Liste di puntatori (doppia lista)

Vediamo adesso una possibile soluzione con una doppia lista di puntatori, una dei blocchi allocati e una di blocchi liberi.

I campi della coda dei Processi sono:

- IP unsigned long identificatore unico del processo>0
- Bini unsigned long blocco iniziale della memoria
- BEnd unsigned long blocco finale della memoria
- Next link al blocco successivo

I campi della coda dei Blocchi Liberi sono:

- 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;

typedef struct MEM_CODA
{unsigned long BIni;
 unsigned long BEnd;
 MEM_CODA *next;
} mem_coda;


void Inserisci(proc_coda *prc, mem_coda *memc,unsigned long
mem,unsigned long pid);
unsigned long CercaBloccoLiberoFirstFit(mem_coda *memcun,unsigned long
Len,unsigned long mem);
void Elimina(proc_coda *prc,mem_coda *memc);
void visualizza(proc_coda *prc,mem_coda *memc);
```

```
int main(int argc, char* argv[])
{proc_coda *prcod;
 mem_coda *memcod;
 unsigned long mem;
 unsigned long pid=1;
 printf("dai la quantità di memoria disponibile :");
 scanf("%d",&mem);
 memcod=(mem_coda *)malloc(sizeof(MEM_CODA));
 memcod->BIni=0;memcod->BEnd=mem-1;
 memcod->next=NULL;
 prcod=(proc_coda *)malloc(sizeof(PROC_CODA));
 prcod->IP=0;
 char scelta[2];
 do{
    do{printf("Per inserire  un processo nella lista : I\nPer
eliminare il processo dalla lista : 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,memcod,mem,pid++);
     if(scelta[0]=='E') Elimina(prcod,memcod);
 }while(scelta[0]!='s');
 return 0;

}
```

```
void Inserisci(proc_coda *prc,mem_coda *memc,unsigned long mem,unsigned
long pid)
{unsigned long Bini;
 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 di memoria non consentito!!!!\n");
 else
 { Bini=CercaBloccoLiberoFirstFit(memc,Lungh,mem);
   if(Bini!=mem)
    {P_prc=prc;//processo messo in coda
     if(P_prc->IP==0) P1_prc=P_prc;
     else
      {while(P_prc->next!=NULL) P_prc=P_prc->next;
        P1_prc=(proc_coda *)malloc(sizeof(PROC_CODA));
       P_prc->next=P1_prc;
      }
     P1_prc->IP=pid;P1_prc->next=NULL;
     P1_prc->BIni=Bini;
    P1_prc->BEnd=Bini+Lungh-1;
    }
   else printf("Lunghezza Blocco di memoria non consentito!!!!\n");
 }
 visualizza(prc,memc);
}
//Funzione FirstFit trova il Blocco iniziale con la strategia FirstFit
```

```
unsigned long CercaBloccoLiberoFirstFit(mem_coda *memcun,unsigned long
Len,unsigned long mem)
{mem_coda *P_memcod,*P_Prec_memcod,*P_Next_memcod;
 char trovato=0;
 unsigned long Bini;
 P_memcod= memcun;P_Prec_memcod=NULL;
 while( (P_memcod!=NULL) && (trovato==0))
   {if( (P_memcod->BEnd-P_memcod->BIni)+1>=Len)
      {Bini= P_memcod->BIni;trovato=1;}
   else
      {P_Prec_memcod=P_memcod;P_memcod=P_memcod->next;}
   }
if(trovato==1)
 { P_memcod->BIni+=Len;
   if(P_memcod->BIni==P_memcod->BEnd)
   //ho esaurito il blocco lo devo cancellare
   {if((P_memcod==memcun)&&(P_memcod->next!=NULL))
    // se sono il primo ma non l'ultimo
    {P_memcod->BIni=P_memcod->next->BIni;
     P_memcod->BEnd=P_memcod->next->BEnd;
     P_Next_memcod=P_memcod->next;
     P_memcod->next=P_Next_memcod->next;
      free(P_Next_memcod);
     }
   else
    {if(P_Prec_memcod!=NULL)
     {P_Prec_memcod->next=P_memcod->next;
      free(P_memcod);
```

```
     }
    }

   }
   return(Bini);

 }
else return(mem);
}

void Elimina(proc_coda *prc,mem_coda *memc)
{unsigned long ip;
 printf("Dai IP del Processo da Eliminare:");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)
    {mem_coda *P_memcod,*P_Prec_memcod,*P_Next_memcod,*P1_memcod;
     P_memcod= memc;P_Prec_memcod=NULL;P_Next_memcod=P_memcod->next;
    while( (P_prc->BEnd>P_memcod->BIni) && (P_Next_memcod!=NULL) )
     {P_Prec_memcod=P_memcod;
      P_memcod=P_memcod->next;
      P_Next_memcod=P_memcod->next;
    }
```

```
     P1_memcod=(mem_coda *)malloc(sizeof(MEM_CODA));
     //P [1,0,60]->[8,61,69]->[11,73,80]->[21,91,100]->[31,101,115]->
     //  [41,116,139]->[71,200,239]->NULL
     //B [70,72]->[81,90]->[140,199]->[240,499]
     // se voglio eliminare P=8 P_memcod=[70,72],P_Next_memcod=[81,90]
     // se voglio eliminare P=21
P_memcod=[140,199],P_Next_memcod=[240,499]
     P1_memcod->BIni=P_memcod->BIni;P1_memcod->BEnd=P_memcod->BEnd;
      //P=8 P1_memcod=[70,72] P=21 P1_memcod=[140,199]
     P_memcod->BIni=P_prc->BIni;P_memcod->BEnd=P_prc->BEnd;
      //P=8  [61,69]->[81,90]->[140,199]->[240,499]
     //P=21 [70,72]->[81,90]->[91,100]->[240,499]
     P_memcod->next=P1_memcod;P1_memcod->next=P_Next_memcod;
      //P=8  P_memcod=[61,69]->[70,72]->[81,90]->[140,199]->[240,499]
     //P=21 [70,72]->[81,90]->P_memcod=[91,100]->[140,199]->[240,499]
     P_Next_memcod=P1_memcod;
     // Compatto memoria
     //guardo avanti
     if(P_Next_memcod!=NULL)  // non sono l'ultimo
      {if(P_memcod->BEnd==P_Next_memcod->BIni-1)
        {P_memcod->BEnd=P_Next_memcod->BEnd;
         P_memcod->next=P_Next_memcod->next;
         free(P_Next_memcod);
        }
      }
     if(P_Prec_memcod!=NULL)//non sono il primo
      {if(P_Prec_memcod->BEnd==P_memcod->BIni-1)
        {P_Prec_memcod->BEnd=P_memcod->BEnd;
```

```
          P_Prec_memcod->next=P_memcod->next;
          free(P_memcod);
        }
      }

     //
     if(P_Prec_prc!=NULL)
      {P_Prec_prc->next=P_prc->next;
       free(P_prc);
      }
     else
     {P_Next_prc=P_prc->next;
      P_prc->IP=P_Next_prc->IP;
      P_prc->BIni=P_Next_prc->BIni;
      P_prc->BEnd=P_Next_prc->BEnd;
      P_prc->next=P_Next_prc->next;
        free(P_Next_prc);
     }
    }//if trovato
 else
    printf("Processo non Trovato\n");
 visualizza(prc,memc);
}

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