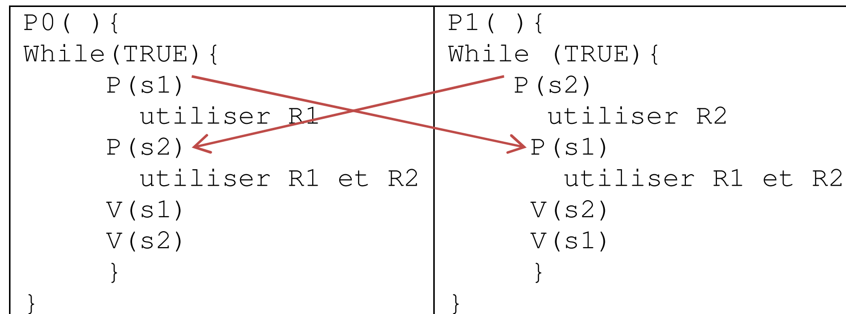
UNIVERSITE A.MIRA-BEJAIA
FACULTE DES SCIENCES EXACTES
DEPARTEMENT INFORMATIQUE

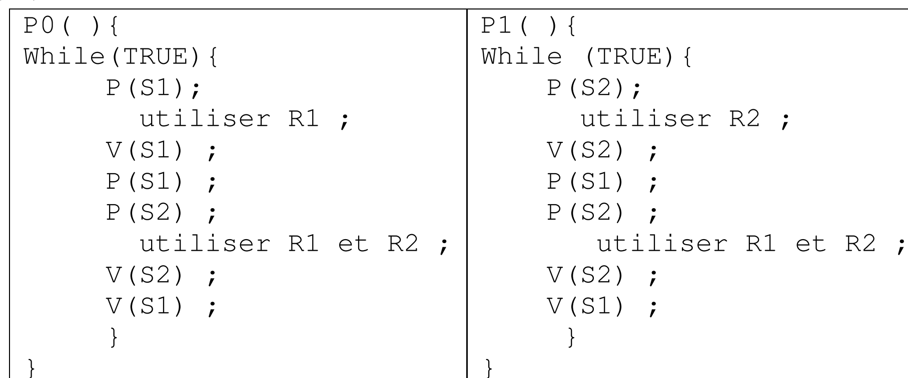
Série 2 : Synchronisation

Exercice1 : 1) Il y'a un risque d'interblocage.
Cas simultané : P1 et P2 arrivent en même temps, P1 prend le ticket de S1 et utilise R1, P2 prend le ticket de S2 et utilise R2, puis P1 sera bloqué par S2 en attente de P2, et P2 sera bloqué par S1en attente de P1.



P0( ){
While(TRUE){
P(s1)
utiliser R1
P(s2)
utiliser R1 et R2
V(s1)
V(s2)
}
}
	P1( ){
	While (TRUE){
	P(s2)
	utiliser R2
	P(s1)
	utiliser R1 et R2
	V(s2)
	V(s1)
	}
	}

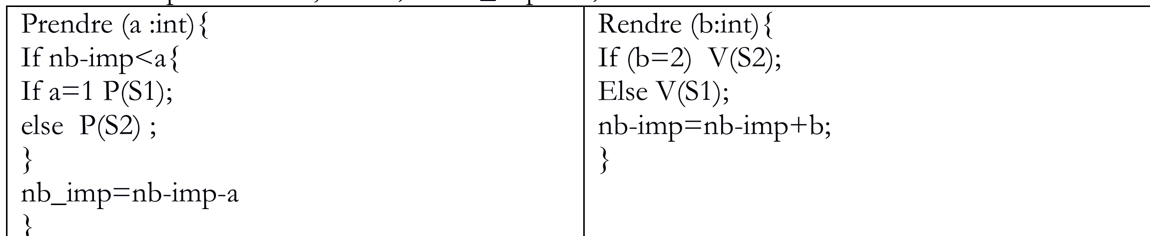
2) solution :



P0( ){
While(TRUE){
P(S1);
utiliser R1 ;
V(S1) ;
P(S1) ;
P(S2) ;
utiliser R1 et R2 ;
V(S2) ;
V(S1) ;
}
}
	P1( ){
	While (TRUE){
	P(S2);
	utiliser R2 ;
	V(S2) ;
	P(S1) ;
	P(S2) ;
	utiliser R1 et R2 ;
	V(S2) ;
	V(S1) ;
	}
	}

Exercice2 : Deux types de processus qu'on ne peut mettre dans la même fila d'attente⟹ 2 sémaphores : S1 : file d'attente des processus demandants 1 imprimante et S2 : file d'attente des processus demandants 2 imprimantes.

