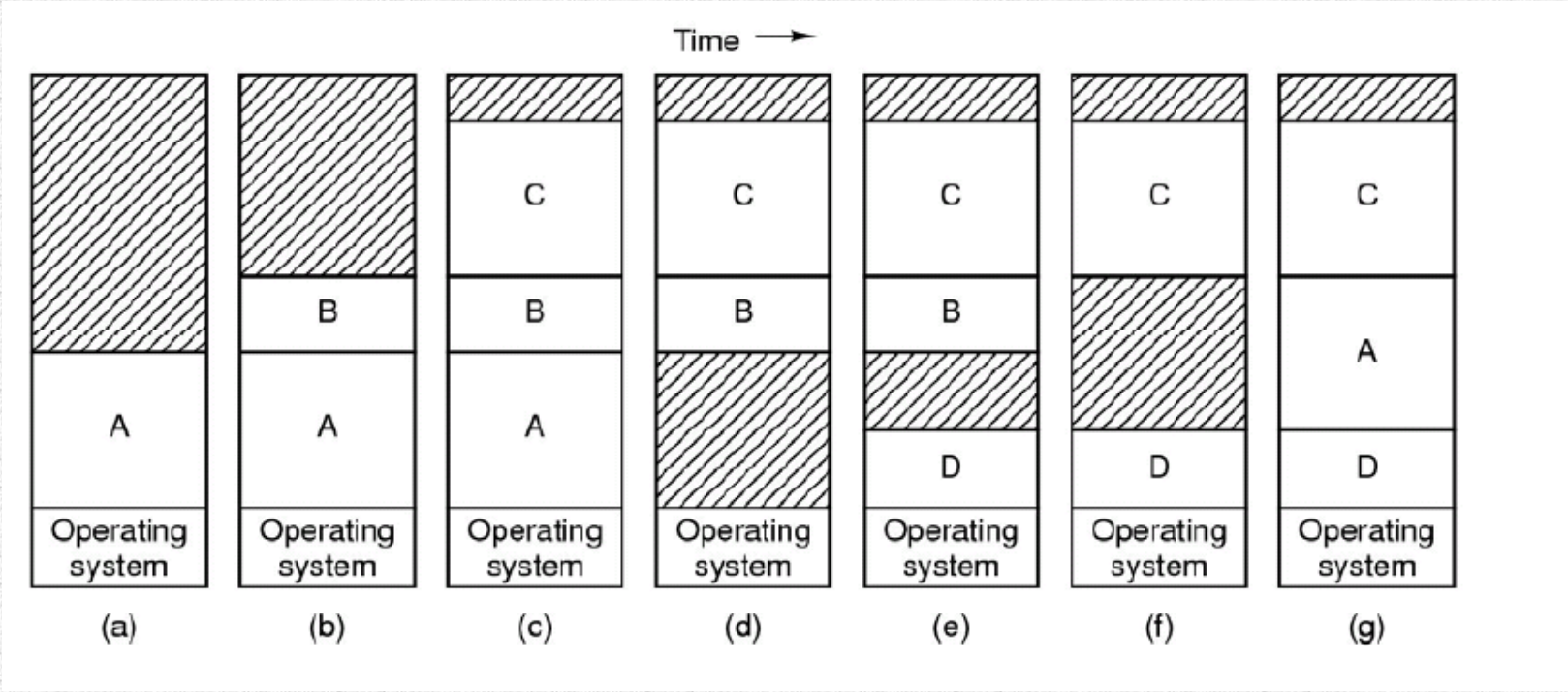
Gestione della Memoria

Uno dei compiti di un S.O. è quello di gestire la memoria che rappresenta una delle due risorse indispensabili all'elaborazione.
Le problematiche sono:

- Necessità di allocare in memoria lo spazio di lavoro di più processi
- Anche in un contesto uniprogrammato la memoria e condivisa tra processo corrente e sistema operativo
- In un contesto di multiprogrammazione i processi sono molti, e la commutazione deve essere veloce
- Più processi in memoria significa meno probabilità che la CPU sia inattiva: migliore throughput



Quando la memoria è assegnata dinamicamente ai vari processi, si utilizza una struttura dati per mantenere informazioni sulle zone libere e sulle zone occupate.

