Gestione della Memoria

In questa lezione vediamo un possibile codice della gestione della memoria sempre col la rappresentazione a mappa di bit con la metodologia del First Fit.

```
#include <stdio.h>
#include <malloc.h>
#define MaxProc 20
void Inserisci(unsigned char *mappa,unsigned long mem,unsigned
long Pid[][MaxProc]);
void Elimina(unsigned char *mappa,unsigned long mem,unsigned long
Pid[][MaxProc]);
void visualizza(unsigned char *mappa,unsigned long mem,unsigned
long Pid[][MaxProc]);
unsigned long CercaBloccoLibero(unsigned char *mappa,unsigned long
mem,unsigned long Lengh);

int main(int argc, char* argv[])
{unsigned char *mappa;
 unsigned long Pid[MaxProc][MaxProc];
 unsigned long i,mem,memT;
```

```
 printf("dai la quantità di memoria disponibile in k:");
 scanf("%d",&mem);
 memT=(mem/8)+1;
 mappa=(unsigned char *)malloc(sizeof(char)*memT);
 for(i=0;i<memT;i++) mappa[i]=0;
 for(i=0;i<MaxProc;i++) Pid[i][1]=0;
 char scelta[2];
 do
 {do{printf("Per inserire  un processo : I\nPer eliminare il
processo : E\nPer Uscire :s\nScelta:");
  scanf("%s",scelta);
  if( (scelta[0]!='I') && (scelta[0]!='E') && (scelta[0]!='s'))
   printf("Scelta non consentita\n\n");
 }
 while( (scelta[0]!='I') && (scelta[0]!='E') && (scelta[0]!='s'));
 if(scelta[0]=='I') Inserisci(mappa,mem,Pid);
 if(scelta[0]=='E') Elimina(mappa,mem,Pid);
 }while(scelta[0]!='s');
 return 0;
}
```

```
void Inserisci(unsigned char *mappa,unsigned long mem,unsigned
long Pid[][MaxProc])
{unsigned long i,j;
 unsigned long BIni,BiniT,offset;
 unsigned long Lungh;
 unsigned char a=128;
 printf("Dai la memoria occupata dal Processo:");
 scanf("%d",&Lungh);
 BIni=CercaBloccoLibero(mappa,mem,Lungh);
 if( (BIni>mem) || (BIni+Lungh>mem))
  printf("Blocco di memoria non consentito! \n");
 else
 {a=128;BiniT=BIni/8; offset=BIni-(BiniT*8);
  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;
   }
  for(i=0;i<=MaxProc && Pid[i][1]>0;i++);
  Pid[i][0]=BIni;Pid[i][1]=Lungh;
  visualizza(mappa,mem,Pid);
 }
}
```

```
void Elimina(unsigned char *mappa,unsigned long mem,unsigned long
Pid[][MaxProc])
{unsigned long i,j;
 unsigned long BIni,BiniT,offset;
 unsigned long Proc;
 unsigned long Lungh;
 unsigned char b,a=128;
 printf("Dai il pid del Processo da eliminare:");
 scanf("%d",&Proc);
 BIni=Pid[Proc][0];Lungh=Pid[Proc][1];
 a=128;
 BiniT=BIni/8;
 offset=BIni-(BiniT*8);
 a=a>>offset;
 j=BiniT;
 for(i=BIni;i<BIni+Lungh;i++)
 {if( (i>BIni)&&(i%8)==0) {j++; a=128;}
  b=255-a;mappa[j]=mappa[j]&b;a=a>>1;
 }
 Pid[Proc][1]=0;
 visualizza(mappa,mem,Pid);
}
```

```
unsigned long CercaBloccoLibero(unsigned char *mappa,unsigned long
mem,unsigned long Lengh)
{unsigned long i,j,Bini,error,LenPar;
 unsigned char a=128;
 error=Bini=j=LenPar=0;
 for(i=0;i<mem&&LenPar<Lengh;i++)
 {if( (i>0)&&(i%8)==0) {j++; a=128;}
  if( (mappa[j]&a)>0)
    {Bini=i+1;LenPar=0;}
  else
    {LenPar++;}
  a=a>>1;
 }
 return(Bini);
}
void visualizza(unsigned char *mappa,unsigned long mem,unsigned
long Pid[][MaxProc])
{unsigned long i,j;
 unsigned long memT=(mem/8)+1;
 for(i=0;i<memT;i++)
 {printf("mapp[%d-%d]",i*8,(i*8)+7);
  unsigned char a=128;
  for(j=0;j<=7;j++)
```