Shared semaphore S1=0 , S2=0 ; int nb_imp=N ;



Prendre (a :int){
If nb-imp<a{
If a=1 P(S1);
else P(S2) ;
}
nb_imp=nb-imp-a
}
	Rendre (b:int){
	If (b=2) V(S2);
	Else V(S1);
	nb-imp=nb-imp+b;
	}

Cette solution à l'air correcte mais les processus rendant les imprimantes exécutant des V sans que des P soient obligatoirement exécutés alors il y'aura génération de plus de tickets qu'il n'en faut. Ici S1 et S2 sont initialisés à 0 car on va les utiliser comme file d'attente. Avant qu'un processus n'exécute V(S) on doit être sûr qu'il y'a un processus bloqué dans la file d'attente de S !

Shared semaphore S1=0, S2=0 ; int nb_imp=N ;int cpt-S1=0, cptS2=0;

Prendre (a :int){
If nb-imp<a{
If a=1 { cptS1++ ; P(S1);cptS1-- ; }
else { cptS2++ ; P(S2) ;cptS2-- ; }
}
nb_imp=nb-imp-a ;
}
	Rendre (b:int){
	If (b=1) { if (cpS1>0) V(S1);}
	Else {if cptS2>0 V(S2);
	Else {if cptS1>0 V(S1);
	if cptS1>0 V(S1);}
	}
	nb-imp=nb-imp+b;
	}

Les procédures prendre et rendre sont exécutées par plusieurs processus et cptS1, cptS2 et nb_imp sont des variables partagées donc section critique donc nécessitent un sémaphore mutex (un seul !). Mais comme S1 et S2 sont bloquants il faut rendre le ticket de mutex avant qu'un processus ne soit mis en file d'attente. Voici la solution finale.
Shared semaphore S1=0 , S2=0, mutex=1 ; int nb_imp=N ;int cpt-S1=0, cptS2=0;

Prendre (a :int){
P(mutex);
If nb-imp<a{
If a=1 { cptS1++ ; V(mutex); P(S1); P(mutex); cptS1-- ; }
else { cptS2++ ; V(mutex); P(S2) ;P(mutex); cptS2-- ; }
nb_imp=nb-imp-a ;
V(mutex);
}
	Rendre (b:int){
	P(mutex);
	If (b=1) { if (cpS1>0) V(S1);}
	Else {if cptS2>0 V(S2);
	Else {if cptS1>0 V(S1);
	if cptS1>0 V(S1);}
	}
	nb-imp=nb-imp+b;
	V(mutex);
	}

Exercice3 :Producteur consommateur avec un registre de taille infini donc le producteur ne se bloque jamais. Pas besoin du sémaphore vide

Shared semaphore mutex=1 ; plein :=0 ; ~~vide :=n~~ ;

Prod() {while(true){
produire(objet) ;
~~P(vide)~~ ;
P(mutex) ;
mettre (objet, buffer) ;
V(mutex) ;
V(plein) ;
}}
	Cons() {while(true){
	P(plein) ;
	P(mutex) ;
	objet=retirer(buffer) ;
	V(mutex) ;
	~~V(vide) ;~~
	consommer(objet);
	}}

Exercice4 : P1 est un producteur et P2 son consommateur. P2 devient producteur après avoir traité le message et P3 son consommateur.
1) Shared semaphore S1=N1, S2=0, S3=N2, S4=0, mutex1=1, mutex2=1 ; type message m, m'