Le strutture dati possibili sono:

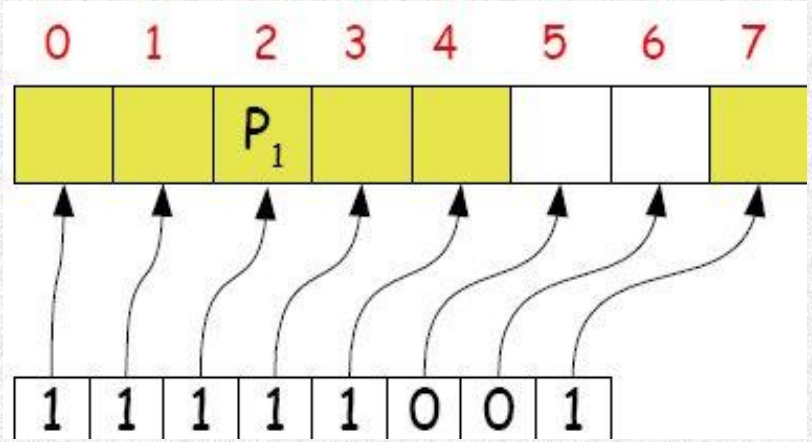
- mappe di bit
- liste di puntatori

Mappe di bit

La memoria viene suddivisa in unità di allocazione.

Ad ogni unità di allocazione corrisponde un bit in una bitmap.

Le unità libere sono associate ad un bit di valore 0, le unità occupate sono associate ad un bit di valore 1.



Nel nostro caso consideriamo che lo spazio di memoria sia di K unità.

La mappa di bit può essere rappresentata in C con un vettore di k caratteri (K Byte)

```
insigned char mapp[k]; //ossia con 8*k bit
```

oppure con un vettore in cui ogni bit è rappresentato da un singolo bit (K/8 Byte)

```
insigned char mapp[k/8];  //ossia con k bit
```

La gestione della mappa mediante la prima rappresentazione è molto semplice.

Nel secondo caso bisogna adoperare una maschera creata con l'operatore shift >> e l'operatore and & per trovare il bit corrispondente.

Ad esempio

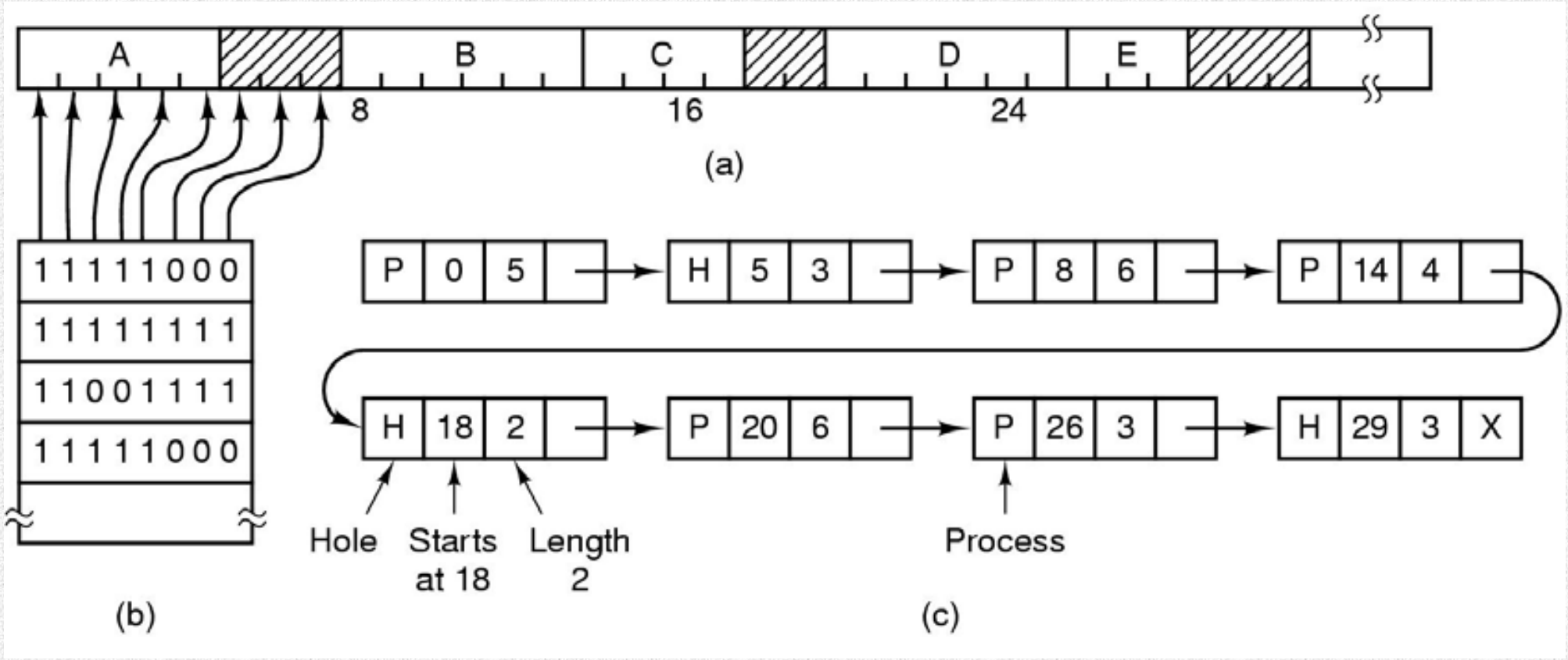
```
// Bini Blocco iniziale della memoria da occupare
// Lungh Lunghezza della memoria da occupare
//mem Memoria totale
//esempio BIni=5 Lungh=9
char a=128;
if( (BIni>mem) || (BIni+Lungh>mem))
    printf("Blocco di memoria non consentito!!!! \n");
 else
 { error=0;
   BiniT=BIni/8; //BiniT=0
   offset=BIni-(BiniT*8); //offset=5-0=5
   a=a>>offset; //a=00001000
   j=BiniT;
```

```
//ciclo di controllo se la memoria è occupata
   for(i=BIni;i<BIni+Lungh && error ==0;i++)
    {if((i>Bini) && (i%8)==0) {j++; a=128;}
      if( (mappa[j]&a)>0) error=1;
      a=a>>1; }
   if(error==0)
   {a=128;
    a=a>>offset;
    j=BiniT;
    for(i=BIni;i<BIni+Lungh;i++)
     {if((i>Bini) && (i%8)==0) {j++; a=128;}
      mappa[j]=mappa[j]|a;
      a=a>>1;}
   }
 else
      printf("Blocco di memoria occupato\n");
```

