Esercizio E1.2

Impostazione

1. Quali tipi di dati?

   - la struttura dati `buffer` utilizzata come supporto per l'invio di messaggi e risposte. La struttura contiene due campi uno di tipo `T` per contenere l'informazione relativa al messaggio o alla risposta, uno di tipo `int` destinata a contenere l'identiicatore (`PID`) del processo che invia l'informazione;
   - un tipo astratto, `gestore`, tipo dell'oggetto `gestore_pool` che, condiviso tra tutti i processi, alloca ai richiedenti mediante le funzioni `richiesta` e `rilascio` le `B` istanze del tipo buffer;
   - un tipo astratto `coda_FIFO`, che costituisce una coda di indici di `buffer` e di cui verranno create `N_proc` istanze condivise, una istanza per ogni processo. Un processo P invia un messaggio al processo Q inserendo l'indice del `buffer` contenente il messaggio nella `coda_FIFO` associata a Q tramite la funzione `inserimento`. Il processo Q può estrarre il `buffer` contenente il messaggio dalla `coda_FIFO` tramite la funzione `estrazione`.

2 Quali stutture dati?

   - l'array `pool` di `B` istanze del tipo `buffer` che costituiscono le risorse allocate tramite il `gestore_pool`;
   - l'oggetto `gestore_pool`, istanza del tipo astratto `gestore`
   - l'array `canali` di `N_proc` istanze del tipo astratto `coda_FIFO`
   - l'array `attesa` di semafori *evento*, uno per ogni `buffer` del `pool`; ogni semaforo viene usato per sincronizzare un processo mittente in attesa che arrivi la risposta tramite lo stesso `buffer` usato per inviare il messaggio.

3 Le funzioni

```
int send_message(int mit, int ric, T messaggio){
  <richiesta di un buffer al gestore_pool che restituisce l'indice i del buffer allocato o
        blocca il processo se non ci sono buffer disponibili>;
  <inserimento nell'iesimo buffer del pool di messaggio e di mit>;
  <inserimento dell'indice i del buffer con il messaggio nella coda_FIFO di ric>;
  <attesa della risposta sul semaforo evento attesa[i]>;
}

int wait_message(int &mit, T &messaggio) {
  <estrazione dalla coda_FIFO associata al processo in esecuzione dell'indice i del primo buffer
        presente o blocco del processo se non ci sono buffer in coda>;
  <estrazione dal buffer del messaggio e del nome mit del mittente>;
  <l'indice i del buffer viene restituito dalla funzione>;
}

void send_answer(T risposta, int indice_buffer) {
  <inserimento nel buffer del pool il cui indice è indice_buffer della risposta. Il
        processo, nel chiamare questa funzione, le passa come indice_buffer l'intero
        precedentemente restituito da wait_message>;
  <sveglia il processo in attesa della risposta segnalando il semaforo attesa[indice_buffer]
        >;
```

```
T wait_answer(int indice_buffer) {
  <blocco del processo sul semaforo attesa[indice_buffer] >;
  <una volta svegliato, il processo estrae dal buffer del pool di indice indice_buffer la
        risposta che viene poi restituita >;
  <il buffer del pool di indice indice_buffer viene quindi restituito al gestore >;
}
```

