d'un processus

```
PROCESSUS_ID creer()
{
        PROCESSUS_ID Pid ;

        if ((Pid=Allouer_Pid())==-1)
                {
                        perror("␣Cannot␣create␣new␣process␣\n␣") ;
                        exit(1) ;
                }

        process_table[Pid].pid=Pid ;
        process_table[Pid].stack= init_stack() ;
        process_table[Pid].state=PRET_SUSPENDU ;
```

```
        process_table[Pid].resources= init_resources();

        return(Pid) ;
}
```

Questions

1. Analysez et commentez le code de la fonction créer.
2. En observant le diagramme d'état, écrire le pseudo-code C des fonctions suivantes :
   (a) Activation de processus

   ```
   void activer(PROCESSUS_ID pid) ;
   ```

   Cette fonction permet de passer d'un état SUSPENDU à un état ACTIF.

   (b) Suspension d'un processus

   ```
   void suspendre(PROCESSUS_ID pid) ;
   ```

   SUSPENDRE place le processus pid dans l'état SUSPENDU.

   (c) Arrêt d'un processus

   ```
   void arreter() ;
   ```

   ARRETER place le processus qui s'exécute sur l'unité centrale dans l'état PRET_ACTIF.

   (d) Blocage d'un processus

   ```
   void bloquer(RESSOURCE_ID rid) ;
   ```

   BLOQUER place le processus qui s'exécute sur l'unité centrale dans l'état BLOQUE_ACTIF. Cette fonction est appelée lorsqu'un processus n'a pas pu obtenir la ressource rid. Il sera mis en attente de cette ressource. L'unité centrale devient alors libre.
3. Le processeur CRAPS peut recevoir une interruption TIMER : l'exécution en cours sera stoppée et le PC est positionné à la ligne 1 qui correspond au début du traitement associé à l'interruption TIMER. Une fonction associée à ce traitement peut alors être appelée :

   ```
                ba __main; // branch to the main function
   it_timer:    call ...
   ```

   Quelle fonction sera appelée lors de la réception de l'interruption TIMER ?
4. Le processeur CRAPS dispose aussi d'une instruction `int` qui génère une interruption logicielle. Le processeur déroute alors l'exécution en cours vers la ligne 2 de la mémoire :

   ```
                ba __main; // branch to the main function
   it_timer:    call ...
   it_soft:     call ...
   ```

   Nous nous proposons de construire l'appel-système `PROCESS_ID fork()` qui permet de créer un processus par l'utilisateur. Le fonction `syscall` exécute les opérations du noyau associées aux appels systèmes. Elle utilise le registre `%r1` pour lire le numéro de l'appel système à exécuter et retourne le résultat dans le registre `%r2`. Les numéros des appels systèmes disponibles sont :

   ```
   #define NR_exit 0
   #define NR_fork 1
   #define NR_read 2
   ...
   ```

Une table d'appels systèmes permet de savoir quelle fonction doit être appelée par `syscall` :

```
syscall_table: .word __terminer, ...
```

Remarque. Lorsque les fonctions que vous avez écrites précédemment (`creer, arreter, ...`) sont compilées, l'étiquette __nom_fonction leur est associée. Par exemple, la fonction **void** terminer() compilée peut être appelée via l'étiquette __terminer.

(a) Écrire le code de la fonction PROCESS_ID fork().

(b) Compléter la table des appels systèmes et écrire le code la fonction syscall (en pseudo-assembleur CRAPS).