P1( ){ while(TRUE){ Produire(message); P(S1); P(mutex1); déposer(m, T1); V(mutex1); V(S2) ; }}	P2( ){While(TRUE){ P(S2) ; P(mutex1) ; m=prélever(T1) ; V(mutex1) ; V(S1) m'=traiter(m) ; P(S3) ; P(mutex2); déposer(m',T2); V(mutex2) ; V(S4) ; }}	P3( ) {While(TRUE){ P(S4) ; P(mutex2); m'=prélever(T2); V(mutex2) ; V(S3) ; }}

2) C'est la même solution sauf qu'au lieu d'avoir T1 et T2 on a un seul registre T, on a plus besoin de deux sémaphores mutex1 et mutex2 ! un seul suffit mutex (initialisé à 1), la synchronisation reste la même.
Shared semaphore S1=N1, S2=0, S3=N2, S4=0, mutex=1 ; type message m, m'

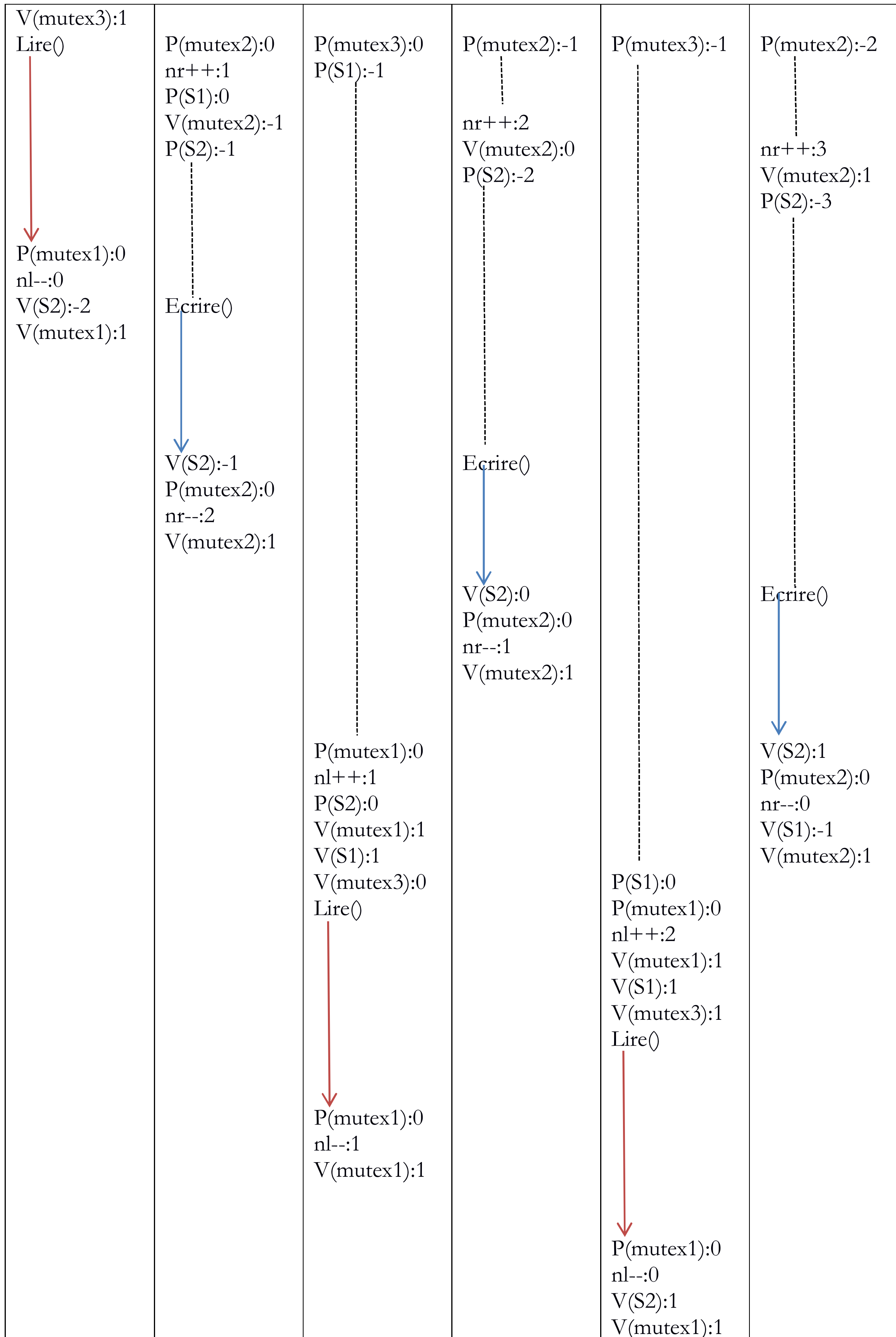
P1( ){ while(TRUE){
Produire(message);
P(S1);
P(mutex);
  déposer(m, T);
V(mutex);
V(S2) ;
}}
	P2( ){While(TRUE){
	P(S2) ;
	P(mutex) ;
	  m=prélever(T) ;
	V(mutex) ;
	V(S1)
	m'=traiter(m) ;
	P(S3) ;
	P(mutex);
	  déposer(m',T);
	V(mutex) ;
	V(S4) ;
	}}
		P3( ) {While(TRUE){
		P(S4) ;
		P(mutex);
		  m'=prélever(T);
		V(mutex) ;
		V(S3) ;
		}}