Soluzione

```
/* Definizione della struttura buffer*/
typedef struct {
   T mes;
   int mittente;
} buffer

/* Dichiarazione del pool di buffer*/
buffer pool[B];

/* Definizione del tipo astratto gestore*/
class gestore {
   boolean libero[B]={true, …, true};
   semaphore mutex=1; /* semaforo di mutua esclusione*/
   semaphore disponibili=B; /* semaforo risorsa*/

   public int richiesta() {
      int i=0;
      P(disponibili); /*blocca il richiedente quando non ci sono buffer disponibili*/
      P(mutex);
      while(!libero[i]) i++:
      libero[i]=false;
      V(mutex);
      return i;
   }

   public void rilascio(int b) {
      P(mutex);
      libero[b]=true;
      V(mutex);
      V(disponibili); /*sveglia un eventuale richiedente in attesa*/
   }
}

/* Dichiarazione del gestore del pool di buffer*/
gestore  gestore_pool;

/* Definizione del tipo astratto coda_FIFO*/
class coda_FIFO {
   int coda[B]; /*array di interi in cui vengono inseriti gli indici dei buffer contenenti i messaggi
        inviati tramite questa coda. Essendo l'array di dimensione pari al numero dei buffer del pool,
        la coda non può mai essere trovata piena da un processo mittente*/
   int primo=0, ultimo=0;  /*indici per operare sulla coda*/
   semaphore mutex=1; /* semaforo di mutua esclusione fra i processi che inseriscono*/
   semaphore pieno=0; /* semaforo risorsa; il valore coincide col numero di elementi in coda*/

   /* funzione per inserire l'indice di un buffer nella coda */
```

```
    public void inserimento(int indice_buffer) {
        /* come sopra indicato il processo chiamante non può mai bloccarsi per coda piena*/
        P(mutex);
        coda[ultimo]=indice_buffer;
        ultimo=(ultimo+1)%B;
        V(mutex);
        V(pieno); /*segnala che in coda c'è un elementom in più*/
    }

    /* funzione per estrarre l'indice di un buffer dalla coda  e restituirlo*/
    public int estrazione() {
        int b;
        P(pieno); /*blocca il processo se non ci sono elementi in coda*/
        /*la mutua esclusione non è necessaria essendoci un solo ricevente e potendo consentire inserimenti
                e estrazioni concorrenti in quanto la coda è realizzata mediante un array circolare*/
        b=coda[primo];
        primo=(primo+1)%B;
        return b;
    }
}

/* Dichiarazione dell'array canali*/
coda_FIFO canali[N_Proc];


/* Dichiarazione dell'array dei semafori evento attesa */
semaphore attesa[B]= {0, 0, ….., 0};


/* funzione di invio messaggio */
int send_message(int mit, int ric, T messaggio){
    int indice_buffer;
    indice_buffer=gestore_pool.richiesta(); /*il processo si blocca qui se il gestore non ha
                buffer disponibili*/
    pool[indice_buffer].mes=messaggio; /*inserimento del messaggio nel buffer*/
    pool[indice_buffer].mittente=mit; /*inserimento del mittente nel buffer*/
    canali[ric].inserimento(indice_buffer); /*inserimento dell'indice del buffer nella
                coda_FIFO del ricevente*/
    return indice_buffer; /*l'indice del buffer utilizzato viene restituito*/
}


/* funzione di ricezione messaggio */
int wait_message(int &mit, T &messaggio) {
    int indice_buffer;
    int ricevente=PIE()/*il ricevente coincide col processo in esecuzione */
    indice_buffer=canali[ric].estrazione(); /*il processo si blocca qui se la coda è
                vuota*/
    messaggio=pool[indice_buffer].mes; /*estrazione del messaggio dal buffer*/
    mit=pool[indice_buffer].mittente; /* estrazione del mittente dal buffer*/
    return indice_buffer; /*l'indice del buffer utilizzato viene restituito*/
}


/* funzione di invio risposta */
void send_answer(T risposta, int indice_buffer) {
    pool[indice_buffer].mes=risposta; /*inserimento della risposta nel buffer*/
```

```
    V(attesa[[indice_buffer]] /*segnalazione al mittente che la  risposta è pronta*/
}

/* funzione di attesa risposta */
T wait_answer(int indice_buffer) {
    T risposta;
    P(attesa[[indice_buffer]] /*attesa che la  risposta sia pronta*/
    risposta=pool[indice_buffer].mes; /*estrazione della  risposta  dal buffer*/
    gestore_pool.rilascio(indice_buffer;  /*il buffer viene restituito al gestore*/
    return risposta; /* la risposta viene restituita*/
}
```