Cognome ..... *Nome* ..... *Matr.* .....

Lo studente legga attentamente il testo e produca il programma, il makefile, ed i casi di test necessari per dimostrarne il funzionamento. La mancata compilazione dell'elaborato, la compilazione con errori o l'esecuzione errata del programma daranno luogo alla valutazione come ***prova non superata****. Ricordarsi di indicare Nome, Cognome e matricola su questo stesso foglio, che dovrà essere in ogni caso consegnato alla Commissione. Al termine della prova lo studente dovrà fare verificare il funzionamento del programma ad un membro della Commissione.*

Testo della prova

Si realizzi in linguaggio C/C++ lo schema **produttore-consumatore multi-processo** basata su **monitor** e **shared memory**. L'applicazione dovrà simulare l'attività di un ufficio postale, con N processi *cliente* ed 1 processo *sportellista.* I processi cliente sono differenziati in clienti che effettuano operazioni di tipo postale (*cliente_post*), e clienti che effettuano operazioni di tipo finanziario (*cliente_fin*). Lo schema **produttore-consumatore** da seguire è quello con **vettore circolare di buffer di tipo intero** (dim=3), gestito con due puntatori logici *testa* e *coda*. Come nel classico problema del produttore-consumatore, i produttori di entrambi i tipi non possono produrre finché non vi è un buffer disponibile. Inoltre, è necessario implementare il seguente vincolo: **un produttore *client_post* può produrre solo se non vi è alcun produttore *client_fin* in attesa di un buffer libero**. La sincronizzazione tra i processi deve essere basata sul costrutto **monitor** (con semantica **signal-and-continue**), introducendo un opportuno numero di *condition variables* per gestire i vincoli del problema.

Utilizzare i seguenti prototipi per lo sviluppo del codice:

```
typedef struct{
        int job_id[3]; //buffer
        int testa; int coda;
        // ... inserire qui le variabili per la sincronizzazione ...
}ClientiSportellistaMon;

void inizializza_ClientiSportellistiMon (ClientiSportellistaMon *p);
void produci_cliente_postale (ClientiSportellistaMon *p);
void produci_cliente_finanziario (ClientiSportellistaMon *p);
void consuma_job (ClientiSportellistaMon *p);
void rimuovi_ClientiSportellistiMon (ClientiSportellistaMon *p);
```

Il programma dovrà istanziare N=4 processi clienti, di cui **1 *cliente_fin*** e **3 *cliente_post***; inoltre, bisognerà istanziare **1 processo sportellista**. Il processo *cliente_fin* invoca per 4 volte il metodo `produci_cliente_finanziario` che produrrà un valore intero casuale tra 1 e 50 per l'identificativo del job da eseguire; inoltre, il processo attende 4 secondo tra una produzione e la successiva. I processi *cliente_post* invocano per 4 volte il metodo `produci_cliente_postale` producendo sempre un intero casuale tra 50 e 100, ma il processo attende 2 secondi tra una produzione e la successiva. Il processo *sportellista* invoca il metodo *consuma_job* per 12 volte attendendo 1 secondo tra le invocazioni. Il programma principale attende la terminazione dei processi figli e termina a sua volta.