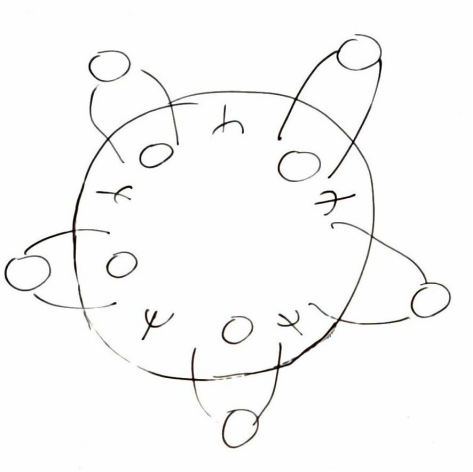
Systèmes d'exploitation - Moniteurs & Sémaphores

Othmane AJDOR

2018-2019

Table des matières

1 Philosophes



1.1 Moniteurs

```
enum EtatPhilo{MANGE, PENSE};
EtatPhilo etatsPhilo[N] = {PENSE, ...};
mtx_t m;
cnd_t prive[N];

Philo(int i){
    while(1){
        pense();
        prendre_fourchette();
        mange();
        poser_fourchette();
    }
}

prendre_fourchettes(int i){
    mtx_lock(&m);
    int v1 = (i+N-1)%N;
    int v1 = (i+N+1)%N;
    while (etatsPhilo[v1] == MANGE || etatsPhilo[v2] == MANGE){
        cnd_wait(prive[i], &m);
    }
    etatsPhilo[i] = MANGE;
    mtx_unlock(&m);
}

poser_fourchettes(int i){
    mtx_lock(&m);
```

```
    etatsPhilo[i] = PENSE;
    int v1 = (i+N-1)%N;
    int v1 = (i+N+1)%N;
    cnd_signal(&prive[v1]);
    cnd_signal(&prive[v2]);
    mtx_inlock(&m);
}
```

1.2 Dijkstra - Semaphore

```
// Philo: un grand saladier de fourchettes + elastique => paire de fourchette
int c; // Compteur
sem_t s = N/2;

Philo(int i){
    while(1){
        pense();
        prendre_fourchette();
        mange();
        poser_fourchette();
    }
}

// sem_wait()
P(){
    c--;
    if (c<0){
        // mettre le thread en attente
    }
}

// sem_post()
V(){
    c++;
    // debloquer un thread bloqué
    if(c <= 0)
}

poser_fouchettes(){
    V(&s);
}
prendre_fourchettes(){
    P(&s);
}
```

Tous les philos ont pris la fourchette de droite

```
// PHILO 1 FAUX
sem_t fourch[N] = {1,1...1};
prendre_fourchettes(i){
    P(&fourch[i]);
    P(&fourch[(i+1)%N])
}

poser_fourchettes(i){
    V(&fourch[i]);
    V(&fourch[(i+1)%N]);
}
```

