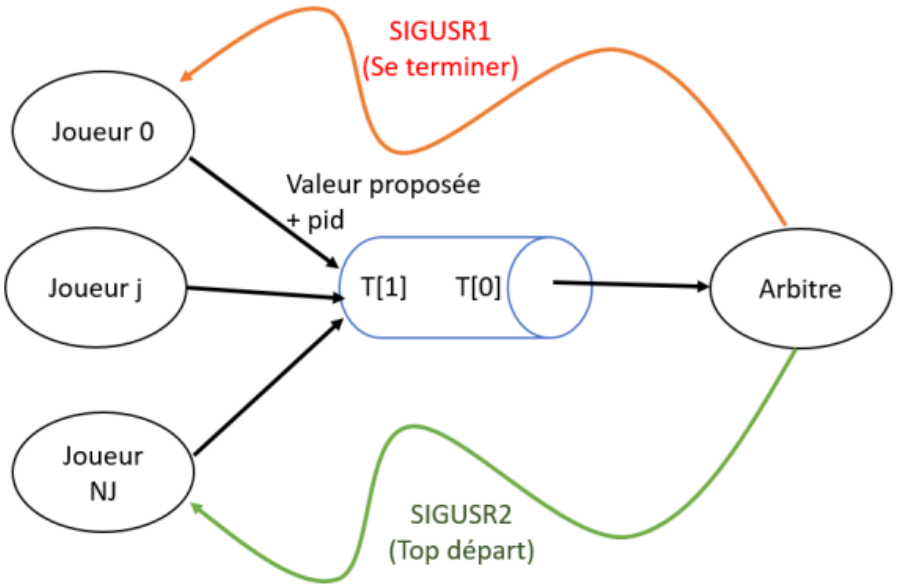
Université Paul Sabatier
31062 Toulouse Cedex 09

Licence 3 Informatique

Programmation Système
Éléments de solution Session 1 décembre 2016

Exercice 1 – Processus UNIX



Main

Créer le tube [pipe]

Protéger les fils contre SIGUSR1 et SIGUSR2 grâce à 2 handlers [signal / sigaction]

Créer NJ joueurs en mémorisant leurs pids [fork]

Créer l'arbitre (qui hérite des pids) [fork ou devenir l'arbitre]

Arbitre

Fermer tube en écriture [close]

Tant qu'il reste des joueurs en lice faire

    Choisir la valeur cible

    Envoyer le top départ à tous les joueurs (SIGUSR2) [kill]

    Lire les réponses des joueurs dans le tube [read]

        en mémorisant les pids des plus mauvaises propositions

    Si on ne veut pas éliminer tous les joueurs **alors**

        Éliminer ceux marqués en leur envoyant SIGUSR1 [kill]

    Mettre à jour le nombre de joueurs restant

FinTantQue

Afficher le pid du joueur gagnant

Fermer le tube en lecture [close]

Joueur (numJoueur)

1er handler pour gérer la réception de SIGUSR1 : positionner seTerminer à vrai

2e handler pour gérer la réception de SIGUSR2 : début du tour

Fermer le tube en lecture

Tant que seTerminer est faux faire

Écrire une proposition dans le tube ainsi que son pid (en une fois ➔ structure) [write]

Attendre la réponse de l'arbitre (fin ou top départ) [pause / sigsuspend]

FinTantQue

Fermer le tube en écriture [close]

Remarque : Dans cet exercice, il n'y a pas de traitement périodiquement. Mais, si, par exemple, les joueurs avaient dû envoyer périodiquement leur réponse, il aurait fallu utiliser un alarm() et gérer le signal SIGALRM du côté du processus qui le reçoit (cf. exercices 4, 5 et 6 des Cours-TD).

Exercice 2 – Synchronisation par sémaphores

Question 1

```
Semaphore SAccesPaniers, SAccesCabines;
Init(SAccesPaniers, NP);        /* NP paniers accessibles */
Init(SAccesCabines, NC);        /* NC cabines accessibles */

Nageur {
    P(SAccesPaniers)            /* prendre un panier */
    P(SAccesCabines)            /* occuper une cabine */
    <se déshabiller>
    V(SAccesCabines)            /* libérer la cabine */
    <nager>
    P(SAccesCabines)            /* occuper une cabine */
    <se rhabiller>
    V(SAccesCabines)            /* libérer la cabine */
    V(SAccesPaniers)            /* libérer le panier */
}
```

Question 2

```
Semaphore SAccesPaniers, , SEntrants, SSortants, mutex;


Init(SAccesPaniers, NP);        /* NP paniers accessibles */
Init(SEntrants, 0);
Init(SSortants, 0);
Init(mutex, 1);                 /* E.M. sur variables partagées */


int nbCabinesOccupees = 0,      /* Combien il reste de cabines libres ? */
        nbEntrants = 0,         /* Combien d'entrants ? */
        nbSortants = 0;         /* Combien de sortants ? */
```

```
Nageur {
    P(SAccesPaniers);              /* Obtenir un panier */
    demanderCabine(&nbEntrants, SEntrants) ;
    <se déshabiller>
    libererCabine() ;
    <nager>
    demanderCabine(&nbSortants, SSortants) ;
    <se rhabiller>
    libererCabine() ;
    V(SAccesPaniers);              /* Rendre le panier */
}


void demanderCabine(int *n, Semaphore acces) {
    P(mutex);                                          /* E.M. sur compteurs */
    if (nbCabinesOccupees < NC) {                      /* Une cabine libre encore, */
        nbCabinesOccupees = nbCabinesOccupees + 1 ;    /* on la prend */
        V(mutex) ;
    }
    else {
        *n = *n + 1;               /* Un entrant ou un sortant de plus /
        V(mutex);
        P(acces);                  /* Se bloquer, selon sa catégorie, en attente d'une cabine */
    }
}

void libererCabine(void) {
    P(mutex) ;                             /* E.M. sur compteurs */
    if (nbEntrants > 0) {                  /* Un entrant en moins */
        nbEntrants = nbEntrants –  1;
   V(SEntrants);                           /* Un accès entrant de plus */
  }
    else
        if (nbSortants > 0){
            nbSortants = nbSortants –  1;  /* Un sortant en moins */
            V(SSortants);                  /* Un accès sortant de plus */
   }
        else
        nbCabinesOccupees = nbCabinesOccupees - 1;    /* Cabine libérée */
    V(mutex) ;
}
```