Prod-Cons (1 vs 1)

```
init (full, 0);
init (empty, MAX);

Producer () {
Message m;
while (TRUE) {
produce (m);
wait (empty);
enqueue (m);
signal (full);}}

Consumer () {
Message m;
while (TRUE) {
wait (full);
m = dequeue ();
signal (empty);
consume (m);}}
```

Prod-Cons (n vs n)

```
init (full, 0);
init (empty, MAX);
init (MEp, 1);
init (MEc, 1);

Consumer () {
Message m;
while (TRUE) {
wait (full);
wait (MEc);
m = dequeue ();
signal (MEc);
signal (empty);
consuma m;}}

Producer () {
Message m;
while (TRUE) {
produce m;
wait (empty);
wait (MEp);
enqueue (m);
signal (MEp);
signal (full);}}
```

Precedenza Reader

```
nR = 0;
init (meR, 1);
init (w, 1);

wait (meR);
nR++;
if (nR==1)
wait (w);
signal (meR);
...
lettura
...
wait (meR);
nR--;
if (nR==0)
signal (w);
signal (meR);


wait (w);
...
scrittura
...
signal (w);
```

Precedenza ai writer

```
nR = nW = 0;
init (w, 1); init (r, 1);
init (meR, 1); init (meW,
1);

wait (meW);
nW++;
if (nW == 1)  wait (r);
signal (meW);
wait (w);
...scrittura...
signal (w)
wait (meW);
nW--;
if (nW == 0) signal (r);
signal (meW);
wait (r);


wait (meR);
nR++;
if (nR == 1) wait (w);
signal (meR);
signal (r);
...lettura...
wait (meR);
nR--;
if (nR == 0) signal (w);
signal (meR);
```

I 5 Filosofi-Modello 1

```
init (mutex, 1);
while (true) {
Pensa ();
wait (mutex);
Mangia ();
signal (mutex);
}
```

Modello 2

```
init (chopstick[0], 1);
...
init (chopstick[4], 1);

while (true) {
Pensa ();
wait (chopstick[i]);
wait
(chopstick[(i+1)mod5]);
Mangia ();
signal (chopstick[i]);
signal
(chopstick[(i+1)mod5]);}
```

Soluzione finale

```
int state[N]
init (mutex, 1);
init (sem[0], 0); ...; init
(sem[4], 0);

takeForks (int i) {
wait (mutex);
state[i] = HUNGRY;
test (i);
signal (mutex);
wait (sem[i]);}
```

Soluzione finale

```
test (int i) {
if (state[i]==HUNGRY &&
state[LEFT]!=EATING &&
state[RIGHT]!=EATING) {
state[i] = EATING;
signal (sem[i]);}}

putForks (int i) {
wait (mutex);
state[i] = THINKING;
test (LEFT);
test (RIGHT);
signal (mutex);}
```

TUNNEL

```
n1 = n2 = 0;
init (s1, 1); init (s2, 1);
init (busy, 1);

//left2right
wait (s1);
n1++;
if (n1 == 1)
wait (busy);
signal (s1);
Run (left to right)
wait (s1);
n1--;
if (n1 == 0)
signal (busy);
signal (s1);

//rigth2left
wait (s2);
n2++;
if (n2 == 1)
wait (busy);
signal (s2);
Run (left to right)
wait (s2);
n2--;
if (n2 == 0)
signal (busy);
signal (s2);
```

Banchiere ALGORITMO

Verifica di una richiesta da parte di P_i

se
$\forall j$ Richieste[i][j]≤Necessità[i][j] AND
$\forall j$ Richieste[i][j]≤Disponibili[j] ALLORA
$\forall j$ Disponibili[j]=Disponibili[j]-Richieste[i][j]

$\forall j$ Assegnate[i][j]=Assegnate[i][j] +Richieste[i][j]

$\forall j$ Necessità[i][j]=Necessità[i][j]-Richieste[i][j]

se stato risultante è sicuro si conferma assegnazione else si ripristina statoprecedente

Banchiere

1.(verifica di uno stato)
$\forall i \forall j$ Necessita[i][j]=Massimo[i][j]-Assegnate[i][j]
$\forall i$ Fine [i]=false

2.
Trova i per cui
Fine[i]=falso AND $\forall j$ Necessità[i][j]≤Disponibili[j]
Se tale i non esiste goto step 4

3.
$\forall j$ Disponibili[j]=Disponibili[j] +Assegnate[i][j]
Fine[i]=true
goto step 2

4.
Se $\forall i$ Fine[i]=true il sistema è in uno stato sicuro

FIFO – Queue – Buffer Circolare

i

```
void enqueue(int val)
{
if(n>SIZE) return;
queue[tail] = val;
tail=(tail+1)%SIZE;
n++;
return;}
```

j

```
int dequeue(int *val)
{
if(n<=0) return;
*val=queue[head];
head=(head+1)%SIZE;
n--;
return;}
```

LIFO-STACK

i

```
void push (int val) {
if(top<0)return;
stack[top] = val;
top--;
return;
}
```

J

```
int pop (int *val) {
if(top>SIZE-2)return;
top++;
*val=stack[top];
return;
}
```

SOLUZIONE HARDWARE

Mutua esclusione con Test-And-Set

(Starvation)

```
char TestAndSet(char *lock)
{
char val = *lock;
*lock=TRUE;Set new lock
return val; // Return old
lock}
--------------------
char lock=FALSE;
...
while(TRUE) {
while(TestAndSet lock));
// lock
SC
lock = FALSE;// unlock
sezione non critica
}
```

