

```

// *****
// ModalitàDiServizio:
// - Definisce priorità e quanto di tempo di un processo
// *****
typedef struct
{
    int indicePriorità;
    int deltaT;
} ModalitàDiServizio;

// *****
// DescrittoreProcesso
// - Definisce tutti i parametri e il contesto di un processo
// *****
typedef struct
{
    // Nome univoco (PID)
    int nome;

    // Modalità di servizio
    ModalitàDiServizio servizio;

    // Contesto
    TipoContesto contesto;

    // Successivo processo nella coda
    int successivo;
} DescrittoreProcesso;

// *****
// descrittori
// - Insieme dei descrittori
// *****
DescrittoreProcesso descrittori[NUM_MAX_PROCESSI];

// *****
// Code dei processi pronti (con priorità)
// *****
typedef struct
{
    int primo;
    int ultimo;
} DescrittoreCoda;

typedef DescrittoreCoda CodaALivelli[NUM_PRIORITA];

CodaALivelli codaProcessiPronti;

// *****
// Funzioni di gestione delle code
// *****
void Inserimento (int P, DescrittoreCoda C)
{
    int ultimo = C.ultimo;
    ultimo.successivo = P;
    C.ultimo = P;
    P.ultimo = NULL;
}

```

```

int Prelievo (DescrittoreCoda C)
{
    int primo = C.primo;
    C.primo = C.primo.successivo;
    return primo;
}

// *****
// Processo attualmente in esecuzione sulla CPU
// *****
int processoInEsecuzione;

// *****
// Funzioni per il cambio di contesto
// *****
void SalvataggioStato()
{
    int attuale = processoInEsecuzione;
    descrittori[attuale].contesto = << REGISTRI CPU >>;
}

void RipristinoStato()
{
    int attuale = processoInEsecuzione;
    << REGISTRI CPU >> = descrittori[attuale].contesto;
}

void AssegnazioneCPU()
{
    for (int k = 0; codaProcessiPronti[k].primo == -1; ++k);
    int prossimo = Prelievo(codaProcessiPronti[k]);
    processoInEsecuzione = prossimo;
}

void CambioContesto()
{
    SalvataggioStato();
    int attuale = processoInEsecuzione;
    int priorità = descrittori[attuale].servizio.priorità;
    Inserimento(attuale, codaProcessiPronti[priorità]);
    AssegnazioneCPU();
    RipristinoStato();
}

// *****
// Semafori: definizione e primitive P e V
// - Caso monoprocesso
// *****
// Descrittore del semaforo
typedef struct
{
    int contatore;
    CodaALivelli coda;
} DescrittoreSemaforo;

// Insieme dei semafori
DescrittoreSemaforo semafori[NUM_MAX_SEMAFORI];

```

```

// Ogni semaforo è identificato dal suo indice nell'insieme
typedef int Semaforo;

// Operazione P
void P (Semaforo s)
{
    if (semafori[s].contatore == 0)
    {
        SalvataggioStato();
        int attuale = processoInEsecuzione;
        int priorità = descrittori[attuale].servizio.priorità;
        Inserimento(attuale, semafori[s].coda[priorità]);
        AssegnazioneCPU();
        RipristinoStato();
    }
    else
        semafori[s].contatore--;
}

// Operazione V
void V (Semaforo s)
{
    // Primo processo pronto in coda (la coda può essere vuota, mentre in quella
    // dei processi pronti c'è sempre almeno dummy!
    for (int k = 0; semafori[s].coda[k].primo == -1 && k < MIN_PRIORITA; ++k);
    if (semafori[s].coda[k].primo != -1)
    {
        // C'è un processo in attesa
        int primo = Prelievo (semafori[s].coda[k]);
        int attuale = processoInEsecuzione;
        int prioritàAttuale = descrittori[attuale].servizio.priorità;

        if (prioritàAttuale >= k)
        {
            // Valore maggiore = priorità minore?
            Inserimento(primo, codaProcessiPronti[k]);
        }
        else
        {
            SalvataggioStato();
            Inserimento(attuale, codaProcessiPronti[prioritàAttuale]);
            processoInEsecuzione = k;
            RipristinoStato();
        }
    }
    else
    {
        // Non ci sono processi in attesa
        semafori[s].contatore++;
    }
}

// *****
// Semafori: definizione e primitive P e V
// - Caso multiprocessore con canale di comunicazione
// *****
void P (Semaforo s)

```

```

{
  if (<< SEMAFORO DELLA PROPRIA MEMORIA PRIVATA >>)
    << come nel caso monoprocesore >>
  else
  {
    lock(x);
    << Eseguo la P con eventuale sospensione del RAPPRESENTANTE di P nella coda >>
    unlock(x);
  }
}

void V (Semaforo s)
{
  if (<< SEMAFORO DELLA PROPRIA MEMORIA PRIVATA >>)
    << come nel caso monoprocesore >>;
  else
  {
    lock(x);
    if (<< coda del semaforo non vuota >>)
    {
      if (<< processo appartiene al nodo del segnalante >>)
        << come nel caso monoprocesore >>;
      else
      {
        << eliminazione del processo dalla coda >>
        << si determina area di comunicazione con il nodo a cui il processo appartiene >>;
        if (<< area occupata >>)
        {
          << attendo >>
        }
        else
        {
          << si inserisce in area identificatore del processo riattivato e si pone
          indicatore = 1 >>;
          << si invia interrupt al nodo cui appartiene il processo >>;
        }
      }
    }
  }
  else
  {
    << incrementa contatore del semaforo >>
  }
  unlock(x);
}
}

```