Liste di puntatori (singola lista)

Si mantiene una lista dei blocchi allocati e liberi di memoria.Ogni elemento della lista specifica consiste in:

- Un capo di tipo carattere
  - valore P se si tratta di un blocco che appartiene ad un processo
  - valore H se si tratta di un blocco libero (Hole)
- Un long Blocco Iniziale
- Un long Lunghezza del Blocco
- Puntatore al prossimo Blocco



(a) Situazione della memoria: 5 partizioni e tre buchi
(b) Gestione con mappa di bit: 1 occupato 0 libero
(c) Gestione con liste: P partizione, H hole (buco)

In Pratica si utilizza una struttura i cui campi sono:

IP	unsigned long identificatore unico del processo>0 o buco=0
BIni	unsigned long blocco iniziale della memoria
BEnd	unsigned long blocco finale della memoria
Next	link al blocco successivo

La cui struttura in C potrebbe essere questa

```
typedef struct PROC_CODA
{unsigned long IP;
 unsigned long BIni;
 unsigned long BEnd;
 PROC_CODA *next;
} proc_coda;
```

Alla presenza di un nuovo processo la lista si modificherà opportunamente.

Esempio

Si supponga che lo spazio di memoria sia di 300 Blocchi quindi all'inizio la coda della memoria sia in questa configurazione:

[0|0|300]->NULL

Inseriamo il processo con pid=1, BIni=0, e BEnd=20

la coda della memoria si trasforma in:

[1|0|20]->[0|21|300]->NULL

Inseriamo il processo con pid=2,BIni=200, e BEnd=250

[1|0|20]->[0|21|199]->[2|200|250]->NULL

Liste di puntatori (doppia lista)

Due liste una dei blocchi allocati e una dei blocchi liberi di memoria.

Ogni elemento della lista specifica la dimensione (inizio/fine) del segmento

I campi della coda dei Processi sono:

IP	unsigned long identificatore unico del processo>0
BIni	unsigned long blocco iniziale della memoria occupata
BEnd	unsigned long blocco finale della memoria occupata
Next	link al Blocco successivo

La cui struttura in C potrebbe essere questa

```
typedef struct PROC_CODA
{unsigned long IP;
 unsigned long BIni;
 unsigned long BEnd;
 PROC_CODA *next;
} proc_coda;
```

I campi della coda dei blocchi liberi dalla Memoria sono:

BIni	unsigned long blocco iniziale della memoria libera
BEnd	unsigned long blocco finale della memoria libera
Next	link al Blocco successivo

La cui struttura in C potrebbe essere questa

```
typedef struct H_CODA
{unsigned long BIni;
 unsigned long BEnd;
 H_CODA *next;
} h_coda;
```

Alla presenza di un nuovo processo le liste si modificano opportunamente.

Esempio

Si supponga che lo spazio di memoria sia di 300 Blocchi quindi all'inizio le coda saranno in questa configurazione

[0|300]->NULL //Coda dei blocchi liberi

NULL //Coda dei processi

Inseriamo il processo con pid 1, BIni 0, e BEnd 20

[21|300]->NULL //Coda dei blocchi liberi

[1|0|20]->NULL//Coda dei processi

Inseriamo il processo con pid 2, BIni 200, e BEnd 250

[21|199]->[251|300]->NULL //Coda dei blocchi liberi

[1|0|20]->[2|200|250]->NULL //Coda dei processi

Inseriamo il processo con BIni 280, e BEnd 300

[1|0|20]->[2|200|250]->[3|280|300]->NULL

[20|199]->[251|279]->NULL