Corr DS SE&PC 2018/2019

Ex0 :
Famine : Des processus attendent infiniment une ressource partagée sans pouvoir y accéder
Interblocage : Tous les processus sont bloqués et ne peuvent pas avancer

Ex1 :
1) 

2) affichage possible :
Mon nom est <**A**> Mon nom est <**B**> Mon nom est <**C**> Mon nom est <**D**> Mon nom est <**E**>

3) modification : ajouter **wait(NULL) ;** juste avant le printf.

Ex2 :

1) Les résultats corrects de l'exécurion de PA et PB sont obtenues losqu'on exécute PA puis PB ou PB puis PA. C'est à dire {PA avec k=2, PB avec k=6 ⇒ (2,6)} ou {PB avec k=5, PA avec k=6 ⇒ (5,6)}

2)
R1 (2,2) possible et non correcte car elle est différente de (2,6) et (5,6):
⇒ Exécution de PA1 ; PA2 ; PB1 ; PB2 ; PB3 ; PA3 ; PA4; PB4

R2 (2,6) possible et correcte car équivalente à PA puis PB
R3 (5,2) impossible et non correcte car les deux processus incrémentent k ⇒ la 2ème valeur doit être supérieure à la première

3)
a)possible et correcte : PB ;PA ⇒ (5,6)
b) possible et incorrecte : voir R1 de la question précédente

4)
public int k=1;
public semaphore Mutex=1 ; // ou sem_t Mutex=1 ;

PA	PB

int main ()
{
int i;
P(Mutex); //ou bien sem_wait(Mutex) ;
i=k;
i=i+1;
k=i;
printf("PA avec k=%d\n",k);
V(Mutex);//ou bien sem_post(Mutex) ;
...
}
	int main ()
	{
	int j;
	P(Mutex);
	j=k;
	j=j+4;
	k=j;
	printf("PB avec k =%d\n",k);
	V(Mutex);
	...
	}

Ex3 :

1) Oui. Solution : changer dynamiquement les priorité des processus (par ordonnancement) pour avoir de l'équité.
2)
0—P2—3—P3—10—P4—28—P3—29—P2—41—P1—51
On note par TR le temps de réponse ⇒ TR(P1)=51mn ; TR(P2)=41mn ; TR(P3)=26mn ; TR(P4)=18mn TRmoyen = (51+41+26+18)/4 = 40mn

3)
a)

processus	Tcpu	Priorité initiale	t=5	t=10	t=15	t=20	t=25	t=30	t=35	t=40	t=45
P1	10	2	2	1	1	2	2	---			
p2	15	3	1	1	2	0	1	1	1	0	0
P3	8	4	1	1	2	3	0	1	1	2	--
P4	18	5	--	5	1	1	1	2	0	1	0

==> 0-P2-3-P3-5-P1-10-P4-15-P2-20-P3-25-P1-30-P4-35-P2-40-P3-41-P4-45-P2-47-P4-51

b) Oui ça résoud le problème de famine car le processus qui attend beaucoup sera proritaire (voir formule)

EX4 :

Problème de Prod/cons et RDV.
Deux réponses posibles :
Réponse 1: 3 processus process_H, Process_O et Process_H2O ⇒ 3 producteurs (H, H, O) et un consommateur (H2O) + RDV entre 3 processus H H et O
Réponse 2 : 2 processus Process_H et Process_O ⇒ 2 producteurs (H, H) et un consommateur O et RDV entre 2 H

2)
a) Semaphore Hyd=0, Oxy= 0 ; //Sémaphore de blocage pour RDV et P/C
b)

Hydrogene()
{
While (True) {
V(Oxy) ; //Producteur de H
P(Hyd); //RDV avec un 2nd H
}
}
	Oxygene()
	{
	While (True) {
	P(Oxy); //Consommer H
	P(Oxy); //Consommer le 2nd H
	makeWater(); //H2O
	V(Hyd);
	V(Hyd);
	}
	}

3) Si deux H arrivent alors le semaphore Oxy sera égal à 2. Si deux O (ou plus) arrivent et que deux d'entre eux décrémentent Oxy chacun d'une seule unité ⇒ famine

4) Solution : un seul Processus Oxygene exécute, à la fois, le code de la boucle while ⇒ utiliser un troisième sémaphore initialisé à 1. ⇒ Semaphore Mutex=1 ;

Dans Processus Oxygene , ajouter au début du while P(Mutex) et à la fin V(Mutex).