Exercice5
A) FIFO : Lect1 ; Redact1 ; Lect2 ; Redact2 ; Lect3 ; Redact3
PrioLec : Lect1||Lect2||Lect3 ; Redact1 ; Redcat2 ; Redact3
PrioRed : Lect1 ; Redact1 ; Redact2 ; Redact3 ; Lect2||Lect3
PrioLecSFRed : Lect1 ; Redact1 ; Lect2||Lect3 ; Redact2 ; Redact3

B)FIFO : Lect1(0,10); Redact1(5,40); Lect2(9,50); Redact2(15,80); Lect3(19,90); Redact3(22,120)
PrioLec : Lect1(0,10)||Lect2(9,19)||Lect3(19,29); Redact1(5,59); Redact2(15,89); Redact3(22,119)
PrioRed : Lect1(0,10); Redact1(5,40); Redact2(15,70); Redact3(22,100); Lect2(9,110)||Lect3(19,110)
PrioLecSFRed: Lect1(0,10); Redact1(5,40); Lect2(9,50)||Lect3(19,50); redact2(15,80); Redact3(22,110)

C) Déroulement sur la séquence de processus donné dans A : nl :nbr_lect et nr : nbr_red. A partir du déroulement on déduit :
- Ordre d'exécution: L1;R1;R2;R3;L2||L3
- Variante: priorité aux rédacteurs
- Contraintes: Ecriture en EM et lecture simultané.
- Utilité des variables : mutex1 : protéger nbr_lect ; mutex2 : protéger nbr_red ;
S2 : protéger l'écriture et garantir la contrainte d'EM ;
mutex3 : protéger toute la section d'entrer du lecteur et garantir la lecture simultané;
S1 : contient le premier lecteur (dans le cas d'une écriture en cours), ou le premier rédacteur (dans le cas d'une lecture en cours).

Lect1	Redact1	Lect2	Redact2	Lect3	Redact3
P(mutex3) :0 P(S1) :0 P(mutex1) :0 nl++:1 P(S2):0 V(mutex1):1 V(S1):1					



V(mutex3):1 Lire() P(mutex1):0 nl--:0 V(S2):-2 V(mutex1):1	P(mutex2):0 nr++:1 P(S1):0 V(mutex2):-1 P(S2):-1 Ecrire() V(S2):-1 P(mutex2):0 nr--:2 V(mutex2):1	P(mutex3):0 P(S1):-1 P(mutex1):0 nl++:1 P(S2):0 V(mutex1):1 V(S1):1 V(mutex3):0 Lire() P(mutex1):0 nl--:1 V(mutex1):1	P(mutex2):-1 nr++:2 V(mutex2):0 P(S2):-2 Ecrire() V(S2):0 P(mutex2):0 nr--:1 V(mutex2):1	P(mutex3):-1 P(S1):0 P(mutex1):0 nl++:2 V(mutex1):1 V(S1):1 V(mutex3):1 Lire() P(mutex1):0 nl--:0 V(S2):1 V(mutex1):1	P(mutex2):-2 nr++:3 V(mutex2):1 P(S2):-3 Ecrire() V(S2):1 P(mutex2):0 nr--:0 V(S1):-1 V(mutex2):1