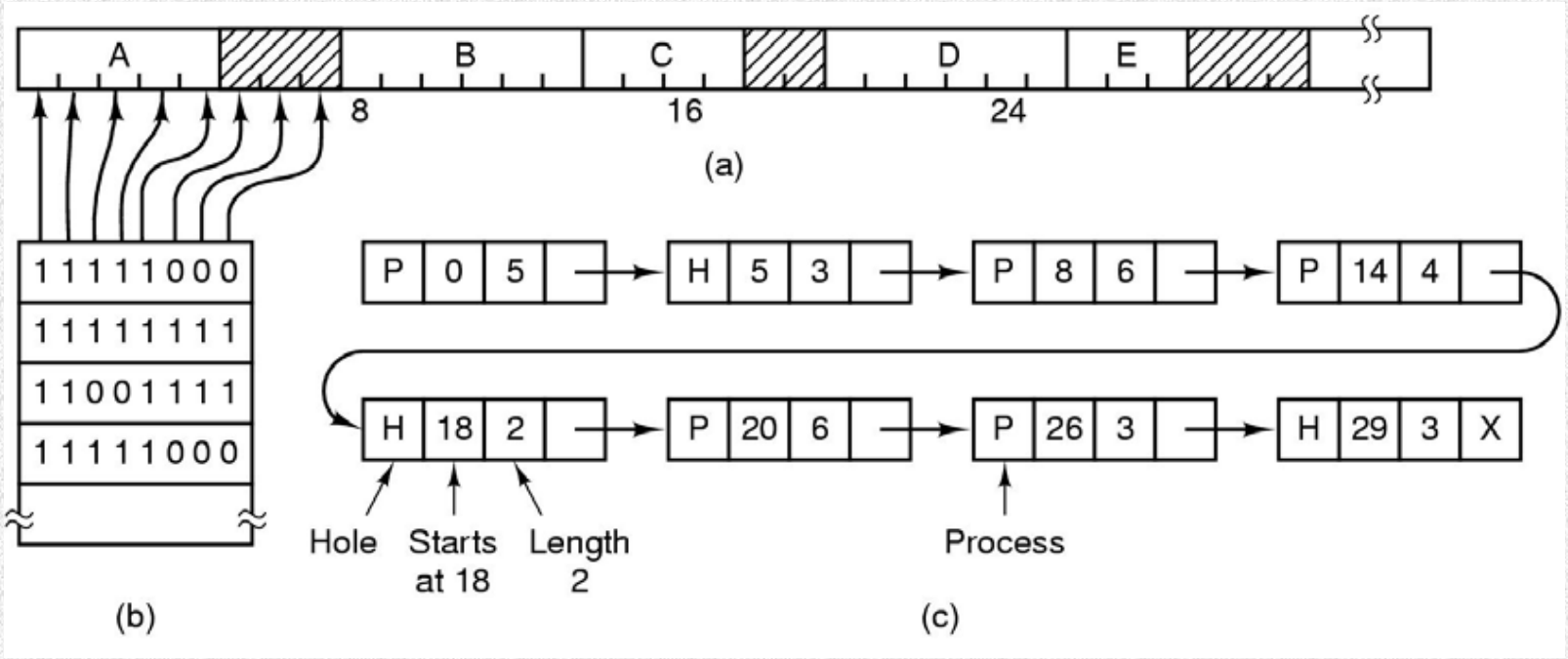
```
  { if( (mappa[i]&a)>0) printf("1");
    else printf("0");
      a=a>>1;
  }
  printf("\n");
 }
 for(i=0;i<MaxProc;i++)
  printf(" Pid[%d]=%d,%d \n",i,Pid[i][0],Pid[i][1]);
 printf("\n");

}
```

Lista di puntatori (singola lista)

Vediamo adesso una possibile soluzione di gestione della memoria con una lista di puntatori.