Mutua esclusi con Swap (Starvation)

```
void swap (char *v1, char *v2) {
char = *tmp; *tmp = *v1;
*v1 = *v2;
*v2 = *tmp;
return; }

char lock = FALSE;
...
while (TRUE) {
key = TRUE;
while (key==TRUE)
swap (&lock, &key); // Lock
SC
lock = FALSE; // Unlock
sezione non critica}
```

Mutua esclusione senza starvation

```
while (TRUE) {
waiting[i] = TRUE;
key = TRUE;
while (waiting[i] && key)
key = TestAndSet (lock);
waiting[i] = FALSE;
SC di Pi
j = (i+1) % N;
while ((j!=i) and
(waiting[j]==FALSE))
j = (j+1) % N;
if (j==i) lock = FALSE;
else waiting[j] = FALSE;
sezione non critica
}
```

SOLUZIONE SOFTWARE

Soluzione 1

```
while (TRUE) {
while (turn==j);
SC di Pi
turn = j;
sezione non critica
}

while (TRUE) {
while (turn==i);
SC di Pj
turn = i;
sezione non critica
}
```

Progresso non assicurato

Soluzione 2(lock)

```
while (TRUE) {
while (flag[j]);
flag[i] = TRUE;
SC di Pi
flag[i] = FALSE;
sezione non critica}

while (TRUE) {
while (flag[j]);
flag[i] = TRUE;
SC di Pi
flag[i] = FALSE;
sezione non critica}
```

Mutua esclusione non assicurata

Soluzione 3

```
while (TRUE) {
flag[i] = TRUE;
while (flag[j]);
SC di Pi
flag[i] = FALSE;
sezione non critica
}

while (TRUE) {
flag[j] = TRUE;
while (flag[i]);
SC di Pj
flag[j] = FALSE;
sezione non critica
}
```

Attesa non definita DEADLOCK

Soluzione 4

```
while (TRUE) {
flag[i] = TRUE;
turn = j;
while (flag[j] &&
turn==j);
SC di Pi
flag[i] = FALSE;
sezione non
critica
}

while (TRUE) {
flag[j] = TRUE;
turn = i;
while (flag[i] &&
turn==i);
SC di Pj
flag[j] = FALSE;
sezione non
critica
}
```

Soluzione corretta

condizione di attesa circolare
implica che esiste un insieme di processi tale pe cui ogni processo dell'insieme attende una risorsa
posseduta da un altro processo dell'insieme
relazione di ordinamento totale tra i
vari tipi di risorse, associando a ciascuno di essi un
numero intero
Se F(Rnew) > F(Rold)
La risorsa viene concessa
HRU
$F(Rk) < F(Rk+1), \forall\ k = 0 .. n - 1$
cioè
$F(R0) < F(R1) < ...< F(Rn) < F(R0)$
$F(R0) < F(R0)$
che è assurdo

Mutua Esclusione

Uno stallo si verifica quando un processo rimaneindefinitiv in attesa su una risorsa non condivisibile

Proebire risorse non condivisibili

Proebire l'attesa su risorse non condivisibili

Possesso e attesa

si verifica quando un processo possiede una o più risorse e ne chiede di ulteriori

Request All First (RAF)
Release Before Request (RBR)

Impossibilità di prelazione

significa che una risorsa non può essere sottratta a un processo
-permettere la prelazione delle risorse possedute dal processo stesso
- permettere la prelazione di risorse possedute da un altro processo purchè esso sia in ase di attesa

Impl Semaforo

```
typedef struct semaphore_tag {
int cnt; // Numero processi
process_t *head; // Lista processi
} semaphore_t;
init (semaphore_t S, int k) {
S.cnt = k; // k>=0
}
destroy (semaphore_t S) {
while (S.cnt>=0) {
free S.head
S.cnt--;
}
wait (semaphore_t S) {
S.cnt--; // May be <0
if (S.cnt<0) {
push P to S.head
block P
}}
signal (semaphore_t S) {
S.cnt++;
if (S.cnt<=0) {
pop P from S.head
wakeup P
}}
```

cut [options] [file]
-d " " il delimitatore è lo spazio
-f 1,3 seleziona i campi 1 e 3 di tutte le righe
-c a cancella tutte le a

tr [options] set1 [set2]
tr –d ab< "file" Visualizza righe dove eliminati i a, b

uniq [options] [inFile] [outFile]
-c Stampa numero ripetizioni prima della riga
-d Visualizza solo le righe ripetute
-f N Ignora i primi N campi per il confronto

sort [opzioni] [file]
-b ignora gli spazi iniziali
-d Considera solo spazi e caratteri alfabetici
-f Trasforma caratteri minuscoli in maiuscoli
-n Confronta utilizzando un ordine numerico
-r Ordine inverso
-k c1[,c2] Ordina sulla base dei campi selezionati
-m Merge file già ordinati (senza riordinare)
-o=f Scrive l'output in f invece che su stdout

grep[opzioni] pattern [file]
-A N Dopo ciascun match stampa N righe
-H Stampa il nome del file per ogni match
-I Case insensitive
-n Stampa il numero di riga del match
-R Procede in maniera ricorsiva
-v Stampa solo righe che nn fanno match