Exercice6 : Problème du coiffeur endormi. On va utiliser un sémaphore (init à 0) pour faire dormir le coiffeur et le faire réveiller par un client (le premier). On aura besoin d'un compteur de clients (protégé par un mutex) pour voir qui est le premier. On aura besoin d'un sémaphore (init à 0) pour faire attendre les clients. Un client doit rendre mutex avant de bloquer !
C'est un problème à 2 types de processus coopérants : Le client a besoin du coiffeur (mais pas l'inverse).
Shared semaphore: dormir = 0, mutex = 1, chaise = 0;
Shared int: nbr_client = 0;

Processus coiffeur(){
While (TRUE){
P(dormir) ;
Coiffer() ;
}}
	Processus client(){
	P(mutex)
	if (nb_client>N) {V(mutex);exit();}
	nbr_client++;
	If(nbr_client >1) { V(mutex;
	P(chaise) ;
	P(mutex);}
	V(mutex);
	V(dormir) ;
	Sefairecoiffer() ;
	P(mutex) ;
	nbr_client - -
	if (nbr_client>0) V(chaise);
	V(mutex);
	}

Problème du Carrefour (cours)

Les processus voitures (2 types) sont en compétition (comme des rédacteurs dans la première question puis comme des lecteurs dans la 2eme question). L'accès est donné par le 3eme processus Feux. La variable booléenne « a » indique l'accès et les feux change toute les m minutes (par exemple 10).

1) une seule voiture dans le carrefour à la fois : ici un simple mutex1/2 suffit (pour chacun) pour donner l'accès. Sempahore feu1=1 et feu=0 indique qu'initialement on donne l'accès à traversée1.

Shared semaphore mutex1=1, mutex2=1, feu1=1, feu2=0; Boolean: a=TRUE; Int m=10;

Traversée1(){
P(mutex1);
P(feu1);
Rouler() ;
V(feu1) ;
V(mutex1) ;
}
	Feux(){While(TRUE){
	Wait(m) ;
	If(a){P(feu1); V(feu2);}
	else{P(feu2);V(feu1);}
	a=¬a;
	}}
		Traversée2(){
		P(mutex2);
		P(feu2);
		Rouler() ;
		V(feu2) ;
		V(mutex2) ;
		}

2) k voitures à la fois dans le carrefour : mutex1/2 est init à k, et on a besoin d'un sémaphore W pour laisser passer la dernière voiture (il y'en a k) quand les feux change. On a besoin d'un compteur de voiture (protégé par mutex) pour voir quelle est la première et quelle est la dernière.

Shared semaphore mutex1=k, mutex2=k, feu1=0, feu2=0, mutex=1, W=0;
Boolean: a=TRUE;
Int m=10, nv=0;

Traversée1( ){
P(mutex1);
P(feu1);
P(mutex) ;
nv++ ;
if(nv==1) P(W) ;
V(mutex);
V(feu1);
Rouler() ;
P(mutex) ;
nv-- ;
if(nv==0) V(W) ;
V(mutex) ;
V(mutex1) ;
}
	Feux( ){While(TRUE){
	Wait(m) ;
	If(a){P(feu1);
	P(W);
	V(feu2);
	V(W);}
	else{P(feu2);
	P(W);
	V(feu1);
	V(W);}
	a=¬a;
	}}
		Traversée2( ){
		P(mutex2);
		P(feu2);
		P(mutex) ;
		nv++ ;
		if(nv==1) P(W) ;
		V(mutex);
		V(feu2);
		Rouler() ;
		P(mutex) ;
		nv-- ;
		if(nv==0) V(W) ;
		V(mutex) ;
		V(mutex2) ;
		}