Come detto si mantiene una lista dei blocchi allocati e liberi di memoria. Ogni elemento della lista specifica consiste:

- 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



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

Una delle possibili soluzioni in C è la seguente

```
#include <stdio.h>
#include <malloc.h>
typedef struct PROC_CODA
{unsigned long IP;
 unsigned long BIni;
 unsigned long BEnd;
 PROC_CODA *next;
} proc_coda;

void Inserisci(proc_coda *prc,unsigned long mem,unsigned
long pid);
void Elimina(proc_coda *prc);
void visualizza(proc_coda *prc);

int main(int argc, char* argv[])
{proc_coda *prcod;
 unsigned long mem;
 unsigned long pid=1;
 printf("memoria disponibile :");
 scanf("%d",&mem);
 prcod=(proc_coda*)malloc(sizeof(PROC_CODA));
```

```
 prcod->IP=0;prcod->BIni=0;
 prcod->BEnd=mem-1;prcod->next=NULL;
 char scelta[2];
 do{do{printf("Per inserire: I\nPer eliminare: E\nPer
Uscire : s\nScelta:");
    scanf("%s",scelta);
if((scelta[0]!='I')&&(scelta[0]!='E')&&(scelta[0]!='s'))
printf("Scelta non consentita\n\n");
       }
while( (scelta[0]!='I') && (scelta[0]!='E') &&
(scelta[0]!='s'));
     if(scelta[0]=='I') Inserisci(prcod,mem,pid++);
     if(scelta[0]=='E') Elimina(prcod);
 }while(scelta[0]!='s');
 return 0;

}

void Inserisci(proc_coda *prc,unsigned long mem,unsigned
long pid)
{unsigned long BEnd;
 unsigned long Lungh;
 char trovato=0;
```

```
proc_coda *P_prc,*P1_prc;
printf("Dai la memoria occupata dal Processo:");
scanf("%d",&Lungh);
if( (Lungh>mem))
  printf("Lunghezza Blocco non consentito!!!!\n");
else
{ P_prc=prc;//[0,0,300]->NULL Lungh=30
  while((P_prc!=NULL)&& (trovato==0))
  {if((P_prc->IP==0)&&((P_prc->BEnd-P_prc>BIni+1)>=Lungh))
    trovato=1;
  else
   P_prc=P_prc->next;
  }
 if(trovato==1)
 {P_prc->IP=pid;//[1,0,300]->NULL
  BEnd=P_prc->BIni+Lungh-1;//BEnd=0+30-1=29
  if((P_prc->BEnd-BEnd)>0)
   {P1_prc=(proc_coda *)malloc(sizeof(PROC_CODA));
    P1_prc->IP=0;
    //[1,0,300]->NULL [0,x,y]->z
    P1_prc->next=P_prc->next;
    //[1,0,300]->NULL[0,x,y]->NULL
    P_prc->next=P1_prc;
```

```
      //[1,0,300]->[0,x,y]->NULL
      P1_prc->BIni=BEnd+1;
      //[1,0,300]->[0,30,y]->NULL
      P1_prc->BEnd=P_prc->BEnd;
      //[1,0,300]->[0,30,300]->NULL
       P_prc->BEnd=BEnd;
      //[1,0,29]->[0,30,300]->NULL
      }

   }
   else
    printf("Lunghezza Blocco non consentito!!!!\n");
 }
 visualizza(prc);
}



void Elimina(proc_coda *prc)
{unsigned long ip;
 printf("Dai IP del Processo da Elimanare:");
 scanf("%d",&ip);
```

```
proc_coda *P_prc,*P_Prec_prc,*P_Next_prc;
P_prc=prc;P_Prec_prc=NULL;
char trovato=0;
while( (P_prc!=NULL)&&(trovato==0))
  {if(ip==P_prc->IP) trovato=1;
   else {P_Prec_prc=P_prc;P_prc=P_prc->next;}
  }
if(trovato)
   {P_prc->IP=0;
   if(P_prc->next!=NULL)  // non sono l'ultimo
    {if(P_prc->next->IP==0) //il prossimo è un buco
        {P_prc->BEnd=P_prc->next->BEnd;
         P_Next_prc=P_prc->next;
         P_prc->next=P_Next_prc->next;
         free(P_prc->next);
        }
    }
   if(P_Prec_prc!=NULL)//non sono il primo
    {if(P_Prec_prc->IP==0)//il precedente è un buco
      {P_Prec_prc->BEnd=P_prc->BEnd;
       P_Prec_prc->next=P_prc->next;
      free(P_prc);
      }
```

```
        }
    }//if trovato
 else
    printf("Processo non Trovato\n");
 visualizza(prc);
}

void visualizza(proc_coda *prc)
{proc_coda *P_prc;
 P_prc=prc;
 while(P_prc!=NULL)
 {printf("[%d|%d|%d]->",P_prc->IP,P_prc->BIni,P_prc->BEnd);
  P_prc=P_prc->next;
 }
 printf("NULL\n");
}
```