2. 不同。(1)为1，(2)为m。

(1)的变化范围为[1-n, 1]，(2)为[m-n, m]

3. (1)全P1优先：x=10, y=9, z=15

(2)全P2优先：x=10, y=9, z=5

(3)先P1后P2优先：x=10, y=19, z=15

5. (1)

semaphore seats =100,mutex[100];

string name[100];

for(int i=0;i<100;i++){

name[i]=null;

mutex[i]=1;

}

process enterRoom (readername){

P(seats);

for(int i=0;i<100;i++){

P(mutex[i]);

if(name[i]==null){

name[i]=readername;

V(mutex[i]);

break;

}

V(mutex[i]);

}

}

process leaveRoom (readername){

for(int i=0;i<100;i++){

if(name[i]==readername){

P(mutex[i]);

name[i]=null;

V(mutex[i]);

break;

}

}

V(seats);

}

(2)

Type readBook=monitor{

　　 Interface Module IM;

　　 String name[100];

　　 for(i=0;i<100;i++){

　　 name[i]=null;

}

semaphore seat=1;

int seat\_count=100;

　　 int wait\_count=0;

Define come,leave;

Use enter, release, wait, signal;

}

void come(readername){

　　enter(IM);

　　If(seat\_count<=0)  wait(seat,wait\_count,IM);

　　seat\_count--;

　　for(int i=0;i<100;i++){

　　 If(name[i]==null){

　　 name[i]=readername;

　　 release(IM);

break;

　　 }

　　}

}

void leave(readername){

enter(IM);

　　for(int i=0;i<100;i++){

　　 If(name[i]== readername){

　　 name[i]=null;

if(wait\_count!=0){

signal(seat,wait\_count,IM);

}

seat\_count++;

　　 release(IM);

break;

　　 }

　　}

}

6. semaphore black\_ready,white\_ready;

black\_ready=0,white\_ready=1;

int black\_count=white\_count=N;

process get\_white(){

if(white\_count>0){

P(white\_ready);

//捡白棋子;

white\_count--;

V(black\_ready);

}

}

process get\_black(){

if(black \_count>0){

P(black \_ready);

//捡白棋子;

black \_count--;

V(white\_ready);

}

}

17. semaphore m[4],p[4];

m[0]=m[1]=3;

m[2]=m[3]=2;

p[1]=p[2]=p[3]=0;

p[0]=3;

process pass\_i(){ //i=0,1,2,3

while(true){

P(p[i]);

//读取信件;

V(m[i]);

P(m[(i+1)%4]);

V(p[(i+1)%4]);

}

}

24. (1)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | Ci-Ai | | | | Available | | | |
| A | B | C | D | A | B | C | D |
| P0 | 0 | 0 | 1 | 2 | 1 | 6 | 2 | 2 |
| P3 | 0 | 6 | 5 | 2 | 1 | 6 | 6 | 4 |
| P1 | 1 | 7 | 5 | 0 | 1 | 12 | 11 | 6 |
| P2 | 2 | 3 | 5 | 6 | 2 | 18 | 16 | 6 |
| P4 | 0 | 6 | 5 | 6 | 4 | 21 | 21 | 12 |

按P0,P3,P1,P2,P4顺序可以执行，是安全的。

(2)不行。因为分配之后系统就不安全了。

29. 注：把一组缓冲区看成n2 组缓冲区，每个发送者需要同时写n2 组缓冲区中相应的n2 个缓冲区，而每一个接收者只需读它自己对应的那组缓冲区中的对应单元

semaphore mutex; mutex=1;

semaphore empty[n2], full[n2];

for(int i=0;i<n2;i++){

　　 empty[i]=m;

　　 full[i]=0;

}

send(){ //发送消息

　 for(int i=0;i<n2;i++){

　　 P(empty[i]);

　　 P(mutex);

　 //加入缓冲区

　 V(mutex);

　　 }

　　 for(int i=0;i<n2;i++){

　　 V(full[i]);

　　 }

}

receive(){ //接受消息

　　 for(int i=0;i<n2;i++){

　　 P(full[i]);

　　 P(mutex);

　　 //从缓冲区中取出信息

　　 V(mutex);

　　 }

　　 for(int i=0;i<n2;i++){

　　 V(empty[i]);

　　 }

}

39. (1)信号量及PV操作：

semaphore full, empty, p\_mutex, c\_mutex;

full=0; empty=9; p=1; c=1;

int index\_in, index\_out; index\_in=0; index\_out=0;

int buf[9];

process producer(){

　　 //generate number[3]