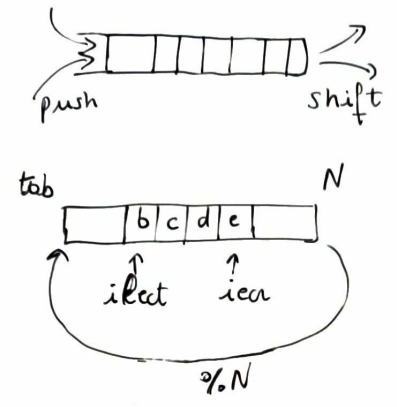
```
// PHILO 2 : 1 gaucher à choisir par le programmeur
// N-1 au lieu de 0 pour prendre les ressources dans l'ordre pour eviter deadlock
prendre_fourchette(i){
    if (i==N-1){
        P(&fourch[(i+1)%N]);
        P(&fourch[i]);
    } else {
        P(&fourch[i]);
        P(&fourch[(i+1)%N]);
    }
}

poser_fourchette(i){
    V(&fourch[i]);
    V(&fourch[(i+1)%N]);
}
```

```
// PHILO 3: sas à N1
sem_t sas = N - 1;
prendre_fourchette(i){
    P(&sas);
    P(&fourch[i]);
    P(&fourch[(i+1)%N]);
    V(&sas);
}

poser_fourchette(i){
    V(&fourch[i]);
    V(&fourch[(i+1)%N]);
    V(&sas);
}
```

```
// Rendez-vous à 2
sem_t nb_a = 0;
sem_t nb_b = 0;

arriveeA(){
    V(&nb_a);
    P(&nb_b);
}

arriveeB(){
    V(&nb_b);
    P(&nb_a);
}
```

```
// Philo 4: 3 états
enum EtatPhilo{MANGE, PENSE, AFAIM};
EtatPhilo etatsPhilo[N] = {PENSE, ..., PENSE};
sem_t sprive[N] = {0,..,0};
sem_t ms = 1; // Exclusion mutuelle

test(i){
    if (EtatPhilo[(i+1)%N] != MANGE && EtatPhilo[(i-1)%N] != MANGE
            && EtatPhilo[i] == AFAIM){
        EtatPhilo[i] = MANGE;
        V(sprive[i]);
    }
}

prendre_fourchette(i){
    // (Changement d'état + test) en exclussion mutuelle
    P(&ms);
    etatPhiio[i] = AFAIM;
    test(i);
    V(&ms);

    // Blocage
    P(&sprive[i]);
}

poser_fourchette(i){
    // En exclusion mutuelle
    P(&ms);
    etatPhilo[i] = PENSE;
    test((i+1)%N);
    test((i-1)%N);
    V(&ms);
}
```

1.3 Producteurs Consomateurs




```
sem_t ms = 1;
sem_t pleines = 0;
sem_t vides = N;
int ilect, iecr;
Tampon tab[N];

push(Msg msg){
    // Verifie s'il existe des cas vide puis se mettre en exclu mut
    P(&vides);
    P(&ms)

    // Ecrire le msg
    tab[iecr] = msg;
    iecr++;
    iecr %= N;

    // Lacher l'exclu mut
    V(&ms);
    V(&pleines)
}

MsgShift(){
    P(&pleine);
    P(&ms);

    Msg msg = tab[ilect];
    ilect = (ilect+1)%N;

    V(&ms);
    V(&vide);
    return msg;
}
```

1.4 Producteurs consommateurs 2 (Lire/Ecrire en même temps)

Utiliser deux semaphores qui permettent de lire et ecrire en même temps sans mélanger.

```
sem_t ms = 1;
sem_t ms2 = 1;
sem_t pleines = 0;
sem_t vides = N;
int ilect, iecr;
Tampon tab[N];

push(Msg msg){
    // Verifie s'il existe des cas vide puis se mettre en exclu mut
    P(&vides);
    P(&ms)

    // Ecrire le msg
    tab[iecr] = msg;
    iecr++;
    iecr %= N;

    // Lacher l'exclu mut
    V(&ms);
    V(&pleines)
}

MsgShift(){
    P(&pleine);
    P(&ms2);

    Msg msg = tab[ilect];
    ilect = (ilect+1)%N;

    V(&ms2);
    V(&vide);
    return msg;
}
```

1.5 Lecteurs redacteurs

Le but est de pouvoir lire à plusieurs mais d'ecrire tout seul.

```
int nblect = 0;
sem_t ms = 1;
sem_t acces_BD = 1; // Acces à la base de données
sem_t sas = 1; // FIFO aux redacteurs

debut_lire(){
    P(&sas);
    P(&ms);
    nblect++;
    if (nblect == 1){
        P(&acces_BD);
    }
    V(&ms);
    V(&sas);
}

fin_lire(){
    P(&ms);
    nblect--;
    if (nblect == 0){
        V(&acces_BD);
    }
    V(&ms);
}

debut_ecr(){
    P(&sas);
    P(&acces_BD);
    V(&sas);
}

fin_ecr(){
    V(&acces_BD);
}
```