　　 P(empty);

　　 P(empty);

　　 P(empty);

　　 P(p\_mutex);

　　 buf[index\_in]=number[0];

　　 index\_in=(index\_in+1)mod9;

　　 buf[index\_in]=number[1];

　　 index\_in=(index\_in+1)mod9;

　　 buf[index\_in]=number[2];

　　 index\_in=(index\_in+1)mod9;

　　 V(p\_mutex);

　　 V(full);

　　 V(full);

　　 V(full);

}

process consumer(){

　　 P(full);

　　 P(c\_mutex);

　　 //take number in buf[index\_out]

　　 index\_out=(index\_out+1)mod9;

　　 V(c\_mutex);

　　 V(empty);

}

(2)管程：

type P\_C =monitor{

　　 int buf[9];

InterfaceModule IM;

　　 int index\_in, index\_out,full;

index\_in=index\_out=full=0;

semaphrone producer,consumer;

producer=consumer =1;

int producer\_count,consumer\_count;

producer\_count=consumer\_count=0;

　　 define producer,consumer;

　　 use wait, enter, leave, signal;

}

process producer(){

enter(IM);

//generate number[3]

if(full>6) wait(producer,producer\_count,IM);

full++;

buf[index\_in]=number[0];

　　 index\_in=(index\_in+1)mod9;

　　 buf[index\_in]=number[1];

　　 index\_in=(index\_in+1)mod9;

　　 buf[index\_in]=number[2];

　　 index\_in=(index\_in+1)mod9;

signal(consumer,consumer\_count,IM);

leave(IM);

}

process comsumer(){

enter(IM);

if(full==0){

wait(consumer,consumer\_count,IM);

}

//take number in buf[index\_out]

index\_out=(index\_out+1)mod9;

full--;

if(full<=6){

signal(producer,producer\_count,IM);

}

leave(IM);

}

47. (1)信号量及PV操作：

semaphrone struct\_use,wheel\_use;

struct\_use=wheel\_use=0;mutex=1;

semaphrone struct\_max =N/5;

semaphrone wheel\_max=4N/5;

process make\_struct(){

while(true){

P(struct\_max);

V(struct\_use);

}

}

process make\_wheel(){

while(true){

P(wheel\_max)

V(wheel\_use);

}

}

process make\_car(){

while(true){

P(struct\_use);

//获取车架

V(struct\_max);

P(wheel\_use);

//获取车轮

V(wheel\_max);

P(wheel\_use);

//获取车轮

V(wheel\_max);

P(wheel\_use);

//获取车轮

V(wheel\_max);

P(wheel\_use);

//获取车轮

V(wheel\_max);

//组装汽车

}

}

(2)管程：

Type cabinet=monitor{

InterfaceModule IM;

semaphrone struct\_max,wheel\_max,struct\_use,wheel\_use;

struct\_max=wheel\_max=struct\_use=wheel\_use=0;

int struct\_max\_count=N/5;

int wheel\_max\_count=4N/5;

int struct\_use\_count=0;

int wheel\_use\_count=0;

int struct\_wait\_count=0;

int wheel\_wait\_count=0;

int car\_wait\_count1=car\_wait\_count2=0;

def make\_struct,make\_wheel,make\_car;

use enter,leave,wait,signal;

}

process make\_struct(){

enter(IM);

if(struct\_max\_count==0){

wait(struct\_max,struct\_wait\_count,IM);

}

//做车架

struct\_use\_count++;

if(car\_wait\_count1>0){

signal(struct\_use,car\_wait\_count1,IM);

}

leave(IM);

}

process make\_wheel(){

enter(IM);

if(wheel\_max\_count==0){

wait(wheel\_max,wheel\_wait\_count,IM);

}

//做车轮

wheel\_use\_count++;

if(car\_wait\_count2>0){

signal(wheel\_use,car\_wait\_count2,IM);

}

leave(IM);

}

process make\_car(){

enter(IM);

if(struct\_use\_count==0){

wait(struct\_use,car\_wait\_count1,IM);

}

//获取车架

struct\_use\_count--;

struct\_max\_count++;

if(struct\_wait\_count>0){

signal(struct\_max,struct\_wait\_count,IM);

}

for(int i=0;i<4;i++){

if(wheel\_use\_count==0){

wait(wheel\_use,car\_wait\_count2,IM);

}

//获取车轮

wheel\_use\_count--;

wheel\_max\_count++;

if(wheel\_wait\_count>0){

signal(wheel\_max,wheel\_wait\_count,IM);

}

}

//拼装车辆

leave(IM);

}